EPFL

Thèse n° 10 505

Green Cryptography and Other Optimisations

Présentée le 29 septembre 2023

Faculté informatique et communications
Laboratoire de sécurité et de cryptographie
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Andrea Felice CAFORIO

Acceptée sur proposition du jury

Prof. C. González Troncoso, présidente du jury
Prof. S. Vaudenay, directeur de thèse
Prof. C. Boura, rapporteuse
Prof. S. Lucks, rapporteur
Dr M. Stojilovic, rapporteuse

École
polytechnique
fédérale
de Lausanne

2023

À M^2, Bellegueule et Reims

Abstract

The spectral decomposition of cryptography into its life-giving components yields an interlaced network of tangential and orthogonal disciplines that are nonetheless invariably grounded by the same denominator: their implementation on commodity computing platforms where efficiency is the overarching dogma. The term efficiency, however, only vaguely captures the intricacies of the field of cryptographic optimisation and can be gauged only in relation to the underlying architectures and their corresponding metrics. In software, these criteria come in the form of memory or instruction cycles of minimisation. Whereas in hardware environments, designers commonly target circuit area or latency reductions.

In this thesis, we blissfully ignore the software realm and fully concentrate our efforts on cryptographic hardware implementations, i.e., application-specific integrated circuits, in an undertaking that encompasses endeavours ranging from classic optimisation work of existing algorithms to the conception of novel constructions. This thesis unfolds over two books:

The first book is a treatise on the energy consumption of cryptographic circuits, an underrepresented metric in the canon of optimisation literature. We commence by devising an energy model for authenticated encryption schemes by investigating the consumptive behaviour of lightweight schemes that are bootstrapped via block ciphers. We then turn our gazes over to hardware-based stream ciphers and propose the first heuristic energy model for this class of algorithms that enables us to design the currently most energy-efficient stream cipher suited for the encryption of larger bulks of data. We conclude this section with the proposal of an energy-efficient small-state stream cipher.

The second book gathers contributions in various other disciplines such as serialisation of block cipher circuits through which we obtain the smallest known implementation of the Advanced Encryption Standard. We then divert our attention toward encryption algorithms for high-throughput networks, as found in the upcoming 6G telecommunication channels. And we design an authenticated encryption scheme that is both secure in the post-quantum setting and reaches unparalleled throughput rates in the Terabit range. Ultimately, the thesis is concluded with an optimisation work on a side-channel-protected threshold implementation of a lightweight family of block ciphers.

Kurzbeschreibung

Ein grober Querschnitt durch die Gefilden der kryptographischen Wissenschaft offenbart ein verqueres Netz von ineinander verwobenen Sparten und Disziplinen, deren gemeinsamer Nenner in ihrer Verwirklichung auf gängigen Rechenplattformen zu finden ist. Das Hauptaugenmerk in dieser Angelegenheit liegt dabei in der Effizienz der Implementationen, was hinsichtlich der mannigfaltigen Welt der Computersysteme nur im Verhältnis zu konkreten Optimierungsmetriken verstanden werden kann. So stehen die Minimierung des belegten Arbeitsspeicher und des Befehlssatzes in Softwareprogrammen im Vordergrund während die Reduzierung der Schaltkreisgrösse und der Latenz bei Hardwareprojekten angestrebt werden.

In dieser vorliegenden Doktorarbeit widmen wir uns ausschliesslich dem Diskurs rund um kryptographische Hardware-Algorithmen, mit anderen Worten anwendungsspezifische integrierte Schaltkreise kurz ASIC. Dieses Vorhaben umfasst Bestrebungen in klassischer Optimierung von bestehenden Algorithmen aber auch die Entwicklung von gänzlich neuartigen Konstruktionen. Organisatorisch teilt sich dieses Dokument in zwei Bücher auf.

Das erste Buch ist eine Abhandlung über den Energieverbrauch von kryptographischen Schaltkreisen, das gegenwärtig noch ein eher unangetastetes Feld im Kanon der einschlägigen Optimierungsliteratur ist. Wir beginnen mit einem Entwurf eines Energiemodells für authentifizierte Verschlüsselungsmethoden durch die Untersuchung des Energieverhaltens von Algorithmen, die auf Blockverschlüsselung basieren. Danach richten wir unseren Fokus auf Stromverschlüsselungsmethoden und ersinnen das erste Energiemodell für diese Art von kryptographischen Algorithmen, was wir dann verwenden um den bisher energiesparendsten Stromverschlüsselungsprogramm zu kreieren für die Verschlüsselung grösserer Datenmengen. Diese erste Buch wird beschlossen mit einem neuen energiesparenden Stromverschlüsselungsalgorithmus, der nur eine vergleichsweise kleine Anzahl an Speicherelementen benötigt.

Das zweite Buch bündelt Projekte, die sich mit anderen Optimierungsmetriken befassen unter anderem eine Arbeit über die Serialisierung von Blockverschlüsselungsmethoden in der wir den zurzeit kleinsten AES Schaltkreis präsentieren. In einem zweitem Vorhaben untersuchen wird das Feld der Algorithmen mit hohen Daten-Bandbreiten und gestalten eine authentifizierte Verschlüsselungsmethode, die sicher ist in der Präsenz von Quantenrechnern und über eine Bandbreite verfügt, die über den Terabit-Bereich hinaus reicht. Das Buch wird beendet mit einer Arbeit über die Effizienz von side-channel-gesicherten Algorithmen.

Retrospective

Doctoral studies are an exercise in introspection. Self-exploration propelled by the dilemmas and hardships that make up scientific work. 2019, questionable competence, application, hiring and a paraphrased dialogue.

M^2: It's a bonus?
A: So they say.

Work progressed smoothly in the early months. The implementation tasks imposed by Chapter 3 occupied most hours, however a deadline was in sight. The very first paper submission as a doctoral student; exciting days cut short by the lukewarm reception of our research. An early moody slump or cognitive dissonance as clinicians would call it, misalignment of expectations with reality. No time to despair, preliminary work on Chapter 4 has already begun. It looks promising, this could be a breakthrough result, an entirely new branch within the discipline. Several co-authors were recruited that share the enthusiasm. Hopes were high that these findings will land a respectable venue and make me pass the first-year exam concurrently. Nonchalance set in, however, contrary to expectations, the exam committee was adversarial. Casual humiliation in a dusty, windowless seminar room.

A: I won't survive this hostility, if this exam serves as a pretext for the coming years.
M^2: But you did pass in the end, didn't you?
A: Pity and disinterest.

Perception of time accelerated after that and work settled in a hypnotising stupor. An endless cycle of appeasing anonymous oracles with often orthogonal agendas. Nonetheless, Chapter 5 and Chapter 6 found a publishing house alongside the aforementioned Chapter 4 which had bounced on numerous occasions. In the meantime, the permanent pressure induced by the competitive nature of the conference circus slowly became detrimental to the physical and mental well-being. Doubts began to permeate the daily research efforts, a meandering poison rushing through mind and body, disintegrating a prospective academic career in slow motion.

M^2: Why quit now?
A: There is no point to any of this.
M^2: Bonuses are often inherently pointless.

Collaborations kept me afloat. Chapter 7 and Chapter 9 were written through a laboratory-internal effort while Chapter 8 was an international endeavour. In the midsts of this phase, a supervisor departed from the laboratory. With the principal catalyst of motivation and architect of this doctoral project missing, apathy took over amplifying the uncertainties and doubts. Incinerate this thesis and let its ashes be blown into the ether of history.

M^2: If not for yourself, do it for your professor...

Serge, it will forever be shrouded in mystery as to what persuaded you to put trust in myself and my abilities. The hiring occurred during a period of intense personal volatility and helped me regain some stability. Your boundless knowledge of the cryptographic craft is both mesmerising and frightening but ultimately always inspiring in our journey to become well-equipped cryptographers. LASEC under your leadership is rock of sanity and studiousness embedded in a faculty where capriciousness is the modus operandi.

M^2: For your supervisor...

Subhadeep, this thesis would not have been possible without you. You stood at the inception of every chapter that constitutes this manuscript. Your wit and and passion for the discipline made each project we tackled a joyful adventure from the first vague ideas to the writing of the eventual paper. Your expertise at the intersection of electrical engineering, optimisation and cryptanalysis is a truly rare skill among the guild of cryptographers and will bless many more unassuming students akin to myself.

M^2: For the lab members...

Betül, you introduced me to LASEC with your semester project which instilled an unwavering curiosity towards the field of cryptography. Martine, our laconic morning chats always initiated the day in a light-hearted fashion. Khashayar, the most ingenious and laid-back office mate one could hope for. Daniel, you are genuinely one of the good ones, never let them deviate you from your path. Fatih, the overlapping content of our theses is proof well-functioning collaboration that benefited greatly from your ceaseless aim for details and elegance. Loïs and Bénédikt, you were an important island of Switzerland for me and this so diverse laboratory. Abdullah, your kindness and ardour for the ins and outs of your research illuminate the corridor and its offices. Laurane, your serenity upholds the humanity of this laboratory. Aymeric and Novak, your stint in our team was brief but impactful and was a daily reminder that other cryptography groups can co-exist peacefully with LASEC. Gwangbae, the server infrastructure that you passed along to me was marvelous feat of engineer and demonstrated that theory-heads like us are able to masterfully maintain real-world systems. Boris and Ritam, your presense is and will be solidifying the place of LASEC among the top-tier research institutions on this planet. All the summer interns, semester and Master project students that crossed paths with me, you are and will be the future faces of cryptographic research whether it be as part of LASEC or other laboratories.

M^2: For the collaborators...

Takanori, Willi, Yosuke, Fukang, Ravi, Fukang, Kosei, Kazuhiko, Ognjen, Francesco and many more. You manifest the pleasure that stands at the core of any research project; collaboration. Your inputs, criticisms, encouragements and contributions form a sizeable portion of these doctoral studies and their scholastic value.

M^2: For Zürich...

Nati, Damian, Flavia, Leana, Mascha, Kiki, you are the light of my existence, an unending torrent of love and companionship. Bonfires whose glow pierce any mist no matter how thick and treacherous it appears, guiding a forlorn soul towards calmer waters. There is no me without you and no life without the embrace of your presence. My heart beats to rhythm of our friendship.

M^2: For Lausanne...

Pera, Rafael, Niro, Thierry, Francis, our friendship is cornerstone of my assimilation in this town both professionally and on weekends. Michi, Roman, Iggy, may we remain close as thanks to you I feel at home here.

M^2: For me...

Your guidance and warmth reassemble the scattered parts of my humanness. You taught me to feel and analyse, to cherish and relinquish, to love and be loved. Our campaign traces the path of suffering, pain and redemption through the guise of empathy. My intimate growth is interlocked with our relationship which has endured for so many years now transforming survival into life. Thank you for everything. This thesis is for you.

Contents

1 Introduction

Я человек, я посредине мира,
За мною – мириады инфузорий,
Передо мною мириады звёзд.
Я между ними лёг во весь свой рост –
Два берега связующее море,
Два космоса соединивший мост.
И – Боже мой! – какой-то мотылёк,
Как девочка, смеётся надо мною,
Как золотого шёлка лоскуток.

— *Arseny Tarkovsky, 1958*

Sealed in an ivory tower above the ephemeral ripples of the terrestrial ether, an endless mirage levitates through the chamber where scholars congregate. Their gazes are directed onto fleeting silhouettes that emerge in the disordered haze. These elusive apparitions hold the decipherment of the riddle that locks the hatch that would grant them renewed freedom. Infrequently, a hollow gasp, kindled by the manifestation of an apparent solution, echoes through the room prompting agitated murmurs among the congregation. Per usual, its dismissal is announced by a diminishment of the commotion, and so, the silent sermon continues. An untold decree from a forgotten epoch once bound these humans to this plight. Unperturbed from earthly matters and affairs they are to conduct their intangible research in this secluded residence until an eventual enlightenment will set them free from their obscure assignment. An adage from past Königsberg recounts that self-incurred immaturity is an idiosyncratic property of an individual mind and thus enshrines a quintessential darkness within the understanding of our surroundings. This Kantian interpretation of the banishment passes the blame onto the scholars themselves thus excluding any external infliction. Transitively, this further establishes that the hallowed spectre is birthed from a collective fever dream that ceaselessly nurtures the devoted researchers. Such a metaphysical perpetuum mobile has no defining beginning or end and thus exists for the sole purpose of maintaining this eternal self-insemination of the immaterial scholastic animus. How can one endure such a reality without succumbing to the permanence of this hopeless state. Absurdists would tell us that even Sisyphus extracted some enjoyment out of his task and hence our scholars must sense a similar indulgence albeit their endeavour appears to revolve less repetitively. Transposing this little Gedankenexperiment onto the faction of people that dabble in cryptographic research, it is possible to notice a fair amount of congruence. That is to

say, at its fundament, large swats of the cryptographic undertaking are marked by their self-propellant nature, in which problems exist as vehicles that spawn puzzles infinitely anew. In this setting, being a cryptographer means to bar his professional countenance from worldly issues by choosing to pursue a quest within the secretive mirage that itself came into existence through insistent castigation in the search for novel problems. Contemporary cryptography thus proceeds for its own subsistence in an scientific ivory tower that, similarly to the Tower of Babel, reaches into the stratosphere of an untethered imaginary void.

1.1 Cryptographic Optimisation

Almost defiantly, in this thesis we reject the purported notion of intellectual autarchy by calling upon a different a parable, the Faustian undoing and subsequent incarnation of a replenished scientific fabric. As a Goethean dramatis persona, we offer the purity of our academic sector to a mischievous entity in an attempt to bridge the cleft between the perceived mundaneness of real-world problems and the astral body of modern cryptographic research. This work, in its full breadth, spans a vertical bridge to the unspoiled floors of this discipline. Unlike the aforementioned genius universalis, a holistic approach is not appropriate, hence it is vital that our battles are chosen wisely. Thankfully, anchors to link the bridge are plenty. It would be possible, for example, to imagine a discourse about the applicability of cryptographic protocols in modern communication applications whose design space remains largely uncharted. On another note, the encroaching reality of cloud-based data-processing solutions requires a new kind of primitive that enables efficient yet nonetheless secure computation on data sets that, more often than not, contain delicate personal information. Finally, the looming advent of quantum-powered computing units poses an existential threat to many implemented cryptographic solutions. However, we believe that even these issues reside in a rather hypothetical head space and are remarkably independent from more pressing questions. There is no doubt that the current waste of resources inexorably thrusts the world onto a disaster from which escape seems only possible through an all-encompassing consensus on all levels of politics, industry, and science. Naturally, this must not exclude the field of cryptography that should strive for the generation and adoption of low-resource algorithms that both fulfil the inherent need for security but also impose as little overhead as possible in terms of their physical requirements. In and of itself, this is primarily an exercise in optimizing both existing and novel algorithms and protocols and is not intrinsic to cryptography alone. The cycle of invention, integration, and optimisation is a dictating force in many research and engineering fields. However, given the relatively short gestation period of the currently considered modern cryptography, much of the optimisation venture still lingers in its infancy.[1] In conclusion, permit us to restate the scope of this body of work, commencing from the desire to engage with topic of cryptography from a more grounded perspective, we fully commit ourselves to the study of different optimisation strategies that optimize existing and newly proposed algorithms in such a way as to enable more resource-friendly op-

[1] Note that folkloric usage of cryptographic techniques can be traced back to ancient times throughout different eras and continents, permeating the associated military and political history.

erations. This journey traverses and ultimately extends the pantheon of techniques carving itself a niche in the canon of cryptographic optimisation work.

The bulk of efforts in this field exist in two separate realms, namely software and hardware; both domains comprising several disciplines. In software, algorithms are optimized for the number of instruction cycles that are reflected in the latency. Other targets include code size and memory usage, as they are crucial for resource-constrained devices, such as micro-controllers that often feature limited amounts of read-only and random-access memory. In hardware, we distinguish between enhancements aimed at application-specific integrated circuits (ASIC) or field-programmable gate arrays (FPGA). A design might strive for conciseness, i.e., small logic gate count on integrated circuits, a low number of used slice units on FPGA platforms, a reduced length of the critical path. Other metrics such as the total number of clock cycles or the incurred power/energy consumption are also of considerable interest. The often straightforward nature of algorithms in symmetric cryptography (i.e., block ciphers, stream ciphers, and modes of operation) proves to be an exemplary playing field for optimisation research. In comparison to a run-of-the-mill public-key scheme, symmetric algorithms bear the lion's share of encryption-related computation work hence, by definition, should be as terse as possible. For instance, a public key algorithm can be used once during a communication channel's lifetime to establish shared key material but, subsequently, a corresponding block-cipher might need to encrypt data in the gigabyte size order. Nevertheless, this perceived one-sidedness has been undergoing a subtle transformation in the past years, due to the increased study of post-quantum resistant solutions that has renewed interest in efficient public-key implementations. Apart from a minor excursion into the software performance of a presented construction, this thesis is exclusively centred on the study of techniques bound to the symmetric domain that improves one or more of these optimisation disciplines on ASIC platforms.

The emergence of the field of cryptographic optimisation is inextricably linked to the establishment of the AES block cipher [60] as the standard symmetric encryption solution. Even though various ciphers had been subject to implementation improvements before, it was the wide adoption of AES, in both the academic and industrial settings that instigated a deepened focus on a single algorithm that has become the most studied and dissected construction in cryptography to this day. The relatively straightforward rationale of AES lends itself well to concerted efforts on the circuit level on the individual component functions, primarily the 8-bit non-linear substitution box [51, 103, 115], the matrix multiplication as part of the diffusion layer [22, 94, 99, 101], and full-fledged implementations of the entire encryption module [16, 17, 70].[2] Note that the overall minimisation of an AES circuit in terms of gate area is usually achieved by restricting the data path, thus effectively serializing the design [16, 17, 86, 107] in the hope of avoiding costly replications of individual computation modules over the entire width of the data path. Advances in serialisation techniques have pushed constructions close to a specific point of optimality in which the combinatorial layers that make

[2] The activity of researching optimized AES software implementations on modern x86-64 processors calmed noticeably after the corresponding instruction sets were extended with dedicated AES encryption procedures. However, on low-end 32-bit architectures, the efforts remain animated, see for example [1, 118].

up cipher logic are minimized to such an extent that the resulting circuits consist of barely more than the actual registers that hold the intermediate cipher state and encryption keys.[3] Typically, a narrow data path incurs a hefty latency overhead. For instance, a round-based AES circuit can compute one encryption in ten clock cycles, whereas a bit-serial equivalent requires several hundred cycles [86].

Nonetheless, a stand-alone block cipher usually does not fulfil the security and performance requirements of modern communication protocols hence has to be deployed as part of a mode of operation. Although, the literature has proposed a plethora of such modes, only a handful find an actual integration in existing systems. The most prominent, AES-equipped Galois counter mode (AES-GCM), adhering to the authenticated encryption with associated data (AEAD) paradigm, resides at the core of the TLS suite [104]. True to its name, the Galois counter mode enriches the AES encryption core with a finite field multiplication module over the full data path of 128 bits in order to produce the ciphertext blocks, with the authentication tag. Although, over the course of more than two decades, research has produced remarkably compact AES circuits, multiplication circuits remain an infamous bottleneck in hardware environments, so much so that the efficiency overhead imposed by AES-GCM is prohibitive for many resource-constrained devices that already make up the majority of available computing machinery. This is effectively due to the rise of smart appliances that need to be equipped with micro-controllers or integrated circuits that enable some kind of interconnection with the Internet of Things. IoT gadgets require an equivalent level of security compared to conventional computers. They offer, however, significantly fewer resources in terms of computational power for the implementation of such features. Consider the following illustrative example. A battery-driven implanted pace maker is extended with a communication interface that enables the associated physician in charge to monitor its status remotely. Naturally, such a communication channel needs to be secured adequately, as any malicious tampering could entail lethal consequences. Yet, replacing the battery is a strenuous and costly procedure for the patient hence should be performed only infrequently. This requirement places a burden on the involved cryptographic circuit to be as small and power-efficient as possible. Externally powered circuits found within smartcards and radio-frequency identification (RFID) tags face a similar challenge, with respect to the overhead that is induced by the implemented cryptographic modules, as they usually are constrained to extremely constrained silicon areas. The skyrocketing demand for IoT gadgets and their increasing integration into critical infrastructure thus exacerbates the demand for resource-friendly cryptographic implementations hence poses a serious, nevertheless compelling, conundrum for the involved research community.

1.2 Lightweight Cryptography

It is specifically this demand for efficient cryptography that, paired with the growing confidence in optimizing symmetric algorithms and the obvious shortcomings of AES-GCM, grad-

[3] These serialized block cipher circuits are reminiscent of hardware-oriented stream ciphers whose state-update function is equally marginal compared to the size of the storage elements.

ually gave rise to a new research branch. Today, commonly termed lightweight cryptography, tasked with directing research efforts on the study and design of new constructions, offer improved implementation metrics compared to existing designs and operate at an equivalent security level. For hardware, this normally corresponds to the reduction of an occupied silicon area or the minimisation of the accumulated power/energy consumption of a circuit, during its running time. The AES block cipher provides a fertile breeding ground for new constructions due to the perceived complexities of its S-box and matrix multiplication. The currently smallest circuit-level implementation of the 8-bit substitution box requires roughly 120 logic gates [103]. This module then needs to be replicated sixteen times to cover the entire 128-bit intermediate state, thus yielding a full substitution layer of approximately 2000 logic gates. Similarly, the smallest known implementation of the matrix multiplication needs 92 exclusive-or gates [101], hence it requires four times the amount for the entire diffusion layer. It is self-evident that replacing these two functions with more efficient alternatives would significantly reduce the hardware footprint. In 2007, the first of its kind was the ultra-lightweight design PRESENT [38] featuring a 64-bit state with key sizes of either 80 or 128 bits. It replaced the heavy 8-bit AES S-box with a simple 4-bit substitution. On similar note, its diffusion layer reduces to a bit-permutation that is realised with only wiring on ASIC platforms. These design decisions led to a circuit that undercuts the AES silicon area up to one third, however they come at a price. In order to sustain an adequate resistance against cryptanalytic attacks with component function as simple as in PRESENT, the round function needs be repeated 31 times, compared to the ten rounds in AES, thus increasing its overall latency considerably. This fact highlights a salient point within the lightweight cryptography world in which research is an endeavour in elegantly manoeuvring various trade-offs, a study in finding a sensible middle ground between security and the numerous aspects of efficiency while seeking some form of grace within the proposed improvements. This dogmatic approach is inherent to this branch and stands orthogonal to other domains within cryptography in which proposed solutions have the tendency to become more sophisticated to adapt to increasingly complex requirements. PRESENT was joined by related block ciphers by introducing different improvements. GIFT features both a 64-bit and 128-bit block size with a 128-bit key; this hardens the security guarantees as set by PRESENT while improving the circuit area requirement in the 64-bit block variant and relying on a 4-bit S-box and a bit-permutation layer for diffusion. Instead of completely erasing the matrix multiplication found in AES, we could imagine a more efficient alteration in conjunction with a more concise S-box. This path was taken by the family of tweakable block ciphers SKINNY [30] that, in overall efficiency, is comparable to GIFT. In a different line of work, instead of opting for a small silicon area, a group of related ciphers optimizes the critical path in hardware, specifically the longest period until all output signals of the block-cipher module reach stability upon toggling some input signals. Constructions in this family include PRINCE, MANTIS and QARMA [8, 30, 40]. Recently, the SPEEDY design set a pronounced example by trading a significant area overhead for a reduced latency that currently stands as the most latency-efficient cipher known in the literature [97].

So far, this discussion has revolved entirely around lightweight block ciphers, however in

the stream cipher domain some similar developments have taken place. This is unsurprising, as stream ciphers tend to be designed in an even more stringent simplistic manner, often being composed of no more than a shift register and a Boolean function that injects new bits into the state. Trivium [61] marks a prime example in this line of research, as its state update function can be described in roughly a dozen logic gates. This perceived simplicity of the Trivium algorithm remains unbroken to this day, as there are no cryptanalytic attacks that recover the 80-bit encryption key with a time complexity better than an exhaustive search. A similarly efficient design can be found in Grain-128a; it offers 128-bit security at a the expense of a more complex state update function [2]. Trivium and, by extension Grain-128a, will play an integral part of this thesis.

Again, these lightweight constructions are mostly incomplete without an accompanying mode of operation that elevates their applicability to real-world protocols. For a countermeasure, in 2018, the National Institute of Standards and Technology (NIST) launched the Lightweight Cryptography Standardisation process (LWC) that became the main accelerator for the vessel of lightweight cryptography research. The stated objective of the LWC competition is to identify lightweight AEAD modes of operations and lightweight hash functions, for deployment in resource-constrained devices and solutions.[4] Of the 56 submissions accepted to the competition, ten remained as finalists in the ultimate round, after which Ascon [65] was selected for standardisation. Among the ten finalists, seven are based on permutations that were either specifically devised for the LWC or stem from earlier proposals. Two designs are based on lightweight block ciphers. GIFT-COFB [21] is a straightforward wrapper around the GIFT block cipher in its 128-bit variant and, similarly, Romulus that deploys SKINNY as its encryption core [74]. Finally, the tenth submission, Grain-128AEAD [78], is the sole competitor built around a stream cipher, in this case Grain-128a. It is noteworthy that in the penultimate round of the process, which included 32 candidates, 15 of these were based on block ciphers with a handful even bootstrapping AES in their respective modes of operations.

1.3 Power/Energy Consumption

In their initial call for submissions for the contest, NIST listed silicon area, latency, throughput and power/energy consumptions as synonymous hardware optimisation criteria but ultimately left it to the designers as to which angle they would prioritize. The strategy behind this decision is the facilitation of a diverse set of candidate constructions that would cover the full spectrum of optimisation disciplines. Still, the resulting set of submissions trails behind this goal. A quick glance at the roster of the final round reveals a clear inclination towards circuit area and throughput optimisations, thus often leaving power and energy consumption considerations as an afterthought in the hope that a lean design would somehow reflect positively on power and energy. Instinctively, this not at all a surprising fact, given that the logic gate-count and throughput are discrete metrics whose optimisations have enjoyed the

[4] In parallel to the LWC, NIST held a competition for the standardisation of post-quantum public key schemes (PQC). The scope of this competition was broader in the sense that it aimed at the standardisation of public key encryption schemes, key encapsulation mechanisms, and digital signature designs. As of the second half of 2022, three signature algorithms and one key encapsulation mechanism have reached the standardisation stage.

most attention in the literature hence are, to some extent, well-understood disciplines. Even in the case of power, we could argue that there exists some degree of correlation to the overall silicon area, as larger circuits tend to draw more power than smaller ones. In contrast, the study of the energy consumption of a circuit is a much more opaque exercise that sits at the intersection of all previously mentioned metrics hence exerts a variety of trade-offs in different directions. We could set out to reduce the gate count or increase the throughput, independently in a narrow framework that only considers the metric at hand. Whereas, the improvement of the energy aspects of a circuit can be achieved only through a meticulous understanding of all other metrics in combination with micro-architectural facets imposed by the underlying platforms hosting the cryptographic circuits.

As hinted before, encryption circuits on ASIC platforms that consume a low amount of energy are a crucial backbone of battery-driven machines that can run on an exceedingly small budgets such as medical implants and various kinds of sensory equipment of radio-frequency identification tags. In the realm of semiconductors, power and energy are correlated parameters insofar as energy is the time integral of power, with power representing the rate of energy consumption, i.e., a less energy-hungry cryptographic circuits drains the battery less. For now, this superficial illustration of the intricacies of energy consumption suffices. A more in-depth treatment of the energy thematic is detailed later in the text. The perceived difficulty of dealing with this topic is mirrored in the deferred interest within the community. It took more than a decade, after the standardisation of AES, for the first dedicated study on the energy consumption of cryptographic circuits to surface in the literature. In 2012, in the pioneering work by Kerckhof et al. [89], they implement and compare several lightweight block ciphers on a standard cell library with a particular focal point on their energy-consumption behaviour as a function of different architectural parameters, such as the clock frequency and the voltage. The main insight of their work was the realisation that, when adjusted for the inherent leakage of the underlying cells, the energy consumption of a block cipher circuit is independent of the clock frequency. This observation was later confirmed in a similar paper [27]. The first attempt at designing a block cipher dedicated to low-energy environment came in the form of Midori [13]. In a painstaking trial-and-error process, the authors identified different functions, i.e., S-box and linear diffusion layers. This combination of layers yielded the most energy-efficient design while guaranteeing a resistance to known cryptanalytic attacks. The resulting block cipher, with a key size of 128 bits, is vaguely reminiscent in its structure to later algorithms such as GIFT and SKINNY. Although, the proposal of Midori is a remarkable accomplishment in the lightweight cryptography canon, its rather observational approach to the question of reducing the energy consumption lacked generality in order for it to be applicable to a broader range of block ciphers. Nevertheless, Midori further invigorated the low-energy endeavour by continuing with the proposal of a general energy model for substitution-permutation network (SPN) block ciphers [18]. More specifically, the project investigated a design strategy that is intrinsic to the implementation of block ciphers on ASIC platforms that unrolls the round function. Unrolling determines how many round function executions are executed in a single clock cycle. And the parametrisation of this factor, in conjunction with the particularities of utilized micro-architectural

implementation features, enables the deduction of a quasi-quadratic polynomial equation that predicts, to remarkable precision, the energy consumption of the block-cipher circuit in question. Unrolling is also a sensible strategy when it comes to stream ciphers; this means that, to compute many of them, instead of generating a single update bit per clock cycle, it is possible to replicate the corresponding Boolean functions. This was explored in [25] in the context of the effects on the energy consumption, where the authors highlighted an interesting phenomenon in the form of unexpected efficiency. For example, it is shown that Trivium consumed the least amount of energy in an unrolled fashion for the encryption of larger amounts of data, thus even outclassing block ciphers such as the supposedly energy-efficient Midori design.

1.4 Preview

The first book of this thesis further illuminates the enigma of energy consumption in symmetric encryption circuits. That is to say, over the course of four papers we extended the existing knowledge by investigating the implementation of lightweight AEAD modes of operations that are based on block ciphers and, ultimately, on stream ciphers.

Energy Analysis of Lightweight AEAD Circuits. The inaugurating paper of this document takes the energy model for block-cipher circuits as proposed in [18] and extends its reach to ten lightweight AEAD schemes from the penultimate NIST LWC round that are bootstrapped with block ciphers. The clear intention of this work was to supplement the lacking treatment of energy aspects, within the competition, and to provide useful insights for designers who seek to understand which particular choices incur significant energy consumption in AEAD schemes. Today, our work is the most comprehensive analysis of the consumptive behaviour of cryptographic circuits in the literature. It was presented at the 19th International Conference on Cryptology and Network Security (CANS'2020) and was bestowed with the Best Paper Award.

Perfect Trees: Designing Energy-Optimal Symmetric Encryption Primitives. With the third paper, we shift our focus onto stream ciphers. As mentioned in Section 1.3, a preliminary investigation of the energy aspects in stream ciphers circuits was conducted in [25], a general energy model for this class of ciphers remained at large. For a remedy to this state, we devise the first heuristic energy model in the realm of stream ciphers, as it links the underlying algebraic properties of the state update function to the consumptive behaviour. The model is then used to derive a metric that exhibits a heavy negative correlation with the energy consumption of a broad range of stream-cipher architectures, namely the families of Trivium-like and Grain-like constructions. For this process, we propose several alternative energy-optimal versions of Trivium. Two of these designs, TRIVIUM-LE(F) and TRIVIUM-LE(S), consume approximately 15% and 25% less energy, respectively, hence they are to date the most energy-efficient encryption primitives inheriting the same security as the original Trivium. We also present TRIAD-SC as an energy-efficient variant that satisfies a higher security level. The simplicity and wide applicability of our model has direct consequences for the conception of future hardware-targeted stream ciphers because, for the first time, it is possi-

ble to optimize energy during the design phase. Moreover, we extend the reach of our model beyond plain encryption primitives and propose a novel energy-efficient message authentication code TRIVIUM-LE-MAC. Our work was included the fourth issue of Transactions on Symmetric Cryptology of the year 2021 (IACR-ToSC'2021-4).

Atom: A Stream Cipher with Double-Key Filter. Finally, we will conclude the first book of this thesis with a new energy-efficient stream-cipher proposal designed from a starting point different from the Trivium derivatives of before. It is a well-known fact that, in order to nullify time-memory zrade-off attacks, the internal state of the cipher must be at least twice the size of the secret key [37]. However, in recent years stream ciphers boasting smaller states have begun to appear. These constructions circumvent the state size requirement by continuously mixing the key in the state update function. Algorithms in this class include Sprout, Plantlet and Lizard [6, 75, 105], all of which possess cryptanalytic weaknesses. We propose the stream cipher Atom that offers 128-bit security and a state size of 159 bits; this is only 25% more than the key size. Our design avoids falling into the same pitfalls that plagued the aforementioned trio of short-state ciphers hence is proven secure in the context of all known attacks against similar constructions. The resulting circuit is competitive in both area and power/energy to other stream ciphers, thus making it a viable choice on resource-constrained ASIC platforms. Atom was published in the first issue of Transactions on Symmetric Cryptology of the year 2021 (IACR-ToSC'2021-1).

In the second book of this thesis, we focus on other optimisation aspects that appear to be left by the wayside in the NIST LWC. First, this involves serialisation techniques of block ciphers and of associated modes of operations. A serialized cryptographic circuit reduces the data path width in order to avoid costly replication of modules. Recall the AES S-box that needs to be integrated sixteen times in a straightforward round-based configuration; it computes one round function in one clock cycle. For example, restricting the data path to eight bits would thwart this silicon area overhead by incurring an increased latency. The first byte-serial implementation appeared in 2005 and required more than 1000 clock cycles to compute either one encryption or decryption [70]; it was subsequently improved to 226 cycles in an encryption-only design [107]. This encryption-only limitation was then eliminated by the Atomic-AES proposal that offered the same latency of 226 cycles but featured both encryption and decryption capabilities with an area overhead of roughly 10% [17]. Note that these 226 cycles are composed of sixteen cycles for loading the plaintext into the state register and of 21 cycles for each round function computation. Both constructions possess only a single S-box module that is shared between the round function and the key schedule algorithm. It turned out that restricting the data path to one byte is not optimal in terms of circuit area. This prompted the construction of block cipher circuits whose data paths were reduced to a single bit. The bit-sliding technique is a generic strategy for converting a round-based block cipher circuit into a serialized version with varying data path widths [86]. Through this technique, the authors achieve a bit-serial AES circuit with an encryption latency of 1776 cycles, which improves the area footprint by 30% with respect to byte serial designs as proposed

in [17, 107]. It was similarly applied to SKINNY and PRESENT.[5] Although serialized circuits minimise the occupied silicon area and are thus, by extension, also beneficial in terms of the power consumption, this design choice is sub-optimal for the energy aspect due to the exceedingly increased latency, i.e., in general, a serialized block cipher consumes several times the amount of energy than its round-based counterpart. The bit-serial catalogue was recently extended by the swap-and-rotate work that attempts to implement the bit-permutation layers of PRESENT and 64-bit GIFT, solely by the virtue of a small number of flip-flop pairs swapping two bits in-place at specific points in time during the round function computation while the state bits rotate through the register pipeline, one position per clock cycle [11]. In this setting, the latency of an encryption is parametrised by the number of equipped swaps, i.e., the more swaps the lower the number of clock cycles needed to compute a round function. Consequently, swap-and-rotate enable the construction of the smallest known bit-serial circuits for both PRESENT and GIFT, thus undercutting the bit-slide variant from [86] in both area and latency.

The initial frame of reference of the second book in this manuscript is thus circuit area, which we explore in two papers on the topic of bit-serial implementations followed by a work on throughput maximisation with a proposal of a novel AEAD mode of operation scheme tailored for 5G-and-beyond network telecommunication networks and, ultimately, the thesis is concluded with optimised side-channel implementations.

The Area-Latency Symbiosis: Towards Improved Serial Encryption Circuits. In this part, we delve into the swap-and-rotate thematic by investigating the latency overhead that is induced by such bit-serial circuits. More specifically, we devise implementations that compute one round function in exactly as many cycles as there are state bits while keeping the number of utilized swaps, hence the area overhead, moderate. We achieve this specificity for AES, SKINNY, and the 128-bit version GIFT. We extend it to 4-bit and 8-bit serialized implementations and, ultimately, to four AEAD schemes of the NIST LWC, namely SUNDAE-GIFT, SAEAES, Romulus and SKINNY-AEAD; of these, Romulus competes as a finalist. Our work was presented in the first issue of Transactions on Cryptographic Hardware and Embedded Systems of the year 2021 (IACR-TCHES'2021-1).

A Small GIFT-COFB: Lightweight Bit-Serial Architectures. The second effort applies the swap-and-rotate technique to GIFT-COFB; it was not investigated in the previous chapter. Unlike the aforementioned bit-serial AEAD implementations, GIFT-COFB involves finite field arithmetic for which there is no straightforward mapping into a bit-serial setting that is both circuit area and latency efficient. We fill this gap by proposing three bit-serial circuits that to date stand as the most area-efficient GIFT-COFB implementations known in the literature; they exert various trade-offs in terms of latency and silicon area. Our work was presented at the 13th International Conference on Cryptology in Africa (AFRICACRYPT'2022).

Rocca-S: An Ultra-high Throughput and Quantum-Secure Authenticated Encryption Scheme.

[5] The bit-sliding work is preceded by dedicated block ciphers KATAN and SIMON that are reminiscent of stream ciphers with round functions based on shift registers [28, 62] hence can be efficiently implemented in a bit-serial fashion.

We propose Rocca-S, an AES-based authenticated encryption scheme with a 256-bit key and a 256-bit tag. The scheme is specifically concocted to meet the requirements of eventual 6G cellular communication networks, in terms of performance as well as security. Unlike existing schemes for 5G and 6G, Rocca-S can guarantee 256-bit, as well as 128-bit, security for not only key recovery attacks but also forgery attacks in both the classic and quantum settings, respectively. Nevertheless, this security level Rocca-S achieves throughput rates of more than two Terabits per second, without sacrificing other metrics such as occupied silicon area or power/energy consumption, thus making it a competitive choice that satisfies the requirement of a wide spectrum of environments. Furthermore, in software, Rocca-S attains encryption/decryption rates that cross the 200 Gigabits per second boundaries, when run on recent processor architectures. Our work is included in the proceedings of the European Symposium on Research in Computer Security (ESORICS) of the year 2023 and is concurrently submitted to the 6G standardisation committee hosted by the 3GPP consortium.

We close the principal body of this thesis with a foray into side-channel countermeasures. It is evident that cryptographic research is currently undergoing a golden age of side-channel analysis that permeates all disciplines, from hardware to software and from post-quantum implementations to block cipher circuits. In contrast with cryptanalytic attacks that exploit structural weaknesses of an algorithm, side-channel attacks analyse vulnerabilities and leakages that arise in particular implementations. Most prominently and perhaps calamitous are exploits that infer secret values from power traces of a device; these traces are usually readily obtainable. The destructiveness of such power analysis exploits prompted the establishment of an adjacent research branch tasked with devising countermeasures through which implementations can be hardened against such attacks. The core principle of a countermeasure is the inclusion of randomness in the cipher state by splitting the state into multiple independently masked shares that obscure sensitive values in the power trace. Masking, as an effective countermeasure, first appeared at the beginning of the millennium and has since then gone through countless iterations supported by various security models that target different security levels. In the context of this thesis, we are specifically interested in the optimisation of masking schemes that adhere to the threshold implementation methodology. A threshold circuit of a function randomises only the input shares and gives guarantees against an adversary that is able to probe a certain number of wires within the circuits, even in the eventual presence of glitches.

Improving First-Order Threshold Implementations of SKINNY. Finding efficient threshold circuits with a specific number of shares is an arduous tasks for all but the simplest non-linear Boolean functions. One of these functions is the 8-bit S-box of the SKINNY block cipher. The SKINNY white paper proposed a decomposition of this S-box into four sub-functions of algebraic degree two that could be implemented in a 3-share threshold circuit. This means that the S-box is executed over four cycles, which is a significant latency overhead. In this section, we demonstrate the feasibility of decomposing the SKINNY S-box into three quadratic sub-functions that enable the performance of the substitution over the course of three cycles. In a second step, we propose a decomposition into two cubic sub-functions that enable

an execution in two clock cycles. Our constructions significantly reduce latency and energy consumption per encryption operation. Our work was presented at the 22nd International Conference on Cryptology in India (Indocrypt'2021).

Personal Bibliography. In the following paragraph, we give a comprehensive list (in order of appearance) of all the co-authored publications accomplished in the course of this PhD research. This document aggregates, in an expanded format, the content of the articles marked in bold font. The papers "A Study of Persistent Fault Analysis" and "Energy Analysis of Lightweight AEAD Schemes" were each decorated with the Best Paper Awards at their respective conference.

1. Andrea Caforio and Subhadeep Banik. A Study of Persistent Fault Analysis. In: *Security, Privacy, and Applied Cryptography Engineering - 9th International Conference, SPACE 2019, Gandhinagar, India, December 3-7, 2019, Proceedings*. Ed. by Shivam Bhasin, Avi Mendelson, and Mridul Nandi. Vol. 11947. Lecture Notes in Computer Science. Springer, 2019, pp. 13–33 [46]

2. **Andrea Caforio, Fatih Balli, and Subhadeep Banik. Energy Analysis of Lightweight AEAD Circuits. In: *Cryptology and Network Security - 19th International Conference, CANS 2020, Vienna, Austria, December 14-16, 2020, Proceedings*. Ed. by Stephan Krenn, Haya Shulman, and Serge Vaudenay. Vol. 12579. Lecture Notes in Computer Science. Springer, 2020, pp. 23–42 [43]**

3. **Fatih Balli, Andrea Caforio, and Subhadeep Banik. The Area-Latency Symbiosis: Towards Improved Serial Encryption Circuits. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.1 (2021), pp. 239–278 [9]**

4. Andrea Caforio, Fatih Balli, and Subhadeep Banik. Melting SNOW-V: improved lightweight architectures. In: *J. Cryptogr. Eng.* 12.1 (2022), pp. 53–73 [44]

5. **Subhadeep Banik, Andrea Caforio, Takanori Isobe, Fukang Liu, Willi Meier, Kosei Sakamoto, and Santanu Sarkar. Atom: A Stream Cipher with Double Key Filter. In: *IACR Transactions on Symmetric Cryptology* 2021.1 (2021), pp. 5–36 [20]**

6. Andrea Caforio, F. Betül Durak, and Serge Vaudenay. Beyond Security and Efficiency: On-Demand Ratcheting with Security Awareness. In: *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II*. ed. by Juan Garay. Vol. 12711. Lecture Notes in Computer Science. Virtual Event: Springer, Heidelberg, Germany, May 2021, pp. 649–677 [50]

7. Andrea Caforio, Fatih Balli, Subhadeep Banik, and Francesco Regazzoni. A Deeper Look at the Energy Consumption of Lightweight Block Ciphers. In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*. IEEE, 2021, pp. 170–175 [45]

8. Andrea Caforio, Fatih Balli, and Subhadeep Banik. Complete Practical Side-Channel-Assisted Reverse Engineering of AES-Like Ciphers. In: *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*. Ed. by Vincent Grosso and Thomas Pöppelmann. Vol. 13173. Lecture Notes in Computer Science. Springer, 2021, pp. 97–117 [42]

9. **Andrea Caforio, Daniel Collins, Ognjen Glamocanin, and Subhadeep Banik. Improving First-Order Threshold Implementations of SKINNY. in: *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings*. Ed. by Avishek Adhikari, Ralf Küsters, and Bart Preneel. Vol. 13143. Lecture Notes in Computer Science. Springer, 2021, pp. 246–267 [49]**

10. **Andrea Caforio, Subhadeep Banik, Yosuke Todo, Willi Meier, Takanori Isobe, Fukang Liu, and Bin Zhang. Perfect Trees: Designing Energy-Optimal Symmetric Encryption Primitives. In: *IACR Trans. Symmetric Cryptol.* 2021.4 (2021), pp. 36–73 [47]**

11. **Andrea Caforio, Daniel Collins, Subhadeep Banik, and Francesco Regazzoni. A Small GIFT-COFB: Lightweight Bit-Serial Architectures. In: *Progress in Cryptology - AFRICACRYPT 2022 - 13th International Conference on Cryptology in Africa, Fes, Morocco, July 18-20, 2022, Proceedings*. Ed. by Abderrahmane Nitaj and Lhoussain El Fadil. Vol. 13143. Lecture Notes in Computer Science. Springer, 2022, pp. 246–267 [48]**

12. **Subhadeep Banik, Andrea Caforio, Kazuhide Fukushima, Takanori Isobe, Shisaku Kiyomoto, Fukang Liu, Yuto Nakano, Kosei Sakamoto, Nobuyuki Takeuchi, and Ravi Anand. Rocca-S: Ultra High-Throughput and Quantum-Secure Authenticated Encryption. 2023 [19]**

1.5 Repositories

In a thesis, where most contributions are experimentally backed by implementations, reproducibility reigns supreme. In order to abide by this unwritten rule, which is far too often left by the wayside in the cryptographic literature, we list below the corresponding publicly available repositories corresponding to each chapter in this thesis.

- Chapter 3. We provide the HDL code and test vectors for each of the nine investigated NIST LWC AEAD candidates; they are based on lightweight block ciphers in all their configurations (round-based, unrolled, clock-gated, threshold implementation).

  `http://bit.ly/3wYSdIU`

- Chapter 4. In order to aid reproducibility of the results, we provide the HDL code used for all the experiments and the implementation of all schemes, along with detailed instructions and executions scripts, .

  `http://bit.ly/3DLwZ50`

- Chapter 5. We make available the source code (in VHDL) for the novel and Atom short-state stream cipher, with a software reference implementation for the generation of test vectors.

  `http://bit.ly/3HXAaJk`

- Chapter 6. Publicly available are the complete HDL source code suite for all the proposed serial block ciphers, modes, and operations in all their configuration, as well as a comprehensive collection of correction tests. This repository is hosted as a peer-reviewed TCHES artefact.

  `http://bit.ly/3HZBhbi`

- Chapter 7. We provide the serial HDL implementations for GIFT-COFB-SER-S, GIFT-COFB-SER-F and GIFT-COFB-SER-TI, with a test suite to check their correctness.

  `http://bit.ly/40xptEr`

- Chapter 8. This repository includes the hardware source code and test suite for all proposed Rocca-S configurations (round, unrolled, partially unrolled, byte-serial), as well as the source for AEGIS-256, AES-256-GCM and SNOW-V-GCM. This source constitute the first publicly available hardware implementation for the latter three schemes in the literature.

  `http://bit.ly/3YoZWvs`

- Chapter 9 Lastly, we provide the HDL source code for all proposed threshold implementations of SKINNY, with test vectors and scripts that verify the soundness of the security properties.

  `http://bit.ly/3RDFfd6`

2 Preliminaries

The scientific papers whose amalgamation comprises the bulk of this document were written over a span of multiple years and cover a diverse variety of topics. Although, an overarching cohesive narrative is established in a candid fashion by coupling intersecting high-level domains, it is the disjointness of preliminary material, necessary to grasp the contributions at hand, that can induce an unpleasant cognitive overhead on the reader when presented in a disadvantageous manner. This is especially the case when a chapter that covers a specific article is preceded by its own particular preliminary section. Such a design choice dilutes the cohesiveness of the document and ultimately hurts its readability. Consequently, theses in cryptography have adopted the strategy of prefacing the main body of the text by a comprehensive preliminary chapter that covers key technical concepts succinctly and thus serves as a sort of document-internal encyclopaedic database. The choice of what subjects are to be covered in the preliminary chapter conclusively decides the composition of the readership. An incomplete and superficial treatment results in an exigent text entailing a high expertise among the readers to the point where even intimate familiarity is not sufficient enough to comprehend the conferred material. At the other extreme, preliminaries that indulge in the transcription of too many complexities have an overbearing effect to the point where the actual contributions later in the text appear trivialized and dulled. In our case, this preliminary chapter attempts to find a sensible middle ground that conveys information in a digestible fashion to both experts and the occasional layperson.

As mentioned before, this thesis unfolds over two books that each comprise the description of several scientific articles. Both of them revolve around optimisation aspects of cryptographic implementations in hardware and thus it is only prudent to commence the preliminaries with a concise introduction to Application-Specific Integrated Circuits. For presentation purposes we conduct this survey in a top-down approach starting from actual computing systems to the actual transistors that lie at the fundament of any chip. Naturally, this further covers the relevant optimisation disciplines and metrics associated with cryptographic hardware implementations and additionally the Electronic Design Automation tools used to generate, investigate and measure all presented circuits. Secondly, since all selected papers in this thesis are works that either optimize existing constructions or devise new algorithms by bootstrapping older designs, it is crucial that some of these block ciphers, stream ciphers and modes of operations are introduced properly. A short detour is then taken onto the expla-

nation of generic hardware implementation techniques, in particular the Swap-and-Rotate methodology for reducing the circuit area in serial block cipher circuits and Threshold Implementations for the mitigation of side-channel-based power analysis attacks. However, we commence this preliminary chapter with a summary on common mathematical notation.

2.1 Notation

As we work in the field of hardware cryptography, the basic unit are bits which are designated by lowercase letters x. Bits can be grouped in ordered lists of arbitrary size in order to compose bitstrings. Lists are denoted by uppercase characters; for example a byte is represented by $X = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ with x_0 and x_7 being the most and least significant bit respectively. With one exception in Section 2.6.1, indexing of lists always starts with 0. Enclosing a list variable in vertical bars evaluates its size, e.g., $|X| = 8$ for the byte from before. Furthermore, lists can be rotated to the left by a certain amount of positions k using the $X \lll k$ operator and analogously to the right with the $X \ggg k$. Elements can be arranged in matrices, also denoted by uppercase letter, such as, for instance, a 2×2 bit matrix

$$X = \begin{bmatrix} x_0 & x_1 \\ x_2 & x_3 \end{bmatrix},$$

whose element indices indicate a row-major ordering, i.e., x_0 is the most significant bit followed by x_1. Depending on the application a matrix may also be ordered column-major fashion. In term of operators, we distinguish between bitwise and arithmetic ones:

- *Bitwise.* $\oplus$ represents the XOR function while $\wedge$ is used for the AND operator and $\vee$ for the OR function. A bit is inverted using an overline such that $\overline{x} = x \oplus 1$. These operators can be applied to bitstrings as well where they are bit by bit. As an illustration, the XOR of two 2-bit strings $X = (x_0, x_1)$, $Y = (y_0, y_1)$ is given by

$$X \oplus Y = (x_0 \oplus y_0, x_1 \oplus y_1).$$

- *Arithmetic.* Any other arithmetic operation that does not act on bits, whether it be numbers in $\mathbb{Z}$ or any other algebraic structure, are represented by the schoolbook symbols such as $+$, $\cdot$ and $/$.

2.2 Application-Specific Integrated Circuits

Abstraction of first principles lies at the very core of a computing system. In a reductionist approach it is possible to crystallise the illustrious Turing machine, the hypothetical progenitor of contemporary machinery, out of the assembly of any computer. In this case, the first principles refer to the interpretation of binary data within the intersection of Theoretical Computer Science and Boolean Algebra. The insight that simple Boolean functions acting on one or more bits, given enough time and space, are sufficient to derive any statement that is deemed computable is without any doubt the most transformative upheaval since the

heydays of Renaissance Europe and it is going to shepherd to world into unprecedented territories for decades to come. It is important to note that the defining trademark of modern computer systems is the ability to parse and ultimately execute external, user-defined commands. It is this programmability that unlocks the full unbounded potential of this type of machinery and is conclusively the sole lifeline of the field of Computer Science. It is worth mentioning that the notion of programmability is hardly a concept that came up during the early days of contemporary Computer Science but had been stoking curious minds for much longer. An illustrative example, is Charles Babbage's mechanical Analytical Machine, a concoction that predates all subsequent general-purpose computers.[6]

Although, the perceived powers of a programmable computing machines are undeniable, in an almost paradoxical twist they are often not needed. In a myriad of such occurrences it is more sensible and especially more economical to regress to computers that fulfil a single purpose and are thus application-specific. Any sort of analog-to-digital converter found in any sensor-based installations or more crucially, in the context of this thesis, independent cryptographic circuits are prime instances of application-specific computing. Naturally, any contemporary application-specific computing device is a semiconductor-based electrical circuit whose collection of transistors implement the desired logic. From a high-level perspective, a transistor is simply a contraption that switches an electrical signal composed of an input source, a gate and a output sink where the gate modulates the flow of electricity between source and sink. It is then relatively straightforward to see how individual transistors can be linked in order to compute simple Boolean functions. Furthermore, transistors can be used to create storage elements that retain the value of an input signal over the course of multiple clock cycles. In the remainder, we will denote a single-bit storage unit as a flip-flop and a multi-bit unit composed of multiple flip-flops as a register. In the remainder of this text, we will refer to these units that implement a single Boolean function as logic gates as per the terminology usually used in the literature. Early application-specific integrated circuits were equipped a few thousand logic gates nowadays, depending on the complexity of the application, circuits are composed of potentially billions of interlinked transistors. In summary, an ASIC designates an electronic circuit utilized in an highly individualized domain for which a general computing unit is surplus to requirements. We remark that an application-specific integrated circuit does not preclude a general-computing ability. In fact, the ubiquitous and cheap manner of semiconductor manufacturing allows for the implementation of a full-fledged CPU equipped with both volatile and non-volatile memory on exceedingly small chips. A more precise denotion for such devices is Systems-on-Chip which is beyond the scope of this thesis as it is exclusively concerned with the study and creation of individual cryptographic circuits that each implement a single algorithm.

[6] The Difference Engine was an mechanical calculator and precursor to the Analytical Machine. The earliest archaeological calculator is the Antikythera mechanism dated to approximately 100 BC.

2.3 Hardware Metrics

In the context of optimizing cryptographic hardware algorithms, researchers operate within well-defined disciplines as typically, they do not depart from a generic intentions but attempt to find a angle that allows them improve a specific metric of an algorithm whether the algorithm in question is an existing construction or a new conception. In the following, a handful of hardware metrics are detailed which are covered in all the showcased works. Furthermore, in order to facilitate a minimum degree of comparability between the papers, the majority of obtained measurement results are tabulated for the same set of metrics.

Circuit Area. Instinctively, with any newly devised circuit, the actual size of the network is the figure that first jumps into a designer's eye. It further is the simplest metric to evaluate being the sum total of all logic gates that comprise the full implementation. Depending on the utilized technology, transistors have a certain size and their composition as part of various logic gates occupies a certain area on a chip usually denoted in μm^2. Importantly, this definition of circuit area blissfully ignores the overhead incurred after physically placing and routing the gates, see Section 2.4 for more details. Different technologies vary widely in the amount of area the resultant circuits occupy, as a result, comparing μm^2 figures for the same algorithm in different semiconductor manufacturing processes can yield inconclusive verdicts. For the sake of a more reasonable juxtaposition between technologies, scientific papers usually list the occupied circuit area of investigated constructions in a manufacturing-technology-independent unit. Gate equivalent (GE) denotes the ratio of the total circuit area and the area of a single two-input NAND logic gate ultimately expressing the complexity of the circuit in terms of a single logical unit.[7]

Example 1. *A striking case of a cryptographic work that specifically set out to optimize the logic area of an algorithm below a certain threshold is the eponymous Side-Channel Resistant Crypto for Less than 2,300 GE [112] paper by Poschmann et al. published in 2011 that proposed a bit-serial circuit that implements the PRESENT block cipher [38] in a power-analysis-resistant fashion.*

For the remainder, all tabulated data lists the circuit area metric in both physical units (μm^2) and in more conclusive Gate Equivalents (GE).

Critical Path. Digitalisation is a by-product of software abstraction that camouflages the physical phenomenons inherent to semiconductor technology. Electrical signals, by default, are not switched instantaneously but are subject to delays induced by the underlying composition of the transistors whose capacitors require a certain time frame for being charged or discharged. As the depth of a circuit increases with its complexity so does the delay until a change in an input signal is reflected in stable output signals. By definition, the critical path denotes the longest delay path within a circuit. The reciprocal of the critical path specifies the maximum clock frequency allowed to run the device without violating the correctness properties of the design, i.e., the smaller the critical path the faster can the circuit be clocked. For

[7] The two-input NAND logic gate is usually the simplest function to implement in any CMOS technology process. Its functionality can be achieved with a total of only four transistors.

cryptographic algorithms, the length of the critical path and by extension the clock frequencies are echoed in the throughput metric which the literatures stipulates in different ways. As part of this thesis, we opt for an asymptotic characterisation in the sense that our investigated algorithms are described by the number of processed bits per second for asymptotically large inputs when the device is run at the maximum clock frequency permitted by the critical path.

Definition 1 (Throughput)**.** *Let τ be the critical path of a circuit in seconds and denote by B the number of processed bits per clock cycles for asymptotically large messages. Then the throughput of a cryptographic circuit T is given by the following relation:*

$$T = B/\tau.$$

Remark that the parameter B not only depends on the given algorithm but also the primitive family. For example, hardware-oriented stream ciphers produce a single keystream bit per clock cycle for asymptotically large messages hence for these algorithm we set $B = 1$. On the other hand, round-based block cipher implementations can only encrypt a message length corresponding to their block size n every R rounds where R is equal to the number of round function invocations, consequently for block ciphers, we have $B = n/R$. Similarly, for modes of operation or authenticated encryption with associated data schemes, B depends on the number of data bits processed for asymptotically large chunks of associated data and message bits.

After the circuit area minimisations, the optimisation of throughput is a key aspect of lightweight cryptography especially when it comes to the adoption of lightweight algorithms in commodity applications. Nevertheless, area and throughput are often orthogonal disciplines exemplified by implementations with constrained data paths. These serialised constructions often trade area with an significant decrease in throughput. Where a round-based block cipher encrypted a message every R-th clock cycle a serialized variant of the same algorithm may take several thousands of cycles for the same task. Recently, several high-throughput encryption schemes meant for inclusion in cellular 5G communication networks have appeared, for instance SNOW-V and Rocca [68, 116]. These algorithms make parallel use of a single AES round function in order to process data blocks in a single clock cycle which yields throughput rates in the range of several Gigabits per second in both hardware and software.

Power. This physical unit denotes the time-averaged electrical work a circuit performs over the course of its running time. In CMOS technology, the power consumption is rooted in two dominant sources.

- Static Power. This source is due to leakage current and other current continuously drawn from the power supply. For example, a capacitor naturally leaks its charge and thus permanently requires current to balance the leakage. Importantly, static power is generally not dependent on the frequency of the clock driving the circuit or user data.
- Dynamic Power. This type of power consumption originates from the repeated charging and discharging of load capacitances needed to drive the output signal of a gate.

Each $0 \rightarrow 1$ and $1 \rightarrow 0$ adds to the overall consumption and is effectively dependent on the clock frequency, the input data as well as the switching probability of the gate. For instance, on average, a logical AND gate switches fewer times than a XOR gate and thus draws less dynamic power. Note that this description of dynamic power consumption is a simplification. In reality, there further is a transient term that denotes the charging and discharging of gate-internal capacitors based on the switching activity of the input signals.

The majority of dynamic power consumption is not due to gates switching as part of the intended functionality of the circuit but is caused by dynamic hazards, colloquially termed glitches which denote unnecessary transitions produced by unequally delayed input signals. Glitches exhibit a ripple effect throughout a network that is amplified by the complexity of the circuit, consequently a gate's output signal may switch several times before it attains a desired stable value. The existence of glitches has further security implication in the context of side-channel-based power analysis. Inadvertent signal switches may leak secret values that normally are not recoverable in glitch-free circuits. In fact, the study and design of countermeasures that offer protection against glitch-caused leakages has been at the forefront among the side-channel researchers in recent years.

Energy. Power and energy consumption are correlated metrics. Energy is a measure of the total electrical work performed by the power source during the execution of a operation. Mathematically, energy is usually calculated through the time integral of the power consumption, more specifically

$$E = \int P \, dt.$$

From this equation we can infer that the leakage energy of a circuit is proportional to the running time of an algorithm, i.e., the longer a circuit is run, for example by lowering the clock frequency, the more leakage energy it consumes. By the same logic, if a circuit is operated in a low-leakage environment, the dynamic energy consumption should be independent of the clock frequency. It is of vital importance to remark that for lower clock frequencies, the physical time taken to encrypt becomes larger and even small leakage power results in significant enough energy consumption of the order of the dynamic energy. Then the total energy increases monotonously as the frequency decreases.

2.4 Electronic Design Automation

Having established the relevant hardware metrics covered in this thesis, it naturally begs the question of how these measurements can be obtained. It stands to reason that physically manufacturing circuits for the sole purpose of cryptographic research is arduous, uneconomical and time-intensive, consequently experiments are solely conducted in a simulated virtual environment with the help of industrial Electronic Design Automation (EDA) toolchains that comprise the majority of the design process before a circuit is eventually printed and shipped.

The lifecycle of a circuit begins at the drawing board with a high-level description that is laid down as a Register-Transfer Level (RTL) design by means of a hardware description language such as VHDL or Verilog. A RTL specification describes the circuit as a composition of storage elements, i.e., registers, and combinational logic that implements the desired functionality. During one clock cycle, data stored in the registers is modified by the by combinatorial functions and subsequently written back. Consequently, computations proceed synchronously with the driving clock such that the registers are only updated at the beginning of each cycle. In a next step, the RTL design is processed by a synthesis tool that generates a netlist of logic gates that implement the specification using a standard-cell library which describes the physical properties of logic gates such as size and timing which allows for the extraction of the first metrics name the total circuit area and the critical path. Furthermore, it is now possible to simulate the resulting circuit under different inputs and clock frequencies. This both serves as a way to verify correctness properties but also to determine other metrics, most importantly power and energy consumption. Note that, at this stage, the netlist at hand does not describe the actual physical placement and routing of the individual gates on a chip which is performed by different tools in ensuing steps and thus the evaluated metrics do not account for this induced overhead. Placing gates and wiring them appropriately during the placement and routing phases adds to the total circuit area. Furthermore, closely packed transistors exert parasitic capacitances between each other that are reflected in altered critical paths and power consumption figures. Nevertheless, the measurements from a simulated netlist are accurate enough to allow for a realistic characterisation of a circuit, still after placing and routing, the design can again be simulated for a more accurate overview of its properties. Lastly, once all the relevant information has been gathered, the chip is ready to be handed over to a semiconductor manufacturer in order to be produced.

All the measurements as part of this thesis are solely collected on post-synthesis netlists and thus exclude any costs in the wake of the placing and routing processes. This choice is commonplace in the cryptographic optimisation literature as researchers operate on the algorithm level where particular improvements are discernibly reflected in the post-synthesis experiments. As for tools in this thesis, all netlists are generated with the Design Compiler® logic synthesizer from Synopsys [8] supplied with RTL designs written in VHDL. The Design Compiler® software is the industry-standard tool when it comes to the synthesis of netlists and offers a complex interface in order to fine-tune parameters that guide the synthesis process. Most of these are irrelevant for our purposes, however the choice of the main compilation directive has tangible effects on the resulting netlist and its measurement metrics. For this body of work, we rely on the following commands that depending on the optimisation goal a particular directive is more appropriate than the others.[9]

- *compile*. This directive moderately attempts to optimize the resulting netlist. The syn-

[8] Version 2017.09

[9] Unfortunately, published research works in cryptographic optimisation often omit details corresponding to the synthesis of their designs which is detrimental to the reproducibility of the result. In this thesis, the exact synthesis routines alongside other occasional parameters are spelled out individually for each for the presented contributions.

thesizer is free to choose the mapping and the corresponding optimisation but in any case entity boundaries, as specified by the RTL specification, are respected meaning that optimisation work between individual components is disabled. This synthesis routine usually leads to larger circuits but can be beneficial when it comes to the critical path or power and energy consumption figures.

- *compile_ultra*. A high-effort routine that optimises beyond entity boundaries in which the RTL description is analysed as a single large component. It commonly yields the most area-efficient netlists and is thus the default synthesis directive for many cryptographic optimisation works in the literature. Note that this command generally brings about circuits with an opaque and hard-to-understand netlists. Furthermore, the feedback loop is more involved than for other directive as its running time is repeatedly an order of magnitude larger.

- *compile_ultra -no_autoungroup*. By means of this command individual RTL components are synthesised with the highest-effort routine but optimisation ceases beyond component boundaries. This synthesis directive is especially useful in the context of side-channel-resistant implementations where succinct netlists is desirable but the exact mapping of gates and their boundaries are of crucial importance as to fulfil the intended security properties imposed by the countermeasures.

Once the netlist has been generated, experiments can be conducted in order to determine its power consumption. However, before this step it is mandatory to verify the functional correctness of the design by simulating it with a set of test vectors. In our case, this gate-level simulation is performed by Synopsys VCS® that utilizes Standard Delay Format (SDF) files generated by Design Compiler® that specify the timing data of each signal in the netlist. SDF files are back-annotated meaning that exact delays are only determined after the netlist has been generated. VCS® further creates a Switching Activity Interchange Format (SAIF) file during the simulation process that records the number of times a signal toggles its value. As previously mentioned, the power consumption is a function of the total switching activity of the combined set of all logic gates that comprise the design which in turn depends on the data that is fed to the circuit as part of a user-specified testbench. Feeding the SAIF file obtained after the gate-level simulation to the Synopsys Power Compiler®, this tool finally calculates the exact power consumption of the design. Remark that for cryptographic algorithms the testbench is usually composed of random data, e.g., a set of uniformly distributed keys and plaintexts, this means that the larger the testbench the more accurate is the determined power consumption. Lastly, the energy consumption figure is gauged by multiplying the power consumption with the total time required to process all the test vectors.

Naturally, any kind of synthesis and subsequent gate-level simulation is inexorably tied to the choice of standard-cell library which comes in the form of a Liberty Timing file (LIB). This format is a monolithic, usually large, collection of all supported logic gates as well as flip-flops alongside their associated physical properties such as size timing and power consumption data. In this thesis, we will make use of four of these libraries (two open-source,

two proprietary) that exhibit varying characteristics and features thus offering a broad panel of insights over the resulting circuits.

NanGate 15 nm. This is the smallest library used in our experiments where the size in the name refers to the width of a single transistor. It is a freely available and developed as an open-source library by Silvaco meant for usage in research and teaching. Compared to more industry-grade technologies it offers a rather basic set of logic gates and flip-flops with simplified physical properties. For example, it does not feature any multi-input linear cells, i.e., XOR, XNOR gates with more than two inputs.

NanGate 45 nm. Silvaco further offers a larger open-source cell library with a similar set of supported gates (again without any multi-input linear gates) compared to its smaller sibling. Its defining feature is its prominent leakage power consumption that trumps even libraries of larger sizes.

Example 2. *At low clock frequencies, the total power consumption of a NanGate 45 nm circuit is disproportionally composed of leakage power meaning that the resulting energy consumption fluctuates strongly as a function of the clock frequency whereas for other libraries this dependence is considerably weaker. In Table 2.1, we compare the power and energy consumption of a fully unrolled Trivium circuit [47] that outputs 288 keystream bits per clock cycle for differing clock frequencies and both the NanGate 15 nm and NanGate 45 nm libraries.*

Table 2.1: Exemplification of the dynamic/leakage power consumption ratio in the NanGate 45 nm cell library compared to its smaller sibling NanGate 15 nm for a fully unrolled Trivium circuit as investigated in more thoroughly in Chapter 4 and [47].

Cell Library	Frequency	Power (μW)			Energy (nJ/1.28 Mbit)		
		Dynamic	Leakage	Total	Dynamic	Leakage	Total
NanGate 15 nm	10 MHz	88.24	38.81	127.1	39.17	17.22	56.39
	100 MHz	897.3	38.83	936.1	39.91	1.732	41.64
	1 GHz	8987	38.83	9026	39.97	0.173	40.15
NanGate 45 nm	10 MHz	461.2	470.9	932.1	205.1	209.5	414.6
	100 MHz	4005	472.1	4477	178.2	20.96	199.2
	1 GHz	39508	472.1	39980	175.1	2.088	177.2

Apart from these two open-source cell libraries with further conducted our experiments with two proprietary cell libraries of a larger sizes.

UMC 65 nm. This semiconductor technology first appeared in 2005 developed by United Microelectronics Corporation and was the worldwide first 65 nm process to be integrated in customer applications. It is a full-fledged library that features a large set of logic gates and tabulates their physical properties in great detail. The complexity difference of UMC 65 nm compared two the previous two libraries is reflected in the size of their LIB files wherein the description of the NanGate 15 nm and NanGate 45 technologies fit within a few Kilobytes, the UMC 65 nm specification requires several hundred Megabytes.

TSMC 90 nm. Similarly to the UMC 65 nm process this library was the first of its size when it was announced in 2004 by the Taiwan Semiconductor Manufacturing Company. Again, this is a large feature-rich technology tailored for the implementation of complex circuitry. Importantly, the number of optimisation steps a synthesis tools is able to perform directly depends on the complexity of the supplied cell library, i.e., more logic gates allow for more equivalent mappings of a RTL description that exhibit different trade-offs. For example, the fact that both NanGate libraries do not feature multi-input linear gates precludes potential circuit area savings as 2 XOR gates usually occupy more area than a corresponding single 3-XOR gate. [10]

All of the aforementioned four cell libraries come in different variants that prioritise distinct metrics including low power consumption or optimised timing. As the nature of our results is comparative, measurement figures of optimised algorithms are not presented independently but always stand in juxtaposition to existing works, the choice of cell library variant is of lesser importance in the sense that, usually, both the new optimisation and the existing result are synthesised anew for each paper using the same variant of a cell library. In Table 2.2, for the sake of completeness, we list the exact variant for each of the four libraries alongside the size of their 2-input NAND gates used in the calculation of the Gate Equivalent metric.

Table 2.2: Tabulation of the utilized cell library variants as part of this thesis and their respective 2-input NAND gate sizes. Note that the difference between the smallest and largest library is more than tenfold.

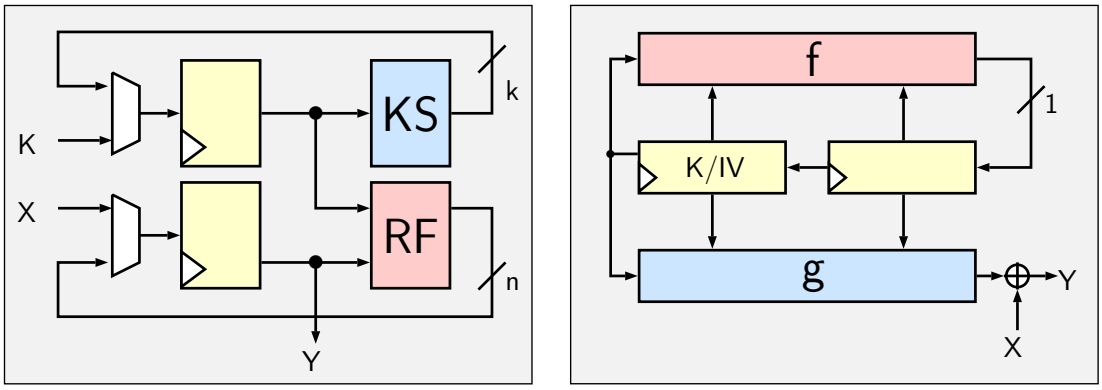
	NanGate 15 nm	NanGate 45 nm	UMC 65 nm	TSMC 90 nm
Variant	Fast	Fast	umk65lscllmvbbr	tcbn90lphp
Characteristic	Fast	Fast	Low Leakage	Low Power
NAND Area (μm^2)	0.196608	0.798	1.44	2.8224

2.5 Cipher-to-Circuit Mapping

Let us briefly distil the acquired knowledge about the synthesis and measurements of circuits from the previous sections and transpose it onto cryptographic circuits. Arguably, hardware implementations of symmetric cryptography algorithms stand as a quintessentially pure application of the RTL methodology in which storage and logic are separate instances. Virtually all relevant contemporary block ciphers iterate the same round function, for a constant number of rounds, over a state that is initialised with the plaintext and use second iterative key schedule function for the generation of keys for usage in each round departing from an initial master key. It is straightforward to map such a configuration into a RTL design in which two

[10] Proprietary foundries of a certain age are often licensed to universities and research institutes under non-disclosure agreements. At the time of writing, TSMC mass-produces chips with 5 nm technology and 3 nm processes are within the foreseeable future.

register banks hold the intermediate cipher and key state alongside a combinatorial layer that implements the round function and key update functions. At start of each clock cycle a newly computed state and key is stored in the registers and then input to the subsequent round. A high-level illustration of a generic block cipher circuit is depicted in Figure 2.1a. Modes of operation designs based on block ciphers are then easily created by treating the block cipher circuit as a black box and adding auxiliary states and logic around it. The same approach also translates to hardware-based stream ciphers where the state is travelling one bit at the time through a shift register. An update Boolean function that takes some state bits as inputs generates a new state bit in each clock cycle, similarly one keystream bit is created through a second Boolean function. A generic stream cipher configuration is shown in Figure 2.1b.



(a) Block Cipher (b) Stream Cipher

Figure 2.1: Basic circuit structure of a block cipher and a stream cipher. X denotes the plaintext and Y the corresponding ciphertext. The encryption key is written as K. Boxes in coloured yellow refer to registers and while the other boxes are combinatorial functions. Both illustrations are simplifications of existing algorithms. As a matter of fact, block ciphers are usually equipped with a third round constant state that is updated with each clock cycle. On the other hand, hardware stream ciphers are commonly composed of more than just a single shift register.

Most symmetric encryption circuits found in the literature can be reduced to the generic blueprint from Figure 2.1, consequently, every investigated algorithm in this thesis is firmly rooted in the same methodology independent of the ensuing optimisation.

2.6 Ciphers

In the following, we detail block and stream cipher algorithms that form the basis of our optimisation endeavours. Each cipher's description is self-contained and thus enables quick forward-referencing to their corresponding papers.

2.6.1 Trivium

This hardware-based stream cipher first appeared in 2006 in a work by Christophe De Cannière [61] and was included as one option in the ECRYPT Stream Cipher Project (eSTREAM), a cryptographic competition organised by the European Union in the search for new lightweight stream ciphers for both software and hardware environments.[11] The construction was later standardised as ISO/IEC 29192-3.[12].

Trivium features a state of 288 bits $X = (x_1, x_2, \ldots, x_{288})$ and key size of 80 bits alongside an initialisation vector of the same length.[13] At the beginning of the initialisation phase, the secret key $K = (k_1, k_2, \ldots, k_{80})$ and the public initialisation vector $\mathrm{IV} = (\mathrm{iv}_1, \mathrm{iv}_1, \ldots, \mathrm{iv}_{80})$ are loaded into the state and subsequently the state update function is run for $4 \cdot 288 = 1152$ rounds which corresponds to four full state rotations. Note that the 288-bit state is actually partitioned into three lanes of different sizes that, in a circular fashion, feed into each other. Lastly, a total of n keystream bits $Z = (z_1, \ldots, z_n)$ are generated one bit at a time where n is user-defined constant. Both the initialisation and keystream functions are detailed in Algorithm 1.

Algorithm 1 Trivium Initialisation and Keystream Functions

function `Initialisation`(K, IV)
- $(x_1, \ldots, x_{93}) \leftarrow (k_1, \ldots, k_{80}, 0, \ldots, 0)$
- $(x_{94}, \ldots, x_{177}) \leftarrow (\mathrm{iv}_1, \ldots, \mathrm{iv}_{80}, 0, \ldots, 0)$
- $(x_{178}, \ldots, x_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$
- **for** $i = 1$ to $4 \cdot 288$ **do**
  - $t_1 \leftarrow x_{66} \oplus (x_{91} \wedge x_{92}) \oplus x_{93} \oplus x_{171}$
  - $t_2 \leftarrow x_{162} \oplus (x_{175} \wedge x_{176}) \oplus x_{177} \oplus x_{264}$
  - $t_3 \leftarrow x_{243} \oplus (x_{286} \wedge x_{287}) \oplus x_{288} \oplus x_{69}$
  - $(x_1, \ldots, x_{93}) \leftarrow (t_3, x_1, \ldots, x_{92})$
  - $(x_{94}, \ldots, x_{177}) \leftarrow (t_1, x_{94}, \ldots, x_{176})$
  - $(x_{178}, \ldots, x_{288}) \leftarrow (t_2, x_{178}, \ldots, x_{287})$

function `Keystream`(X)
- **for** $i = 1$ to n **do**
  - $s_1 \leftarrow x_{66} \oplus x_{93}$
  - $s_2 \leftarrow x_{162} \oplus x_{177}$
  - $s_3 \leftarrow x_{243} \oplus x_{288}$
  - $z_i \leftarrow s_1 \oplus s_2 \oplus s_3$
  - $t_1 \leftarrow s_1 \oplus (x_{91} \wedge x_{92}) \oplus x_{171}$
  - $t_2 \leftarrow s_2 \oplus (x_{176} \wedge x_{177}) \oplus x_{264}$
  - $t_3 \leftarrow s_3 \oplus (x_{286} \wedge x_{287}) \oplus x_{69}$
  - $(x_1, \ldots, x_{93}) \leftarrow (t_3, x_1, \ldots, x_{92})$
  - $(x_{94}, \ldots, x_{177}) \leftarrow (t_1, x_{94}, \ldots, x_{176})$
  - $(x_{178}, \ldots, x_{288}) \leftarrow (t_2, x_{178}, \ldots, x_{287})$

The most striking characteristic of the Trivium stream cipher lies in the simplicity of its state update function, that contributes to an overall unquestionably compact circuit area footprint, composed of three strands of the same 5-bit Boolean function that each contain only a single non-linear term, i.e.,

$$x_{66} \oplus (x_{91} \wedge x_{92}) \oplus x_{93} \oplus x_{171},$$
$$x_{162} \oplus (x_{175} \wedge x_{176}) \oplus x_{177} \oplus x_{264},$$
$$x_{243} \oplus (x_{286} \wedge x_{287}) \oplus x_{288} \oplus x_{69}.$$

[11] `https://www.ecrypt.eu.org/stream`

[12] `https://www.iso.org/standard/56426.html`

[13] As we will see later in Chapter 4, list indexing starting from 1 proves more convenient for Trivium.

With a feature-rich cell-library one such strand can be implemented with a single 2-AND gate and one 4-XOR gate. Despite this succinctness, the algorithm has been resisting cryptanalytic efforts since its conception with there being to-date no key-recovery attack with a complexity below the exhaustive search bound and thus remains an attractive target for the symmetric cryptography community.

2.6.2 GIFT

Although the PRESENT block cipher [38] invigorated research in lightweight hardware encryption primitives its security guarantees were becoming increasingly volatile in the ensuing years. GIFT by Banik et al. [26] spawned out of the reinterpretation of PRESENT's core design principles and culminated in a block cipher that is both more secure and more efficient to implement as an ASIC.

GIFT uses a 128-bit key $K = (K_0, K_1, \ldots, K_7)$ where K_i is a 16-bit word but specifies two variants with either a 64-bit or a 128-bit block size. With regards to thesis only the 128-bit block version is relevant and thus described in this section. It operates over 40 rounds processing a 128-bit state $X = (x_0, x_1, \ldots, x_{127})$ using a round function composed of three subroutines

$$(\text{AddRoundKey} \circ \text{PermBits} \circ \text{SubCells})(X, K).$$

SubCells. As is commonplace in block cipher constructions the substitution layer is the only non-linear. In GIFT, it is a 4-bit invertible function GS that is applied 32 times in on each nibble of full state, i.e.,

$$(x_{4i}, x_{4i+1}, x_{4i+2}, x_{4i+3}) \leftarrow \text{GS}\,(x_{4i}, x_{4i+1}, x_{4i+2}, x_{4i+3})\,,\ \forall i \in \{0, \ldots, 31\}.$$

The function GS has an especially succinct mapping in term of circuit area which can be constructed with as few as ten 2-input logic gates. The complete substitution table is given in Table 2.3.

Table 2.3: 4-bit substitution table of the GIFT block cipher.

w	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
GS(w)	1	10	4	12	6	15	3	9	2	13	11	7	5	0	8	14

PermBits. As with PRESENT, the diffusion layer of GIFT is a bit-wise permutation P. This means that each bit of the intermediate is independently moved to a new position. In other words,

$$x_{P(i)} \leftarrow x_i,\ \forall i \in \{0, \ldots, 127\}.$$

From a hardware perspective, bit-wise permutations are an optimal choice as the computation is easily achieved by wiring and thus does not require any kind of additional logic. The full permutation table is listed in Table 2.4.

AddRoundKey. Finally, in each round a 64-bit round key $U||V$ is added to the cipher state

Table 2.4: 128-bit permutation table of the GIFT block cipher.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	96	1	34	67	64	97	2	35	32	65	98	3	0	33	66	99
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	10	5	38	71	68	101	6	39	36	69	102	7	4	37	70	103
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	104	9	42	75	72	105	10	43	40	73	106	11	8	41	74	107
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	108	13	46	79	76	109	14	47	44	77	110	15	12	45	78	111
i	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
$P(i)$	112	17	50	83	80	113	18	51	48	81	114	19	16	49	82	115
i	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
$P(i)$	116	21	54	87	84	117	22	55	52	85	118	23	20	53	86	119
i	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
$P(i)$	120	25	58	91	88	121	26	59	56	89	122	27	24	57	90	123
i	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
$P(i)$	124	29	62	95	92	125	30	63	60	93	126	31	28	61	94	127

where $U = (u_0, \ldots, u_{31}) = (K_2, K_3)$ and $V = (v_0, \ldots, v_{31}) = (K_6, K_7)$ are each 32-bit words extracted from the key state. These 64 bits of round key material are not added to in consecutive stretches but rather distributed evenly over the full intermediate state in the following fashion:

$$x_{4i+2} \leftarrow x_{4i+2} \oplus u_i, \;\; x_{4i+1} \leftarrow x_{4i+1} \oplus v_i, \; \forall i \in \{0, \ldots, 31\}.$$

Key Schedule. The key state is updated in each round through two separate rotations that, similarly to the PermBits operations, can be implemented in hardware with zero area overhead.

$$(K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7) \leftarrow (K_6 \ggg 2, K_7 \ggg 12, K_0, K_1, K_2, K_3, K_4, K_5).$$

Lastly, in order to break symmetric between the round functions, GIFT utilises a 6-bit LFSR for the generation of round constants $C = (c_0, c_1, c_2, c_3, c_4, c_5)$ that are added to the cipher state at specific positions such that

$$\begin{aligned} x_0 &\leftarrow x_0 \oplus 1, & x_{104} &\leftarrow x_{104} \oplus c_0, & x_{108} &\leftarrow x_{108} \oplus c_1, \\ x_{112} &\leftarrow x_{112} \oplus c_2, & x_{116} &\leftarrow x_{116} \oplus c_3, & x_{120} &\leftarrow x_{120} \oplus c_4, \\ x_{124} &\leftarrow x_{124} \oplus c_5. \end{aligned}$$

C is initialised to the zero bit string then affinely update in each round in the following way:

$$(c_0, c_1, c_2, c_3, c_4, c_5) \leftarrow (c_1, c_2, c_3, c_4, c_5, c_0 \oplus c_1 \oplus 1).$$

Bit-Sliced GIFT. An alternative ordering of the GIFT state bits in the form of a 2-dimensional array exists in which the state bits are arranged in four 32-bit segments. This bit ordering eases the implementation of the block cipher in software as it facilitates parallel computation

of the round function routines. More specifically, the state bits are ordered as follows:

$$X = \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} x_3 & x_7 & \cdots & x_{119} & x_{123} & x_{127} \\ x_2 & x_6 & \cdots & x_{118} & x_{122} & x_{126} \\ x_1 & x_5 & \cdots & x_{117} & x_{120} & x_{125} \\ x_0 & x_4 & \cdots & x_{116} & x_{119} & x_{124} \end{bmatrix} = \begin{bmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,29} & x_{0,30} & x_{0,31} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,29} & x_{1,30} & x_{1,31} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,29} & x_{2,30} & x_{2,31} \\ x_{3,0} & x_{3,1} & \cdots & x_{3,29} & x_{3,30} & x_{3,31} \end{bmatrix}.$$

In this representation, the substitution layer is now computed across the matrix rows, i.e.,

$$(x_{0,j}, x_{1,j}, x_{2,j}, x_{3,j}) \leftarrow \mathrm{GS}\left(x_{0,j}, x_{1,j}, x_{2,j}, x_{3,j}\right), \ \forall j \in \{0, \ldots, 31\}.$$

On the other hand, the bit-wise permutation is decomposable into four sub-permutations P_i that act independently on each row of the state matrix; by abuse of notation we write

$$(X_0, X_1, X_2, X_3) \leftarrow (P_0(X_0), P_1(X_1), P_2(X_2), P_3(X_3)).$$

The complete permutation table of the bit-sliced variant of GIFT is given in Table 2.5.

Table 2.5: The four 32-bit permutation tables of the bit-sliced GIFT block cipher.

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_0(j)$	16	8	0	24	17	9	1	25	18	10	2	26	19	11	3	27
j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_0(j)$	20	12	4	28	21	13	5	29	22	14	6	30	23	15	7	31
j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_1(j)$	8	0	24	16	9	1	25	17	10	2	26	18	11	3	27	19
j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_1(j)$	12	4	28	20	13	5	29	21	14	6	30	22	15	7	31	23
j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_2(j)$	0	24	16	8	1	25	17	9	2	26	18	10	3	27	19	11
j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_2(j)$	4	28	20	12	5	29	21	13	6	30	22	14	7	31	23	15
j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_3(j)$	24	16	8	0	25	17	9	1	26	18	10	2	27	19	11	3
j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_3(j)$	28	20	12	4	29	21	13	5	30	22	14	6	31	23	15	7

For our purposes, later on in this thesis, a different notation of the bit-sliced permutation table shown in Table 2.5 appears to be more practical, This representation is detailed in Table 2.6.

Ultimately, the 64-bit round keys and round constants are now added to adjacent state bits, in other words row, such that

$$(X_0, X_1, X_2, X_3) \leftarrow (X_0, X_1 \oplus V, X_2 \oplus U, X_3),$$
$$(X_0, X_1, X_2, X_3) \leftarrow (X_0, X_1, X_2, X_3 \oplus (1, 0, 0, \ldots, c_0, c_1, c_2, c_3, c_4, c_5)).$$

Note that the key state and round constant do not need to be reordered and are thus equivalent to the original GIFT specification.

On a side note, this technique of rearranging state bits into a more software-suitable or-

Table 2.6: Alternative representation of the bit-sliced GIFT permutation tables. Following the notation by SUNDAE-GIFT proposal [15], the bit identified by j moves to the its new position denoted in X_i after application of the permutation. For example, bit 31 of X_1 is shift one position to the right.

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_0(X_0)$	2	6	10	14	18	22	26	30	1	5	9	13	17	21	25	29
j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_0(X_0)$	0	4	8	12	16	20	24	28	3	7	11	15	19	23	27	31
j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_1(X_1)$	1	5	9	13	17	21	25	29	0	4	8	12	16	20	24	28
j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_1(X_1)$	3	7	11	15	19	23	27	31	2	6	10	14	18	22	26	30
j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_2(X_2)$	0	4	8	12	16	20	24	28	3	7	11	15	19	23	27	31
j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_2(X_2)$	2	6	10	14	18	22	26	30	1	5	9	13	17	21	25	29
j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_3(X_3)$	3	7	11	15	19	23	27	31	2	6	10	14	18	22	26	30
j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_3(X_3)$	1	5	9	13	17	21	25	29	0	4	8	12	16	20	24	28

dering is commonly termed *bitslicing*. In the case of bitsliced GIFT, it is straightforward to see that when the four state rows are each loaded in a 32-bit register, it becomes possible to calculate the 32 substitutions of SubCells in parallel by computing the function as a Boolean circuit instead of using a lookup table. Bitslicing in cryptography first appeared in an optimisation work on DES [33] but arguably had to biggest influence on AES implementations, e.g., see [1, 118].

GIFT-COFB. This Authenticated Encryption with Associated Data scheme based on the bitsliced GIFT block cipher is a NIST LWC finalist. The construction adheres to the *COmbined FeedBack* mode of operation [57] which provides a processing rate of 1, meaning a single block cipher invocation per input data block. The mode only adds an additional 64-bit state L to the existing block cipher state. Let $n = 128$ and denote by E_K a single bitsliced GIFT encryption using a key $K \in \{0,1\}^n$. Furthermore, $N \in \{0,1\}^n$ signifies a nonce and A represents a associated data bitstring and M a message bitstring. GIFT-COFB intersperses E_K calls with that of several component functions. In particular, it uses a truncation procedure $\text{Trunc}_r(X)$ that retrieves the r most significant bits of and a padding function $\text{Pad}(X)$ that pads inputs that are not a multiple of n, in the following manner:

$$\text{Pad}(X) = \begin{cases} X & \text{if } X \neq \epsilon \text{ and } |X| \bmod n = 0 \\ X \| 10^{(n-(|X| \bmod n)-1)} & \text{otherwise.} \end{cases}$$

Additionally, the internal state enters a feedback function between encryptions composed of two rotations of an input $X = (X_0, X_1)$ where $X_i \in \{0,1\}^{n/2}$ such that

$$\text{Feed}(X) = (X_1, X_0 \lll 1).$$

Alongside the execution of Feed, the auxiliary state L is updated through a multiplication over the finite field GF(2^{64}) generated by the root of the polynomial $p_{64}(x) = x^{64} + x^4 + x^3 + x + 1$. Consequently, the doubling of an element $z = z_0 z_1 \cdots z_{63} \in \mathrm{GF}(2^{64})$, i.e., the multiplication by the primitive element $\alpha = x = 0^{62}10$, is conveniently calculated as

$$\alpha \cdot z = \begin{cases} z \ll 1 & \text{if } z_0 = 0 \\ (z \ll 1) \oplus 0^{59}11011 & \text{otherwise.} \end{cases}$$

By leveraging this multiplication, we can similarly triple an element z by calculating $(1+\alpha)\cdot z$. The encryption of the last block of both A and M is preceded by the multiplication of L by $(1+\alpha)^x$ and $(1+\alpha)^y$ respectively, where

$$x = \begin{cases} 1 & \text{if } |A| \bmod n = 0 \text{ and } A \neq \epsilon, \\ 2 & \text{otherwise;} \end{cases} \qquad y = \begin{cases} 1 & \text{if } |M| \bmod n = 0 \text{ and } M \neq \epsilon, \\ 2 & \text{otherwise.} \end{cases}$$

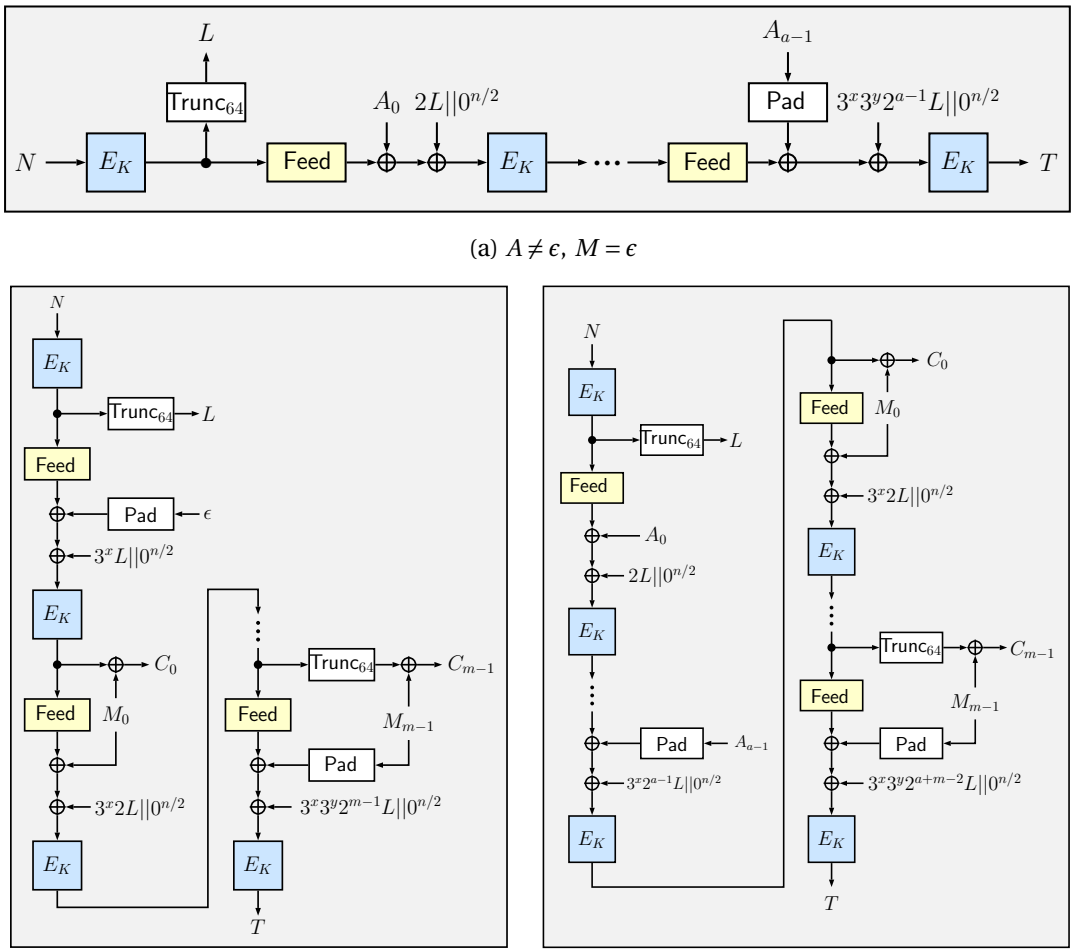
All other encryptions lead to a multiplication of L by α, excluding the initial encryption of the nonce. Ultimately, the mode of operation produces a ciphertext C of size $|C| = |M|$ and a tag $T \in \{0,1\}^n$. The complete GIFT-COFB encryption algorithm is given in Algorithm 2 and the corresponding schematic depiction is shown in Figure 2.2.

Algorithm 2 GIFT-COFB Encryption Function

function Encrypt(K, N, A, M)
  $X_0 \leftarrow E_K(N)$, $L \leftarrow \mathrm{Trunc}_{n/2}(X_0)$
  $(A_0, \ldots, A_{a-1}) \xleftarrow{n} \mathrm{Pad}(A)$
  if $M \neq \epsilon$ **then**
    $(M_0, \ldots, M_{m-1}) \xleftarrow{n} \mathrm{Pad}(M)$
  for $i \leftarrow 0$ to $a-2$ **do**
    $L \leftarrow \alpha \cdot L$
    $Y_i \leftarrow A_i \oplus \mathrm{Feed}(X_{i-1}) \oplus L||0^{n/2}$
    $X_i \leftarrow E_K(Y_i)$
  if $|A| \bmod n = 0$ and $A \neq \epsilon$ **then**
    $L \leftarrow (\alpha+1) \cdot L$
  else $L \leftarrow (\alpha+1)^2 \cdot L$
  if $M = \epsilon$ **then** $L \leftarrow (\alpha+1)^2 \cdot L$
  $Y_{a-1} \leftarrow A_{a-1} \oplus \mathrm{Feed}(X_{a-2}) \oplus L||0^{n/2}$
  $X_{a-1} \leftarrow E_K(Y_{a-1})$
  for $i \leftarrow 0$ to $m-2$ **do**
    $L \leftarrow \alpha \cdot L$
    $C_i \leftarrow M_i \oplus X_{i+a-1}$
    $Y_{i+a} \leftarrow M_i \oplus \mathrm{Feed}(X_{i+a-1}) \oplus L||0^{n/2}$
    $X_{i+a} \leftarrow E_K(Y_{i+a})$
  if $M \neq \epsilon$ **then**
    if $|M| \bmod n = 0$ **then**
      $L \leftarrow (1+\alpha) \cdot L$
    else $L \leftarrow (1+\alpha)^2 \cdot L$
    $C_{m-1} \leftarrow M_{m-1} \oplus X_{a+m-2}$
    $Y_{a+m-2} \leftarrow M_{m-1} \oplus \mathrm{Feed}(X_{a+m-2}) \oplus L||0^{n/2}$
    $X_{a+m-2} \leftarrow E_K(Y_{a+m-2})$
    $C \leftarrow \mathrm{Trunc}_{|M|}(C_0, \ldots, C_{m-1})$
    $T \leftarrow X_{a+m-2}$
  else $C \leftarrow \epsilon$, $T \leftarrow X_{a-1}$

2.6.3 SKINNY

Apart from GIFT there is a second lightweight block cipher that enjoyed the same popularity in recent years. SKINNY [30] is a construction in the tweakey framework in which part of the encryption key is designated as a tweak that may be replaced. It comes with a block size of 64 or 128 but as with GIFT only the 128-bit block size variant is relevant for our purposes. It



(a) $A \neq \epsilon,\ M = \epsilon$

(b) $A = \epsilon,\ M \neq \epsilon$

(c) $A \neq \epsilon,\ M \neq \epsilon$

Figure 2.2: Schematic depiction of GIFT-COFB mode of operation for all associated data and plaintext sizes. We remark that an empty associated data input will always be padded to a full block, hence the minimum number of encryption calls is two.

further specifies tweakey sizes of $t \in \{128, 256, 384\}$ bits with $z = t/128$. Both the cipher state X and the key states K_i for $i \in \{1, \ldots z\}$ are arranged in 4×4 byte matrices of the form

$$X = \begin{bmatrix} X_0 & X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 & X_7 \\ X_8 & X_9 & X_{10} & X_{11} \\ X_{12} & X_{13} & X_{14} & X_{15} \end{bmatrix}, \quad K_i = \begin{bmatrix} K_{i,0} & K_{i,1} & K_{i,2} & K_{i,3} \\ K_{i,4} & K_{i,5} & K_{i,6} & K_{i,7} \\ K_{i,8} & K_{i,9} & K_{i,10} & K_{i,11} \\ K_{i,12} & K_{i,13} & K_{i,14} & K_{i,15} \end{bmatrix}.$$

As SKINNY is a substitution-permutation network its round function is built around a non-linear substitution layer and linear diffusion layer. Depending on the tweakey size the cipher iterates its round function for a different amount of times, namely a total of $40 + (z-1) \cdot 8$

rounds. Below, we list the individual component sub-routines of the round function

$$(\text{MixColumns} \circ \text{ShiftRows} \circ \text{AddRoundTweakey} \circ \text{AddConstant} \circ \text{SubCells})(X, K).$$

SubCells. SKINNY deploys an efficient 8-bit substitution box that consists of a single Boolean function in combination with a bit-wise permutation repeated over four rounds as shown in Algorithm 3. Note that the permutation in the fourth round is different.

Algorithm 3 8-bit S-box of SKINNY

1: **function** S-Box$(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$
2: **for** $i \leftarrow 1$ to 4 **do**
3: $(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \leftarrow (x_0, x_1, x_2, x_3 \oplus \overline{(x_0 \vee x_1)}, x_4, x_5, x_6, x_7 \oplus \overline{(x_4 \vee x_5)})$
4: **if** $i < 4$ **then** $(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \leftarrow (x_5, x_6, x_0, x_1, x_3, x_7, x_4, x_2)$
5: **else** $(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \leftarrow (x_0, x_1, x_2, x_3, x_4, x_6, x_5, x_7)$

AddConstants. After the substitution layer, a round constant generated by a 6-bit affine LFSR $(\text{rc}_0, \text{rc}_1, \text{rc}_2, \text{rc}_3, \text{rc}_4, \text{rc}_5)$ are added into the state at specific indices. For better readability, we arrange the round constant bits in a 4×4 byte matrix of the form

$$X \leftarrow X \oplus \begin{bmatrix} C_0 & 0 & 0 & 0 \\ C_1 & 0 & 0 & 0 \\ C_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

where $C_0 = (0,0,0,0,\text{rc}_2,\text{rc}_3,\text{rc}_4,\text{rc}_5)$, $C_1 = (0,0,0,0,0,0,\text{rc}_0,\text{rc}_1)$ and $C_2 = (0,0,0,0,0,0,1,0)$. The LFSR itself is updated in the same fashion as the GIFT round constant generator, i.e.,

$$(\text{rc}_0, \text{rc}_1, \text{rc}_2, \text{rc}_3, \text{rc}_4, \text{rc}_5) \leftarrow (\text{rc}_1, \text{rc}_2, \text{rc}_3, \text{rc}_4, \text{rc}_5, \text{rc}_0 \oplus \text{rc}_1 \oplus 1).$$

AddRoundTweakey. Subsequently, a 64-bit round tweakeys are extracted and added to the first two rows of the intermediate state. The exact extraction functions differs with regards to z for $i \in \{0, \ldots, 7\}$ such that

$$X = \begin{cases} X_i \oplus K_{0,i} & \text{if } z = 1; \\ X_i \oplus K_{0,i} \oplus K_{1,i} & \text{if } z = 2; \\ X_i \oplus K_{0,i} \oplus K_{1,i} \oplus K_{2,i} & \text{if } z = 3. \end{cases}$$

ShiftRows. After round key and round constants have been added into the cipher state, the second, third and fourth rows are rotated by one, two and three to the right respectively. More specifically

$$X = \begin{bmatrix} X_0 & X_1 & X_2 & X_3 \\ X_7 & X_4 & X_5 & X_6 \\ X_{10} & X_{11} & X_8 & X_9 \\ X_{13} & X_{14} & X_{15} & X_{12} \end{bmatrix}.$$

MixColumns. The last operation of the round function consists in applying a column-wise function on the state described by a binary matrix multiplication of the form

$$X = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X_0 & X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 & X_7 \\ X_8 & X_9 & X_{10} & X_{11} \\ X_{12} & X_{13} & X_{14} & X_{15} \end{bmatrix}.$$

Key Schedule. The tweakeys are individually updated through a byte-wise permutation P_T that shuffles the matrix entries

$$K_{i,j} \leftarrow K_{i,P_T(j)},\ \forall i \in \{0,1\},\ \forall j \in \{0,15\},$$

where P_T is described in Table 2.7.

Table 2.7: Byte-wise SKINNY tweakey schedule permutation.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_T(i)$	9	15	8	13	10	14	12	11	0	1	2	3	4	5	6	7

In addition to the permutation, when $z > 1$, each byte of the first two rows of K_2, K_3 is further updated by two other affine functions:

$$K_{2,i} \leftarrow \begin{cases} F(K_{2,i}) & \text{if } i < 8, \\ K_{2,i} & \text{otherwise} \end{cases} \quad K_{3,i} \leftarrow \begin{cases} G(K_{3,i}) & \text{if } i < 8, \\ K_{3,i} & \text{otherwise} \end{cases},$$

where F and G are given by the following relations:

$$F : (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \leftarrow (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_0 \oplus x_2),$$
$$G : (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \leftarrow (x_7 \oplus x_1, x_0, x_1, x_2, x_3, x_4, x_5 x_6, x_0).$$

2.6.4 AES

The progenitor of all modern block ciphers and still the most widely integrated encryption solution is the Advanced Encryption Standard [60]. It operates with a block size of 128 bits and offers key sizes of 128, 192 and 256 bits that entail 10, 12 or 14 round function repetitions respectively, however in this body of work, only the 128-bit and 256-bit variants are relevant. Similar to SKINNY, the cipher state is represented as column-major byte matrix

$$X = \begin{bmatrix} X_0 & X_4 & X_8 & X_{12} \\ X_1 & X_5 & X_9 & X_{13} \\ X_2 & X_6 & X_{10} & X_{14} \\ X_3 & X_7 & X_{11} & X_{15} \end{bmatrix}.$$

However, unlike SKINNY all algebraic operation on bytes in AES are conducted over the Rijndael finite field GF(2^8) modulo the irreducible polynomial $p(x) = x^8 + x^4 + x^3 + x + 1$. Depending on the key size, its state is either represented by a 4×4 or 4×8 column-major byte matrix of the form

$$K = \begin{bmatrix} K_0 & K_4 & K_8 & K_{12} \\ K_1 & K_5 & K_9 & K_{13} \\ K_2 & K_6 & K_{10} & K_{14} \\ K_3 & K_7 & K_{11} & K_{15} \end{bmatrix}, \quad K = \begin{bmatrix} K_0 & K_4 & K_8 & K_{12} & K_{16} & K_{20} & K_{24} & K_{28} \\ K_1 & K_5 & K_9 & K_{13} & K_{17} & K_{21} & K_{25} & K_{29} \\ K_2 & K_6 & K_{10} & K_{14} & K_{18} & K_{22} & K_{26} & K_{30} \\ K_3 & K_7 & K_{11} & K_{15} & K_{19} & K_{23} & K_{27} & K_{31} \end{bmatrix}.$$

As before, we detail the individual routines of the round function

$$(\text{AddRoundKey} \circ \text{MixColumns} \circ \text{ShiftRows} \circ \text{SubBytes})(X, K).$$

Note that the very first operation (often referred to the key whitening state), before commencing with the round function iterations, consists in adding the first 128-bits of the encryption key into the state, i.e.,

$$X = X \oplus \begin{bmatrix} K_0 & K_4 & K_8 & K_{12} \\ K_1 & K_5 & K_9 & K_{13} \\ K_2 & K_6 & K_{10} & K_{14} \\ K_3 & K_7 & K_{11} & K_{15} \end{bmatrix}.$$

SubBytes. The first operation of each new round is the non-linear layer, represented by an 8-bit substitution box S that is applied on each byte of the state. Mathematically, the substitution in an affine transformation of a multiplicative inverse of the form $S(X_i) = A \times X_i^{-1} + b$, $\forall i \in \{0, \dots 15\}$ or more precisely

$$S(X_i) = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix},$$

where $(y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7)$ are the individual bits of the multiplicative inverse of the input byte $X_i^{-1} = (y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7)$.

ShiftRows. The next function consists in rotating the second, third and fourth row of the state matrix by one, two or three positions to the left as opposed to the rightward rotation found in

SKINNY. Schematically, this means

$$X = \begin{bmatrix} X_0 & X_4 & X_8 & X_{12} \\ X_5 & X_9 & X_{13} & X_4 \\ X_{10} & X_{14} & X_2 & X_6 \\ X_{15} & X_3 & X_7 & X_{11} \end{bmatrix}.$$

MixColumns. After rotating the state rows, each column of the matrix is multiplied over the Rijndael finite field by a constant matrix such that

$$X = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot X.$$

Importantly, the MixColumns operation is, for efficiency reasons, omitted in the ultimate round.

AddRoundKey. Lastly, the first 128-bits of the current state are added into the cipher state byte by byte as seen before during the initial key whitening phase.

Key Schedule. Conceptually, the generation of round keys is the most complex routine in AES as it makes use of both the S-box and a word-based rotation. Furthermore, the routines slightly differ from each other depending on the key size.

- *128-Bit.* In this variant, the the least significant bytes $(K_{12}, K_{13}, K_{14}, K_{15})$ pass through the S-box and are rotated by one position to the left before a round constant is added and the updated 32-bit word is added back into the key state. In the following, denote by $W = [W_0, W_1, W_2, W_3]^T$ a four-byte column vector, then we have

$$W = \begin{bmatrix} K_0 \oplus S(K_{13}) \oplus RC \\ K_1 \oplus S(K_{14}) \\ K_2 \oplus S(K_{15}) \\ K_3 \oplus S(K_{12}) \end{bmatrix}, \; W' = \begin{bmatrix} W_0 \oplus K_4 \\ W_1 \oplus K_5 \\ W_2 \oplus K_6 \\ W_3 \oplus K_7 \end{bmatrix}, \; W'' = \begin{bmatrix} W'_0 \oplus K_8 \\ W'_1 \oplus K_9 \\ W'_2 \oplus K_{10} \\ W'_3 \oplus K_{11} \end{bmatrix}, \; W''' = \begin{bmatrix} W''_0 \oplus K_{12} \\ W''_1 \oplus K_{13} \\ W''_2 \oplus K_{14} \\ W''_3 \oplus K_{15} \end{bmatrix}.$$

  The updated key state is then given by $K = [W, W', W'', W''']$.

- *256-Bit.* In this variant only half of the key state is updated per round. Denote by $V_i = [K_{16+4i}, K_{16+4i+1}, K_{16+4i+2}, K_{16+4i+3}]^T$, $\forall i \in \{0, \ldots, 3\}$ the last for columns of the key state, then the a newly generate state is of the form $K = [V_0, V_1, V_2, V_3, W, W', W'', W''']$. The exact update function varies for even and odd rounds such that for even rounds we have

$$W = \begin{bmatrix} K_0 \oplus S(K_{29}) \oplus RC \\ K_1 \oplus S(K_{30}) \\ K_2 \oplus S(K_{31}) \\ K_3 \oplus S(K_{28}) \end{bmatrix}, \; W' = \begin{bmatrix} W_0 \oplus K_4 \\ W_1 \oplus K_5 \\ W_2 \oplus K_6 \\ W_3 \oplus K_7 \end{bmatrix}, \; W'' = \begin{bmatrix} W'_0 \oplus K_8 \\ W'_1 \oplus K_9 \\ W'_2 \oplus K_{10} \\ W'_3 \oplus K_{11} \end{bmatrix}, \; W''' = \begin{bmatrix} W''_0 \oplus K_{12} \\ W''_1 \oplus K_{13} \\ W''_2 \oplus K_{14} \\ W''_3 \oplus K_{15} \end{bmatrix},$$

while for odd rounds the word rotation as well as the round constant addition are omitted which yields the following update function:

$$W = \begin{bmatrix} K_0 \oplus S(K_{28}) \\ K_1 \oplus S(K_{29}) \\ K_2 \oplus S(K_{30}) \\ K_3 \oplus S(K_{31}) \end{bmatrix}, \; W' = \begin{bmatrix} W_0 \oplus K_4 \\ W_1 \oplus K_5 \\ W_2 \oplus K_6 \\ W_3 \oplus K_7 \end{bmatrix}, \; W'' = \begin{bmatrix} W'_0 \oplus K_8 \\ W'_1 \oplus K_9 \\ W'_2 \oplus K_{10} \\ W'_3 \oplus K_{11} \end{bmatrix}, \; W''' = \begin{bmatrix} W''_0 \oplus K_{12} \\ W''_1 \oplus K_{13} \\ W''_2 \oplus K_{14} \\ W''_3 \oplus K_{15} \end{bmatrix}.$$

Ultimately, the 8-bit round constants depart from an initial value of $RC = 1$ and are doubled $RC = 2 \cdot RC$ over the Rijndael finite field in each round. Note that while ten round constants are needed for the 128-bit variant of AES only seven are required for the 256-bit version as it is only added in even rounds.

2.7 Threshold Implementations

Unprotected implementations of cryptographic algorithms are susceptible to side-channel-assisted attacks that exploit leakages of secret values in sources such as power consumption, electromagnetic emissions or execution time. This means that even if an algorithm has been proven cryptanalytically secure its implementation may not be. In order to disassociate secret values from potential leakages, a common approach is to include random values into the cipher state that mask critical values from being exposed in leakages. This approach is adheres to the secret-sharing methodology [119]. More specifically, key-related, intermediate bit values x_i are split into s independent shares $x_{i,0}, x_{i,1}, \ldots, x_{i,s-1}$ such that

$$\sum_{j=0}^{s-1} x_{i,j} = x_i.$$

In practice, sharing variables implies that each function $f(x_0, \ldots, x_{n-1}) = z$ within an algorithm needs to be decomposed into functions $f_i(\cdot) = z_i$ adhering to the same correctness requirement $\sum_{i=0}^{s-1} f_i = f$. In the following, we assume that an attacker is capable of probing individual wires of a cryptographic circuit and can extract their intermediate values during the computation [84]. More specifically, we consider d-th order security in the *glitch-robust* model, where information of any d wires can be gathered and processed from a circuit that exhibits glitches during its running time. We say that a masked implementation is d-probing secure that any observation made on up to d wires is statistically independent of the secret. In particular, d-probing security implies resistance against d-th order Differential Power Analysis (DPA) in the glitch-robust model.

The challenge of designing d-th order secure masking schemes has spawned various approaches, of which Threshold Implementations have crystallized themselves as one of the most adopted strategies. First introduced by Nikova et al. [35, 111] Threshold Implementations provide complete first-order security i.e., $d = 1$, against DPA in the presence of glitches. [14] With regards to this thesis we will restrict ourselves uniquely to first-order imple-

[14] It has been shown that against higher-order for $d > 1$ attacks Threshold Implementations do not offer ade-

mentations.

The decomposition of an n-variable Boolean function $f(x_0,\ldots,x_{n-1}) = z$ into a set of s functions $f_0,\ldots,f_{s-1}$ such that $\sum_{i=0}^{s-1} f_i = f$ is a first-order Threshold Implementation if and only if the following conditions are met:

- *Non-Completeness.* The functions $f_0,\ldots,f_{s-1}$ are first-order non-complete, if any function is independent of at least one input share.
- *Uniformity.* For all x such that $f(x) = z$, the input masking is said to be uniform if each set of valid input shares of x (i.e., those sum to x) have equal probability of occurring. If this holds, the shared implementation of f is said to be uniform if each valid output share also have equal probability of occurring.

The number of input shares s_{in} respectively number of output shares s_{out} necessary in order to achieve a non-complete and uniform sharing of a function of algebraic degree t is given by the below bounds [35]:

$$s_{\text{in}} \geq t+1, \quad s_{\text{out}} \geq \binom{t+1}{t}.$$

Example 3. *Note that a first-order TI of a quadratic function can thus be obtained with* $s_{in} = s_{out} = 3$. *In this work, we will bootstrap the sharing of an arbitrary quadratic function via the canonical direct sharing of the function* $f(x_2,x_1,x_0) = x_0 + x_1x_2$, *i.e.,*

$$\begin{aligned} f_0 &= x_{0,1} + x_{1,1}x_{2,1} + x_{1,1}x_{2,2} + x_{1,2}x_{2,1} \\ f_1 &= x_{0,2} + x_{1,0}x_{2,0} + x_{1,2}x_{2,0} + x_{1,0}x_{2,2} \\ f_2 &= x_{0,0} + x_{1,0}x_{2,0} + x_{1,1}x_{2,0} + x_{1,0}x_{2,1}. \end{aligned}$$

We use an analogous direct sharing for cubic terms.

2.8 Swap-and-Rotate

This methodology, due to Banik et al. [11], describes a generic technique to serialise block cipher circuits to a 1-bit data path by the means of a rotating shift register combined with purposeful swapping of state bits during specific intervals. We denote a n-bit register pipeline as $\text{FF}_0,\ldots,\text{FF}_{n-1}$ over which we define two basic algorithmic operations as shown in Algorithm 4.

While the rotate function is inherent to all shift registers the swap operator is implemented using regular d-flip-flops that are prefaced with a multiplexer that switches two or more input signals. In order to illustrate the usage of these two function let us indulge in a simple example.

Example 4. *Given a 4-bit state* (x_0,x_1,x_2,x_3) *store in a register pipeline* $FF_0FF_1FF_2FF_3$, *a leftward rotation by two positions can achieved with a single* $\text{Swap}(1,3)$ *over four cycles such that*

quate protection [114] without additional security measures.

Algorithm 4 Swap-and-Rotate Functions

1: **function** `Rotate`(FF, x)
2: $\quad \mathrm{FF}'_{n-1} \leftarrow x$
3: $\quad \mathrm{FF}'_{i-1} \leftarrow \mathrm{FF}_i$ for $i \in \{1, \ldots, n-2\}$
4: $\quad y \leftarrow \mathrm{FF}_0$
5: $\quad$ **return** (FF', y)

1: **function** `Swap`(FF, u, v)
2: $\quad \mathrm{FF}'_i \leftarrow \mathrm{FF}_i$ for $i \in \{0, \ldots, n-1\} \setminus \{u, v\}$
3: $\quad \mathrm{FF}'_u \leftarrow \mathrm{FF}_v$
4: $\quad \mathrm{FF}'_v \leftarrow \mathrm{FF}_u$
5: $\quad$ **return** FF'

at the beginning of the fourth cycle we have the state (x_2, x_3, x_0, x_1)*. A cycle-by-cycle example of the state pipeline with its corresponding swaps is given in Table 2.8 and the corresponding circuit in Figure 2.3.*

Table 2.8: Cycle-by-cycle tabulation of the leftward rotation by two positions.

Cycle	Pipeline	Swap
0	$\mathrm{FF}_0(x_0)\mathrm{FF}_1(x_1)\mathrm{FF}_2(x_2)\mathrm{FF}_3(x_3)$	Swap(1,3)
1	$\mathrm{FF}_0(x_1)\mathrm{FF}_1(x_0)\mathrm{FF}_2(x_3)\mathrm{FF}_3(x_2)$	Swap(1,3)
2	$\mathrm{FF}_0(x_0)\mathrm{FF}_1(x_1)\mathrm{FF}_2(x_2)\mathrm{FF}_3(x_3)$	No swap
3	$\mathrm{FF}_0(x_1)\mathrm{FF}_1(x_2)\mathrm{FF}_2(x_3)\mathrm{FF}_3(x_0)$	No swap
4	$\mathrm{FF}_0(x_2)\mathrm{FF}_1(x_3)\mathrm{FF}_2(x_0)\mathrm{FF}_3(x_1)$	No swap

Note that, of course the same transformation could have been performed by simple rotating the state for two cycles without any swap, however, as explained later, for bit-serial block cipher circuits it is important that any operation is performed in exactly n cycles where n is the block size of the underlying algorithm.

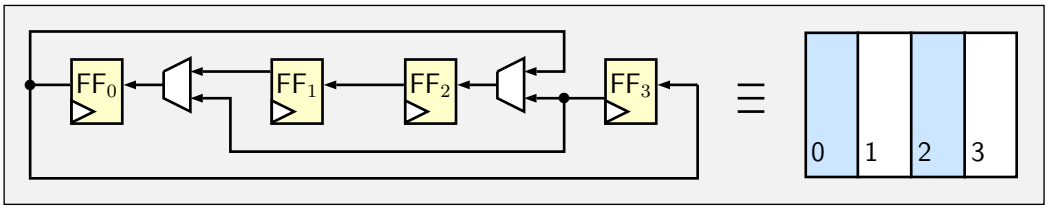


Figure 2.3: Swap-and-Rotate circuit for 4-bit leftward rotation by two positions. In order to ease notation we will use the simplified equivalent schematic on the right.

Green Cryptography Book I

3 AEAD Energy Analysis

[...] Cut away, cut away [...]

We commence the first book in this thesis with survey-type analysis that performs a thorough energy consumption analysis of nine hardware-oriented AEAD schemes of the second NIST LWC round that are constructed around a lightweight block cipher which resulted in the to-date most comprehensive energy study of cryptographic circuits. This chapter is comprised of material from a work in collaboration with Subhadeep Banik and Fatih Balli that was presented at the 19th International Conference on Cryptology and Network Security (CANS-2020) and was bestowed with a Best Paper Award [43].

As already alluded to earlier, energy is a drastically under-represented optimisation criteria in cryptographic hardware research owing to the difficulty in quantifying its behaviour. The most noteworthy entry is due to Banik et al. [18] in the authors presented a model that captures the energy consumption of a block cipher in terms of r, where r denotes the number of unrolling in an implementation. For many ciphers, including AES, their model verifiably predicts that the energy-optimal choice is $r = 1$, where for some lighter block ciphers, such as PRESENT, the optimal point shifts to $r = 2$. However, as stated before, block ciphers usually are not ready-to-use primitives and must be wrapped within a mode of operation. Therefore, the effects of additional circuitry to energy consumption remains unanswered.

Contributions. In this chapter, we claim the following contributions to the study of energy consumption in cryptographic circuits:

1. We explore the effects of the clock-gating and r-round unrolling techniques to deduce the architectural design choices that lead to the most energy efficient implementations. We look at each candidate individually and identify optimal circuit configurations that would reduce the energy consumption of AEAD circuit. The large number of implementations helps us make broader observations regarding energy efficiency in AEAD modes instantiated with lightweight block ciphers.

2. In parallel to the first effort, we provide a fair evaluation of the aforementioned candidates from NIST LWC. The data we obtain shows how each candidate fares, when implemented with a similar approach.

3. In partially unrolled circuits, we demonstrate that the optimal choice of r boils down to three factors; the complexity of the core cipher, the complexity of the surrounding

mode of operation circuitry and the amount of leakage energy for high-leakage cell libraries. Whereas the optimal choice for block ciphers is typically $r \in \{1,2\}$, for full AEAD circuits we experimentally demonstrate that this becomes $r \in \{2,3\}$.

4. In the last part of the paper we move to threshold implementations that provide security against power analysis attacks. Although, there exists numerous works that optimize the area of these circuits, actual projects that look at the energy consumption of these circuits as an optimizable metric are comparably rare. We look at both 3-share and 4-share threshold circuits, and look at factors like number of shares, decomposability of s-boxes that affect the energy consumption of such circuits.

Outline. The chapter unfolds as follows. Section 3.1 reiterates known energy reduction techniques and lays out a common interface and test bench for all implementations. In Section 3.2, we briefly introduce the chosen schemes alongside their internal block ciphers and detail their implementations. Section 3.3 investigates the effects of the individual design choices on the schemes and extends the energy model of of Banik et al. [18] block ciphers to modes of operations in the chosen authenticated encryption algorithms. We also elaborate on the obtained energy measurements and chart the results. In Section 3.4, we turn our attention to first-order threshold implementations of the AEAD schemes. We conclude our paper with the takeaway claims for designers and engineers in Section 3.5.

3.1 Modus Operandi

To guarantee fair conditions in our evaluation we unified our implementations under a common interface. Our hardware API is designed to be simple, as it assumes that the associated data and message bits are properly padded so that they only consists of multiple blocks. This padding must be done according to the individual specification of the AEAD scheme, before the AEAD operation is initiated in the circuit. Then our AEAD implementations can be used in all possible configurations (e.g., partial blocks, no authenticated data or no message blocks) and comply with the exact specification. Our reasoning for favouring this simpler API (with external-padding) is that it ensures that no significant energy is consumed to handle the API itself. For instance, the CAESAR HW API [80] requires padding to be done by the circuit, which brings a large array of multiplexers and amplifies the energy consumption for each loaded associated data and message block. However, depending on the application, this padding cost can be avoided, for instance, handling padding on a microprocessor that makes the call can be less costly, or the application might not even need padding, if the transmitted data always respects the block sizes. Nonetheless, a preprocessor circuit could be placed before our AEAD schemes to ensure CAESAR HW API compatibility.[15] The input and output ports of our hardware API are defined in the following way:

- CLK, RST: System clock and active-low reset signal.

[15] The interaction between padding preprocessor and cipher module has not been investigated in the literature and could be pose as a potential attack vector traditional and side-channel-assisted analysis.

- KEY, NONCE: Key and nonce vectors. These signals are stable once the circuit is reset and are kept active during the entire computation.
- DATA: Single data vector from which both associated data and regular plaintext blocks are loaded into the circuit. This choice saves an additional large multiplexer, since all the schemes process associated data and plaintext blocks separately and not in parallel.
- EAD, EPT: Single bit signals that indicate whether there are no associated data blocks (EAD) or no plaintext blocks (EPT). Both signals are supplied with the reset pulse and remain stable throughout the computation.
- LBLK, LPRT: Single bit signals that indicate whether the currently processed block is the last associated data block or the last plaintext block (LBLK), and also whether it is partially filled (LPRT). Both signals are supplied alongside each data block and remain stable during its computation.
- BRDY, ARDY: Single bit output indicators whether the circuit has finished processing a data block and a new one can be supplied on the following rising clock edge (BRDY) or the entire AEAD computation has been completed (ARDY).
- CRDY, TRDY: Single bit output indicators whether the CT and TAG ports will have meaningful ciphertext and tag values starting from the following rising clock edge.
- CT, TAG: Separate ciphertext and tag vectors. This again saves an additional multiplexer in schemes where the ciphertext and tag are not ready at the same time, or they appear at different wires.

Synthesis Options. All nine schemes were synthesised for all four cell libraries detailed in Section 2.4. Partially unrolled circuits were simulated at a constant clock frequency of 10 MHz. Furthermore, we made use of different synthesis directives depending on the circuit at hand as listed in Table 3.1.

Table 3.1: Synopsys Design Vision synthesis directives utilised for the investigated implementation strategies.

Circuit	Design Compiler® Compilation Flag	Reference
r-Round Unrolled	*compile_ultra*	Section 3.3
Threshold Implementations	*compile_ultra -no_autoungroup*	Section 3.4

Clock-Gating. This technique describes a general power-reduction technique that aims to limit the switching activity of register banks. A classic, non-gated flip-flop is continuously charged and discharged by the system clock which results in wasted activity during periods when the flip-flop needs to preserve its content for multiple cycles. The clock signal in a clock-gated register is artificially held constant through additional logic during these constant phases. There exist many ways to implement clock-gated registers, for this work, we

chose the simple approach of NANDing the clock signal with an active-low enable signal to produce a gated clock. Note that clock-gated registers usually incur a reduction of the total number of gates since the circuitry handling the enable signal is not needed any more. A clock-gated register bank can fall prey to timing issues if a single gated clock signal is used to drive many flip-flops, which can be circumvented by partitioning the register bank into smaller segments such that each segment is driven by its own gated clock as shown in Figure 3.1.

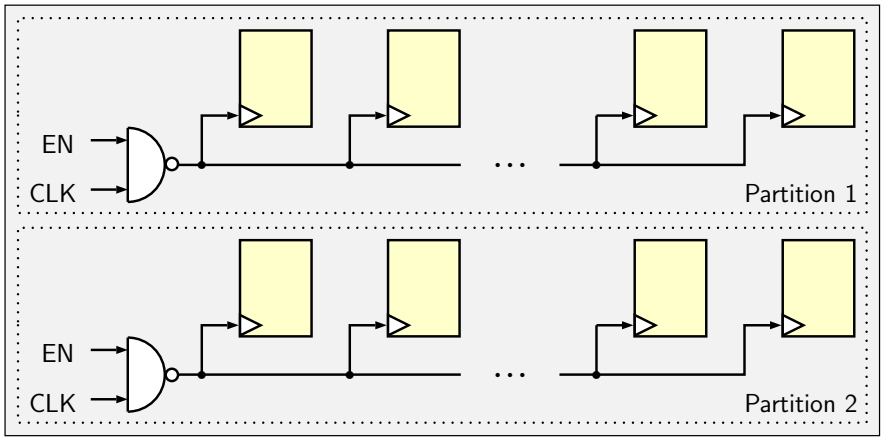


Figure 3.1: Schematic of a partitioned clock-gated register. See [113] for a comprehensive microarchitectural survey of different clock-gating implementation techniques. Note that clock-gating is only applicable if the underlying design permits idle registers during its running time.

3.2 Implementations

Out of the 32 remaining candidates[16] in the second round of the NIST lightweight competition we singled out nine schemes that are bootstrapped either directly via lightweight block ciphers or variants of them. Five out of the ten schemes are directly instantiated with the GIFT block cipher [26] or through a slightly adapted tweakable alteration. Three other schemes are based on the SKINNY tweakable block cipher [30] or a forked version of it [4]. Finally, the Pyjamask and Saturnin AEAD schemes deploy their own dedicated substitution-permutation networks of the same names. Table 3.2 lists all investigated schemes alongside their internal block cipher.

r-Round Unrolled. The sequential placement of multiple round function circuits allows the computation of several rounds during a single clock cycle. This results in fewer required cycles to complete one encryption, i.e., in an r-round partial unrolling setting a block ci-

[16]The majority of schemes in the second NIST LWC round are based on keyed permutations or use AES or other block ciphers that are normally not considered part of the lightweight spectrum. Grain-128AEAD is the only construction that is built around a stream cipher. The portfolio of ten finalist is composed of seven keyed permutations, two constructions based on lightweight block ciphers and Grain-128AEAD.

Table 3.2: Investigated second-round NIST LWC AEAD schemes based on lightweight block ciphers. Highlighted schemes have been chosen as LWC finalists. LOTUS-AEAD was submitted together with another almost equivalent construction termed LOCUS-AEAD which, for the sake of brevity, is omitted. Note that HyENA uses the original bit ordering of GIFT as opposed to the bitsliced variant.

Scheme	Block Cipher	Reference
GIFT-COFB	GIFT*-128	[21]
SUNDAE-GIFT	GIFT*-128	[15]
HyENA	GIFT-128	[56]
LOTUS-AEAD	TWE-GIFT-64	[55]
SKINNY-AEAD	SKINNY-128-384	[29]
Romulus	SKINNY-128-384	[74]
ForkAE	ForkSkinny	[5]
Pyjamask	Pyjamask-128	[71]
Saturnin	Saturnin	[53]

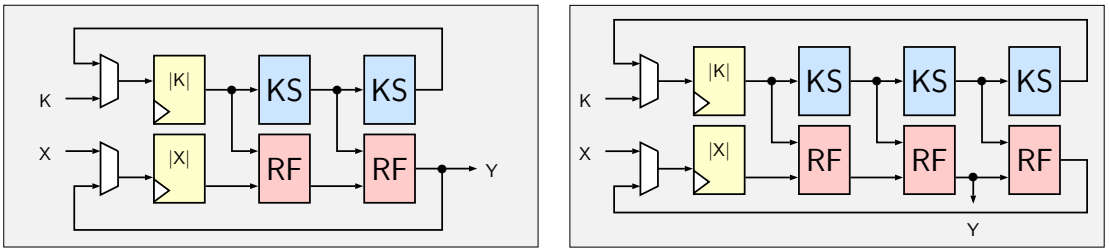
pher composed of R rounds can be computed in $\lceil \frac{R}{r} \rceil$ cycles. The adverse effects of unrolling include a larger overall circuit area and an increased signal delay across the circuit. Nevertheless, as shown by Banik et al. [18], partial unrolling can reduce the energy consumption of certain (especially lightweight) block ciphers noticeably. In broad terms, it is possible to quantify the total amount of consumed energy E as a quadratic polynomial function of the unrolling factor r such that

$$E = (Ar^2 + Br + C)\left(\left\lceil 1 + \frac{R}{r} \right\rceil\right),$$

where A, B and C represent energy values depending on the internal switching activity of the block cipher such as registers, multiplexers and arithmetic logic. Hence, if the block cipher is on the lighter side, E can be minimized for $r \geq 2$, on the other hand complex and heavy circuits such as AES incur large constants A, B and C where E is only minimized for $r = 1$. Using partial r-round unrolling the round function and key expansion circuits can be replicated r times and connected through data paths where the output of the last replicated circuit is stored in the state and key registers. Special care has to be taken when $r \nmid R$, here the ciphertext will not be produced by the last replicated instance but it must come from an intermediate computation as can be seen in Figure 3.2.

3.3 Effects of Design Choices

Let us now discuss the effects exerted on the nine investigated schemes by unrolling and clock-gating circuits in order to obtain a comprehensive picture of the engendered energy consumption landscape.



(a) 2-Round Unrolled

(b) 3-Round Unrolled

Figure 3.2: Basic circuit structure of 2-round and 3-round unrolled block ciphers. X denotes the plaintext and Y the corresponding ciphertext. The encryption key is written as K. Boxes in coloured yellow refer to registers and while the other boxes are combinatorial functions.

Clock Frequency. As indicated in Section 2.3 and independently shown in numerous papers [18, 25, 89] that in low leakage environments, at high enough frequencies, the total energy consumption of a circuit is independent of clock frequency since it is the measure of total circuit glitch. Consequently, what should be the frequency of operation at which one should benchmark energy figures for different AEAD schemes. If we opt for lower frequencies, the results will be heavily influenced by the leakage component, and then the energy optimisation exercise effectively reduces to an area minimisation problem, since circuits with lower silicon area also tend to have lower leakage. Although this problem is also important, it is less intellectually stimulating from purely an energy engineering point of view. When we compare different AEAD schemes for energy efficiency, ideally we should be comparing algorithmic/-circuit level aspects of the scheme that allow for lesser glitching or lesser logic transitions in the circuit nodes. In fact this is also the approach followed in energy optimisation of block and stream ciphers [18, 25] . This is essentially the optimisation of the dynamic energy component. Thus is why in our experiments we keep the clock frequency at 10 MHz so that the leakage power is rendered insignificant, and the paper becomes an exercise in comparing the switching characteristics of different AEAD schemes. Nonetheless, in order to not ignore the leakage energy, the inclusion of the NanGate 45 nm library into our measurement portfolio enables a glimpse into the consumption behaviour caused by high-leakage foundries.

Optimal Unrolling. In the previous section, we briefly mentioned that the energy consumed by an r-round unrolled block cipher due to Banik et al. [18] is given as

$$E = (Ar^2 + Br + C)\left(\left\lceil 1 + \frac{R}{r} \right\rceil\right).$$

If two or more block cipher round functions are connected serially, each transient glitch produced in the first round results in further glitches in the subsequent rounds. Due to this phenomenon, the energy consumption of the second round circuit is generally more than the first. Similarly, if three rounds are connected serially, the third round function circuit is likely to consume more than the second. It was shown in [18], that all other things remaining equal, the power consumption in successive rounds is approximately given by an arithmetic

series. Since the sum of terms of an arithmetic series is a quadratic in number of terms, this is where the quadratic term in the energy consumption comes from. We multiply it by $\lceil 1+\frac{R}{r} \rceil$ because that is the number of clock cycles required to encrypt. For heavyweight round functions like AES, the compounding of glitches across one round to the other increases rapidly. Such circuits would naturally have high values of coefficients for A, B to indicate that energy consumption increases rapidly with increasing r. For lighter round functions like PRESENT, SKINNY and SIMON, the compounding of glitches is not so significant, so it results in lower values for the coefficients A, B. For these circuits, increase in power due to the quadratic term is not higher than the leeway given by the decrease in latency due to the $\lceil 1+\frac{R}{r} \rceil$ term. And it was shown in [18], that for almost all lightweight block ciphers, $r = 2$ is the optimum energy configuration.

The $(Ar^2 + Br + C)$ term is actually the average power consumed by the circuit and is typically output by any standard power compiler engine after inspecting either the switching statistic of every node or the value change dump file that records all the signal transitions in the circuit in a given time period. The term is then multiplied with $\left(\lceil 1+\frac{R}{r} \rceil\right)$ to produce the energy consumed. Consider an example from [18]. The authors had estimated that in the STM 90nm process, the energy consumption of an unrolled implementation of PRESENT followed the expression $(3.15+1.40r+0.795r^2)\cdot\left(1+\lceil\frac{32}{r}\rceil\right)$ pJ. It is elementary to see that $r = 2$ is the minimum of this expression, as for $r = 1,\ 2,\ 3$ the expression evaluates to 176.85, 155.21, 174.06 pJ respectively. Now consider PRESENT used in a mode of operation that employs, some other operations like doubling over a finite field, writing on a register, or Boolean arithmetic, i.e., operations that increase the constant term in the quadratic expression. Suppose that to encrypt 8 blocks of plaintext using the mode requires 10 calls to the block cipher and the extra energy per cycle consumed in the *unrolling-independent* operations is α pJ per cycle. Let's say the mode requires $\left(1+\lceil\frac{32}{r}\rceil\right)$ cycles for the computation (10 block cipher calls). This makes the energy expression for the mode $E(r) = [\alpha + (3.15 + 1.40r + 0.795r^2)]\cdot\left(1+\lceil\frac{32}{r}\rceil\right)\cdot 10$ pJ. If $\alpha \approx 4$ or more, it is now clearly visible that the minima of this expression is $r = 3$, since it evaluates to $3.084, 2.232, 2.220, 2.292$ nJ, for $r = 1,2,3,4$. Thus although, the block cipher itself may be energy-optimal at a particular degree of unrolling, it does not necessarily imply that the mode will also be energy-optimal at the same degree. In fact, this is a phenomenon we have observed for two lightweight modes of operation SUNDAE-GIFT and LOTUS-AEAD. These modes of operation are all based on the GIFT block cipher. Although the block cipher itself is energy-optimal at $r = 2$, the modes are optimal at $r = 3$. Note that this optimum is subject to other operating parameters like choice of library, or the level of compile time optimisation of the circuit, however as all other things remaining same, this observation stands.

To illustrate the point further, we experimented with three non-clock-gated lightweight modes of operation; GIFT-COFB, SUNDAE-GIFT, and LOTUS-AEAD. Table 3.3 illustrates the power consumption breakdown of individual components of the 1, 2 and 3-round unrolled implementations of the modes[17]. Note that the 3-round unrolled implementation uses an

[17] To obtain these figures which illustrate the power consumption of individual circuit elements, we used a different compile directive to the circuit compiler, hence the figures are slightly different from the optimal energy

additional multiplexer to filter signals. Since the total number of rounds in both GIFT-64/128 are not multiples of 3, the signals used to update the state after the execution of 3 rounds in each clock cycle, and the final output of the block cipher are to be tapped from different circuit nodes and hence the need for an extra mux. Note that for this particular implementation, GIFT-COFB and SUNDAE attain optimal energy configuration at $r = 2$, whereas LOTUS-AEAD optimizes at $r = 3$. Take the case of LOTUS-AEAD, in which as the degree of unrolling r increases, the power consumption contribution of the terms depending on r, which are the individual round functions and the incremental components of the state and key registers, increase moderately. This is in contrast to the constant power consumption sources like control system, writing values to various registers and consumption of other gates, all of which increase the constant term in the power consumption. Hence the energy consumed to process eight blocks of plaintext and one block of associated data is around 10.88, 7.20, 6.15 nJ for $r = 1, 2, 3$ respectively. This is not the case for both of these particular implementations of GIFT-COFB or SUNDAE-GIFT, and hence the optimum point remains at $r = 2$.

Table 3.3: Breakdown of power consumptions of three lightweight AEAD schemes using the TSMC 90 nm cell library with a clock frequency of 10 MHz. The designs were synthesised via the *compile_ultra -no_autoungroup* directive.

		μW									
Candidate	r	Key Reg.	State Reg.	Other Regs.	Multiplier	Control	RF1	RF2	RF3	Extra Mux.	Total
GIFT-COFB	1	19.2	22.5	7.3	-	10.2	12.5	-	-	-	71.1
	2	20.1	36.4	3.1	-	14.5	13.8	31.1	-	-	119
	3	20.8	34.3	8.1	-	12.4	17.8	33.9	48.7	13	189
SUNDAE-GIFT	1	19.3	28.7	-	3.0	10.8	13	-	-	-	74.8
	2	20.1	36.4	-	3.1	14.5	13.8	31.1	-	-	119
	3	20.8	45.2	-	3.1	12.1	16.4	31.3	46.7	13.4	189
LOTUS-AEAD	1	22	12.3	44.4	-	17.3	9	-	-	-	105
	2	22.1	13.9	52.4	-	23.9	8.8	17.9	-	-	139
	3	22.2	16.7	45.7	-	20.9	11.5	16.4	25.4	6.2	165

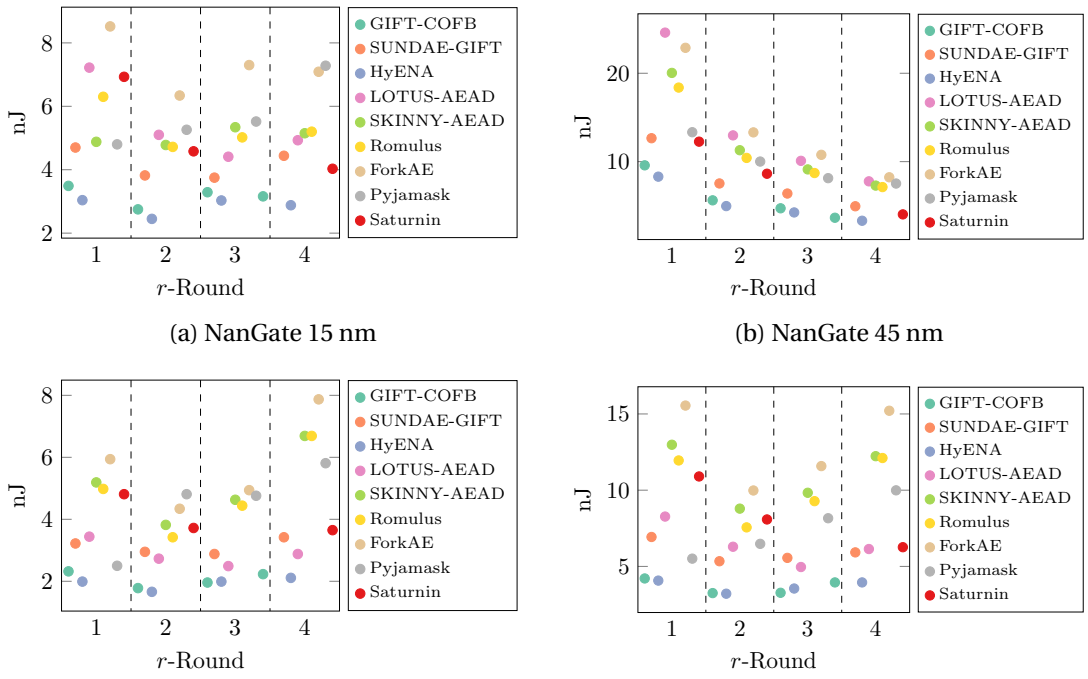
Note that in the presence of large leakage energy the effects of unrolling are delayed meaning that the point of optimality occurs at larger unrolling factors for some schemes than for more leakage-resilient libraries. This is due to the fact that the leakage overhead is only offset when the total time for an encryption is sufficiently small. For instance, the GIFT-COFB circuit attains its most energy-efficient configuration at $r = 4$ when implemented in the leakage-heavy NanGate 45 nm library, which stands in contrast to other cell libraries for which the optimal unrolling degree remains at $r = 2$.

Clock-Gating. We have applied the clock-gating technique only to those implementations which contain idle registers. These are GIFT-COFB, HYENA, LOTUS-AEAD, LOCUS-AEAD, SKINNY-AEAD, ForkAE, Pyjamask, Romulus and Saturnin. As already explained, clock-gating saves energy by preventing unnecessary reloading of registers with the same value, therefore the total energy saving grows proportionally with the total number of clock cycles of an AEAD operation. In other words, the effects of this technique becomes obvious for (1) candi-

figures tabulated later in this text.

dates with more AEAD registers (2) r-round unrolled implementations with small $r \in \{1,2\}$ as they require more clock cycles. For instance, because 1-round unrolled LOTUS-AEAD implementation necessitates more cycles than other schemes, and the design contains a couple of 64-bit registers, this technique saves more than one third in energy. Therefore, as a rule of thumb, clock-gating is a worthwhile effort if the particular design in question contains large number of flip-flops, e.g., registers, that stay frozen over long periods.

Synthesis Results. Figure 3.3 charts the optimal energy value for the encryption of a small message for each r and candidate. The category is dominated by GIFT-COFB and HYENA which both are lightweight in terms of gate count but respond equally well to partial unrolling. A detailed tabulation of all the measurements including gate count, latency and throughput for the NanGate 15 nm and TSMC 90 nm cell libraries can be found in Table 3.5 and Table 3.6. In addition, Appendix A.1 contains the results for the NanGate 45 nm and UMC 65 nm processes.



(a) NanGate 15 nm

(b) NanGate 45 nm

(c) UMC 65 nm

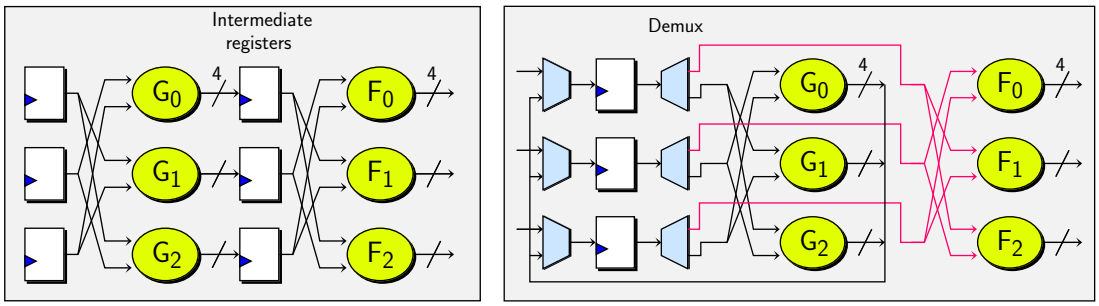
(d) TSMC 90 nm

Figure 3.3: Optimal energy consumption chart of the partially unrolled AEAD circuits measured for the encryption of 1024 bits of associated data and 1024 message bits.

3.4 Threshold Implementations

As elaborated in Section 2.7 that the minimum number of input shares required to implement the first order TI of a function of algebraic degree t is $t+1$. This means that quadratic S-boxes need at least three shares and cubic S-boxes at least four shares first-order TI. However, the larger the number of shares, we proportionally need to scale up the number of registers and

other constituent logic gates in the circuit. Needless to say this comes with proportional scaling up of not only the circuit area but also power and energy consumption of the circuit. Thus at first glance it might appear that, from an energy efficiency point of view, one should rather aim to minimize the number of shares in the circuit. Most lightweight cryptographic S-boxes are of algebraic degree three, e.g., those of PRESENT, GIFT and Midori, and hence for a while it was inconceivable to construct a Threshold Implementation of less than four shares. However, in [112], the authors demonstrated how to construct 3-share TI of a block cipher cubic s-box. In the paper, the authors presented a compact 3-share TI of the PRESENT block cipher. The idea is as follows: although the S-box S of PRESENT is cubic, it can be written as $S = F \circ G$, where F and G are quadratic s-boxes. So a three-share implementation of the PRESENT S-box can be done by implementing the TI of G and F separated by a register bank in between, which suppresses the glitches produced by the TI of G. This approach is schematically depicted in Figure 3.4a.



(a) Cubic TI *with* Register

(b) Cubic TI *without* Register

Figure 3.4: Implementing a Threshold Implementation of a cubic s-box in three shares in two separate ways. Both arrangements require two clock cycles to be computed and thus effectively double the latency of the surrounding block cipher.

The above approach has an obvious disadvantage that additional registers are required in between the G and F layers. However, we can think of an alternate arrangement as shown in Figure 3.4b. The idea is to have a demultiplexer bank in front of the G layer, that switches off the input to this layer in every alternate clock cycle. By doing so, the G-layer outputs once computed are fed back to the register bank. In the next cycle the register feeds the G-layer output through the demux on to the F-layer (shown by the red datapath). Although, this type of a structure does not need an extra register layer, it is counterproductive as far as energy efficiency is concerned (unless there are special algebraic structures/relations between G and F). First, the structure replaces a register with a demultiplexer bank and yet another multiplexer bank that filters the G-layer output back into the register, so as far as circuit area is concerned it offers no real advantages. Secondly, the structure increases the physical length of the datapath in the circuit, and so extra gate delay produced thereof results in additional glitches, which further impacts the energy consumption negatively. Hence, we will not investigate a register-free cubic TI in our evaluation.

Note that the structure of Figure 3.4a has an additional disadvantage that it require two

clock cycles to compute the shared s-box output, whereas a 4-share implementation would require only one cycle. Since time efficiency is also an equally important component of energy efficiency, this implies that it is not immediately evident that a 3-share would beat a 4-share variant, as far as energy consumption of a TI is concerned. To make a fair evaluation of the energy efficiency of the AEAD schemes we implemented first order TI with the following characteristics:

1. We implemented first-order Threshold Implementations of 1-round unrolled circuits. This is necessary because r-round unrolled circuits must necessarily have higher algebraic degree, it will require more shares to construct a TI. For example a 2-round unrolled scheme of a block cipher with a cubic S-box has algebraic degree six, if properly designed and then seven shares are required. The exact algebraic forms of each bit of a 7-share of a first-order TI is likely to be very complicated, due to high degree, with multiple terms in each expression, eventually leading to large costs in area and power.

2. Furthermore, we created TI circuits in which only the state path of the underlying encryption primitive is shared, but not the key path. This design choice is common place in cryptography, as it is adequate for first-order security and for simplicity we follow the suit. If the key paths were also shared, we estimate that it would increase the power consumption of the AEAD schemes by a similar factor, and energy consumption comparisons would probably lead to similar results. In short, we implement threshold circuits for all the AEAD schemes except Saturnin. The mode Saturnin was designed in an unusual way that the output of block cipher is used as the key in the subsequent block cipher call. And so a TI which only considers shares in the data path is not possible for this mode.

S-Box Details. Most of the schemes benchmarked in this work have cubic s-boxes with 4-bit inputs that are decomposable into quadratics of the form $F \circ G$, and so efficient three and four-share implementations are feasible.

- *GIFT.* The S-box of GIFT belongs to the cubic class $\mathscr{C}_{172}$ which is decomposable into two quadratics using a direct sharing approach. The algebraic expressions of the output shares of both the three and four-share TI can be found in [85].

- *SKINNY.* The S-box of SKINNY takes eight input bits and has algebraic degree equal to six. Therefore, a single cycle implementation would require 7 shares. Instead, we implemented a 3-share TI of all SKINNY-based modes based on the recommendation given by the designers in [30]. More specifically, the S-box can be decomposed into $I \circ H \circ G \circ F$ where each of these functions is an 8-bit quadratic function. Hence, the entire substitution layer is computed over four cycles.[18]

Synthesis Results. Table 3.4 lists the simulation results using the same measurement setup as the unshared round-based implementations. It can be seen that the schemes using SKINNY

[18] We will see in Chapter 9 how to decompose the 8-bit SKINNY S-box more efficiently.

exhibit the highest energy consumption, which is intuitive since the S-box needs four clock cycles for evaluation. On the other hand, it is surprising to see that 4-share TI circuits have similar energy efficiency when compared to the corresponding 3-share circuits. This means that the circuit complexity of 4-share implementations are offset by the increased latency of 3-share circuits that take more cycles to complete an invocation.

Table 3.4: Synthesis figures the investigated threshold implementations for the NanGate 15 nm and TSMC 90 nm cell libraries. The energy consumption corresponds to the encryption of 1024 bits of AD and 1024 plaintext bits. The measurements for the remaining libraries are given in Appendix A.2.

Library	Scheme	Shares	Area	Latency	Critical Path	TP	Power	Energy
			GE	Cycles	ns	Mbits/s	μW	nJ
NanGate 15 nm	GIFT-COFB	3	22903	1360	0.27	6037.7	142.8	19.42
	GIFT-COFB	4	40034	680	0.33	9580.8	337.7	22.96
	SUNDAE-GIFT	3	19267	2000	0.13	6250.0	133.6	26.72
	SUNDAE-GIFT	4	35100	1000	0.20	8205.1	322.6	32.26
	HyENA	3	21489	1360	0.15	11034.5	136.4	18.55
	HyENA	4	38167	680	0.21	14953.3	334.8	22.77
	LOTUS-AEAD	3	19923	2856	0.19	8602.2	101.6	29.02
	LOTUS-AEAD	4	29215	1428	0.22	14545.5	267.1	38.14
	SKINNY-AEAD	3	25167	3808	0.15	5228.8	169.2	64.43
	Romulus	3	18982	3877	0.22	3636.4	150.0	58.16
	ForkAE	3	24424	4576	0.18	4444.4	192.3	88.00
TSMC 90 nm	GIFT-COFB	3	16893	1360	4.40	363.6	217.7	29.61
	GIFT-COFB	4	27321	680	7.68	416.7	362.3	24.64
	SUNDAE-GIFT	3	15021	2000	4.37	183.1	221.2	44.24
	SUNDAE-GIFT	4	13294	1000	3.72	430.1	368.9	36.89
	HyENA	3	15888	1360	4.13	387.4	215.5	29.31
	HyENA	4	26137	680	6.22	514.5	346.6	23.57
	LOTUS-AEAD	3	15481	2856	3.81	419.9	118.7	33.90
	LOTUS-AEAD	4	21386	1428	4.83	662.5	298.6	42.64
	SKINNY-AEAD	3	20132	3808	4.18	191.4	266.8	101.60
	Romulus	3	14410	3877	4.60	173.9	213.2	82.66
	ForkAE	3	18853	4576	5.87	136.3	316.4	144.78

3.5 Final Observations

We tried to give a comprehensive guide to designing energy-efficient authenticated encryption schemes by evaluating a selection of ten NIST LWC candidates that make use of a lightweight or a semi-lightweight block cipher at their core. In the process we were able to look at each candidate individually and identify optimal circuit configurations that would reduce the energy consumption of the AEAD circuit as a whole. We were also able to make broader ob-

servations regarding energy efficiency in AEAD modes instantiated with lightweight block ciphers. In the second part of the paper, we turned our attention towards threshold implementations. We looked at both 3-share and 4-share threshold implementations of the schemes and made energy measurements. For schemes based on SKINNY, the fact that four cycles are required to evaluate the shared s-box, means that the AEAD scheme must sacrifice more of its energy for the cipher itself. For the other candidates we note an up to twenty percent decrease in energy consumption for the 4-share implementation in comparison to the 3-share designs. We conclude our paper with the following claims, which applies to block cipher based AEAD paradigm, that can hopefully help achieve the ultimate goal of lightweightness with respect to energy consumption:

1. The size of register banks play an important role, in energy and area of an r-round unrolled AEAD circuit. This is based on the fact that for 1-round unrolling of GIFT-COFB and SUNDAE-GIFT, more than half of the energy is consumed by the registers, even though these two have relatively fewer flip-flops. On the other hand, percentage of energy consumption by registers are much higher for LOTUS-AEAD, because the mode of operation brings many intermediate variables into the circuit which need extra registers to store.

2. The r-round unrolled implementations strike a good balance between area, throughput and the energy consumption. However, the optimal value for r depends both on the block cipher and the surrounding mode of operation. Designers are recommended to experiment with different choices of r for the full AEAD scheme, and keep in mind that experiments based solely on block ciphers are not sufficient.

3. If a given AEAD scheme contains many storage elements, implementors are recommended to employ techniques such as clock-gating as much as possible to reduce the energy consumption. The efficiency of the clock-gating scales up with the number of idle storage elements and the total number of clock cycles during which they remain inactive.

4. From the energy perspective, there is almost a direct correlation between the lightweight nature of non-threshold and threshold implementations of an AEAD scheme. Hence the optimal design choices for TI align well with the aforementioned decisions.

Table 3.5: Synthesis measurements of the investigated AEAD schemes for the NanGate 15 nm cell library. The latency and energy consumption corresponds to the encryption of 1024 bits of AD and 1024 plaintext bits. Auxiliary results for the NanGate 45 nm and UMC 65 nm libraries are given in Appendix A.1.

Cipher	Circuit	Area	Latency	Critical Path	TP	Power	Energy
		GE	Cycles	ns	Mbits/s	μW	nJ
GIFT-COFB	1-Round	6643	680	0.32	10062.9	53.2	3.62

	1-Round-CG	6485	680	0.34	9552.2	51.3	3.49
	2-Round	7395	340	0.40	16120.9	83.1	2.83
	2-Round-CG	7238	340	0.42	15421.7	81.0	2.75
	3-Round	8545	238	0.48	18890.2	140.3	3.34
	3-Round-CG	8387	238	0.49	18812.5	138.1	3.29
	4-Round	8901	170	0.55	23188.4	188.5	3.20
	4-Round-CG	8743	170	0.55	23104.7	185.9	3.16
SUNDAE-GIFT	1-Round	5025	1000	0.17	9523.8	47.0	4.70
	1-Round-CG	-	-	-	-	-	-
	2-Round	5778	500	0.25	13008.1	76.5	3.82
	2-Round-CG	-	-	-	-	-	-
	3-Round	6933	350	0.33	13686.9	107.2	3.75
	3-Round-CG	-	-	-	-	-	-
	4-Round	7284	250	0.40	15880.9	177.5	4.44
	4-Round-CG	-	-	-	-	-	-
HyENA	1-Round	5742	680	0.20	16410.3	46.4	3.16
	1-Round-CG	5585	680	0.19	16842.1	44.7	3.04
	2-Round	6495	340	0.27	23703.7	74.1	2.52
	2-Round-CG	6337	340	0.28	22857.1	72.0	2.45
	3-Round	7629	238	0.36	25682.2	129.7	3.09
	3-Round-CG	7472	238	0.38	24060.2	127.5	3.03
	4-Round	7996	170	0.43	29767.4	171.9	2.92
	4-Round-CG	7843	170	0.44	29090.9	169.5	2.88
LOTUS-AEAD	1-Round	8901	1428	0.21	15037.6	59.9	8.55
	1-Round-CG	8128	1428	0.20	15625.0	50.6	7.22
	2-Round	9359	714	0.31	20874.8	81.0	5.78
	2-Round-CG	8591	714	0.28	22859.6	71.5	5.10
	3-Round	10015	485	0.37	24577.6	101.9	4.94
	3-Round-CG	9974	485	0.38	23997.0	91.0	4.41
	4-Round	10279	357	0.45	28507.8	147.9	5.28
	4-Round-CG	9613	357	0.44	29357.8	138.1	4.93
SKINNY-AEAD	1-Round	10646	952	0.43	7447.1	77.9	7.42
	1-Round-CG	10015	952	0.41	7767.0	51.3	4.88
	2-Round	11785	476	0.53	12075.5	110.3	5.25
	2-Round-CG	11154	476	0.51	12524.5	100.4	4.78
	3-Round	12924	323	0.65	14175.0	171.5	5.54
	3-Round-CG	12293	323	0.64	14219.1	165.4	5.34
	4-Round	14064	238	0.70	18364.4	227.5	5.41
	4-Round-CG	13438	238	0.69	18577.6	216.3	5.15
Romulus	1-Round	8174	970	0.20	16161.6	65.0	6.30
	1-Round-CG	-	-	-	-	-	-
	2-Round	9313	494	0.39	16410.3	95.6	4.72
	2-Round-CG	-	-	-	-	-	-
	3-Round	10854	324	0.50	18470.4	155.0	5.02
	3-Round-CG	-	-	-	-	-	-

	4-Round	11597	256	0.57	22575.0	203.1	5.20
	4-Round-CG	-	-	-	-	-	-
ForkAE	1-Round	10066	1144	0.16	19753.1	82.8	9.47
	1-Round-CG	9323	1144	0.17	19393.9	74.5	8.52
	2-Round	11607	576	0.24	26666.7	118.2	6.81
	2-Round-CG	10864	576	0.24	26556.0	110.0	6.34
	3-Round	12741	384	0.35	25820.0	199.3	7.65
	3-Round-CG	12004	384	0.36	25682.2	190.2	7.30
	4-Round	13875	288	0.40	32080.2	256.6	7.39
	4-Round-CG	13138	288	0.41	31372.5	246.3	7.09
Pyjamask	1-Round	21159	285	0.16	52032.5	177.9	5.07
	1-Round-CG	20182	285	0.17	51097.8	168.4	4.80
	2-Round	26840	152	0.25	64076.9	365.3	5.55
	2-Round-CG	25945	152	0.24	66115.7	345.8	5.26
	3-Round	32511	95	0.28	91454.7	601.8	5.72
	3-Round-CG	31631	95	0.30	86195.3	581.5	5.52
	4-Round	38249	76	0.36	88770.5	1001.3	7.61
	4-Round-CG	37348	76	0.37	87193.5	957.6	7.28
Saturnin	1-Round	18829	285	0.56	15265.4	258.6	7.37
	1-Round-CG	17807	285	0.61	13989.1	243.1	6.93
	2-Round	26683	152	0.28	56939.5	318.5	4.84
	2-Round-CG	25447	152	0.29	55172.4	301.6	4.58
	3-Round	-	-	-	-	-	-
	3-Round-CG	-	-	-	-	-	-
	4-Round	28661	76	0.30	105960.3	559.5	4.25
	4-Round-CG	27430	76	0.31	102893.9	530.6	4.03

Table 3.6: Synthesis measurements of the investigated AEAD schemes for the TSMC 90 nm cell library. The latency and energy consumption corresponds to the encryption of 1024 bits of AD and 1024 plaintext bits.

Cipher	Circuit	Area	Latency	Critical Path	TP	Power	Energy
		GE	Cycles	ns	Mbits/s	μW	nJ
GIFT-COFB	1-Round	4710	680	4.68	683.8	69.3	4.71
	1-Round-CG	4700	680	5.05	633.7	61.9	4.21
	2-Round	5548	340	4.82	1327.8	106.8	3.63
	2-Round-CG	5510	340	6.05	1057.9	95.5	3.25
	3-Round	6372	238	6.78	1348.5	159.0	3.78
	3-Round-CG	6311	238	7.02	1302.4	156.2	3.72
	4-Round	7144	170	8.83	1449.6	237.0	4.03
	4-Round-CG	7036	170	10.10	1267.3	232.4	3.95
SUNDAE-GIFT	1-Round	4710	1000	3.72	430.1	69.3	6.93
	1-Round-CG	-	-	-	-	-	-
	2-Round	5548	500	4.98	642.6	106.8	5.34

	2-Round-CG	-	-	-	-	-	-
	3-Round	6372	350	5.94	769.6	159.0	5.57
	3-Round-CG	-	-	-	-	-	-
	4-Round	7144	250	7.41	863.7	237.0	5.93
	4-Round-CG	-	-	-	-	-	-
HyENA	1-Round	3941	680	3.87	826.9	68.3	4.64
	1-Round-CG	3850	680	4.34	737.3	59.8	4.07
	2-Round	4746	340	5.42	1180.8	97.5	3.32
	2-Round-CG	4787	340	5.39	1187.4	94.4	3.21
	3-Round	5629	238	5.96	1534.0	151.7	3.61
	3-Round-CG	5542	238	5.82	1570.9	149.2	3.55
	4-Round	6327	170	8.08	1584.2	227.2	3.86
	4-Round-CG	6238	170	7.68	1666.7	232.4	3.95
LOTUS-AEAD	1-Round	7362	1428	4.97	643.9	88.1	12.58
	1-Round-CG	6841	1428	4.98	642.6	57.9	8.27
	2-Round	8433	714	6.19	1033.9	108.3	7.73
	2-Round-CG	7618	714	5.90	1084.9	88.1	6.29
	3-Round	9863	485	7.42	1232.2	138.3	6.71
	3-Round-CG	9125	485	7.24	1262.8	102.2	4.96
	4-Round	12082	357	9.24	1385.3	181.5	6.48
	4-Round-CG	11608	357	8.24	1553.4	171.9	6.14
SKINNY-AEAD	1-Round	8011	952	4.17	767.6	159.1	15.15
	1-Round-CG	7451	952	5.14	622.6	136.3	12.98
	2-Round	8701	476	6.37	1004.7	200.9	9.56
	2-Round-CG	8205	476	6.02	1063.3	184.7	8.79
	3-Round	11109	323	8.89	1028.4	320.5	10.35
	3-Round-CG	10546	323	9.35	977.8	304.4	9.83
	4-Round	12890	238	11.89	1076.5	528.4	12.58
	4-Round-CG	12354	238	10.51	1217.9	513.8	12.23
Romulus	1-Round	5279	970	3.49	916.9	123.2	11.95
	1-Round-CG	-	-	-	-	-	-
	2-Round	6315	494	6.69	956.7	153.1	7.56
	2-Round-CG	-	-	-	-	-	-
	3-Round	8960	324	9.57	955.4	286.4	9.28
	3-Round-CG	-	-	-	-	-	-
	4-Round	10398	256	12.78	1001.6	472.9	12.11
	4-Round-CG	-	-	-	-	-	-
ForkAE	1-Round	7362	1144	4.21	760.1	159.4	18.24
	1-Round-CG	6841	1144	4.62	692.6	135.9	15.55
	2-Round	8433	576	6.22	1028.9	198.7	11.45
	2-Round-CG	7618	576	6.33	1011.1	173.3	9.98
	3-Round	9863	384	6.89	1327.0	309.0	11.87
	3-Round-CG	9125	384	7.81	1170.7	301.5	11.58
	4-Round	12082	288	12.49	1024.8	548.7	15.80
	4-Round-CG	11608	288	12.56	1019.1	528.0	15.21

Pyjamask	1-Round	15646	285	3.87	2205.0	221.3	6.31
	1-Round-CG	15158	285	4.31	1979.9	193.3	5.51
	2-Round	19552	152	6.13	2610.1	467.1	7.10
	2-Round-CG	19184	152	6.14	2605.9	426.5	6.48
	3-Round	26707	95	6.99	3662.4	897.4	8.53
	3-Round-CG	26353	95	6.99	3662.4	859.0	8.16
	4-Round	34363	76	8.53	3751.5	1354.9	10.30
	4-Round-CG	34031	76	8.57	3734.0	1315.1	9.99
Saturnin	1-Round	15214	285	6.60	1292.9	413.8	11.79
	1-Round-CG	14540	285	6.77	1260.5	382.6	10.90
	2-Round	20530	152	4.23	3782.5	564.0	8.57
	2-Round-CG	19184	152	4.29	3729.6	531.3	8.08
	3-Round	-	-	-	-	-	-
	3-Round-CG	-	-	-	-	-	-
	4-Round	22895	76	7.16	4469.3	858.1	6.52
	4-Round-CG	22160	76	7.67	4172.1	823.3	6.26

[...] 'Cause I walk onto the water buzzing with electrolytes and sacrilege [...]

4 Perfect Trees

[...] Banked on memory [...]

In this second work, we divert our attention away from modes of operation and onto stream ciphers. As indicated in Chapter 1, no energy model similar to the quasi-quadratic polynomial equation for block cipher circuits proposed by Banik et al. [18] is known in the realm of stream ciphers. However, in a follow-up work, Banik et al. [25] make some broader conclusions about the effects of unrolling stream cipher circuits were made. They show that an unrolled stream cipher circuit that produces multiple keystream bits in one clock cycle is more energy-efficient in an asymptotic sense, i.e., when the encryption of multiple data blocks is considered instead of a single block. In fact, it was shown that for over 320 bits of data, Trivium consumed the least amount of energy on STM 90 nm ASIC circuits and outperformed the Midori block cipher family. For asymptotically large amount of data, the regular Trivium circuit reached its point of optimality relatively late at $r = 160$, and at this degree of unrolling it was around 9 times more energy-efficient than 64-bit Midori. These findings are reflected in Figure 4.1, indicating that the baseline Trivium design is a fitting starting point from which new low-energy constructions can be derived. The findings presented in this chapter were published in the fourth issue of Transactions on Symmetric Cryptology (IACR-ToSC) in 2021.

The reasons why a heuristic energy model for stream ciphers appears to be harder to conceive are manifold. For one, stream ciphers circuits are often not more than a single large register bank whose outputs are fed into a thin combinatorial layer, e.g., in Trivium the state update function only consists of 12 two-input logic gates. This means that for small r the energy consumption of the algorithm is almost entirely determined by the storage elements, i.e., the contribution of the round function circuit is insignificant. Further note that when r is small the switching activity of the state update function heavily depends on the underlying cell library process and can thus vary widely. Only for large r the energy consumption of the round function layer renders itself decisive, however it becomes increasingly complex to reason about the circuit as the algebraic complexity of the underlying equations grows unmanageable, thus preventing any deeper analysis of the involved switching activity. This stands in contrast to block ciphers where the unrolling factor r is usually small and thus the complexity of the round function circuits remains bounded.

Analogously, the reasons why some hardware stream ciphers outperform block ciphers

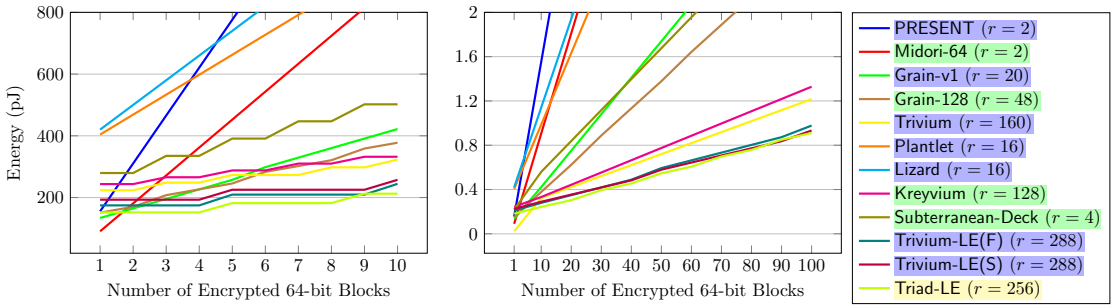


Figure 4.1: Energy consumption (pJ) chart from Banik et al. [25] using the STM 90 nm cell library process at a clock frequency of 10 MHz. Added to the plot are figures for the energy consumptions of Subterranean-Deck, and the designs Trivium-LE(F), Trivium-LE(S), Triad-LE that we propose in this paper, for the same standard cell library and operating frequency. Figures are reported for short messages (1 to 10 blocks of 64-bits) and longer messages (1-100 blocks). Legend entries highlighted in blue and green have a security level of 80 and 128 bits respectively, whereas Triad-LE offers 112-bit security.

in energy efficiency are also many. Most hardware stream ciphers (like Trivium and the Grain family) are designed with a few register locations at the beginning being untapped, i.e., not used in register update. This allows for efficient hardware unrolling, so that, unlike block ciphers, each individual round in these stream ciphers can be implemented in parallel and hence does not increase the circuit depth. As such, the glitches produced in the circuit of round i do not increase the glitches in round $i+1$, at least when the circuit is unrolled for small values of r. Perhaps the most important reason is that stream ciphers perform the key-IV setup only once and then are able to encrypt multiple bits of data without having to do it again. For example, an implementation of Trivium that is unrolled $r = 128$ times, would only need $\frac{1152}{128} = 9$ clock cycles to complete key-IV setup and takes 100 more cycles to encrypt up to 12800 bits of data. The most energy-efficient implementation of 64-bit Midori (at degree of unrolling $r = 2$), needs eight cycles to encrypt every 64-bit block of data, and hence would need $\frac{8*12800}{64} = 1600$ cycles to encrypt the same length of data which is around fifteen times more. Consequently, lightweight stream ciphers are preferable when factors like energy and throughput are concerned.

Contribution. In this chapter, we investigate unrolled stream cipher constructions and make some fundamental discoveries about their energy consumption behaviour. More specifically, our contributions can be summarised as follows:

1. *Perfect Tree Energy Model.* Our first contribution in this paper is to re-implement r-round unrolled stream cipher circuits in a generic more energy-efficient manner. We shall define shortly the concept of a *circuit strand*, which basically comprises of the logic functions involved in one register update. We demonstrate that rather than following the approach in [25], if we adopt a technique in which each strand is implemented separately as a unit and the circuit synthesizer is prevented from performing any *inter-strand* optimisation, then the power consumption increases in a slower man-

ner with the respect to the degree of unrolling r. Trivium is especially suited for this restricted mode of compilation and reaches its point of optimality in the fully unrolled setting at $r = 288$. This optimal energy is significantly lower than the 160-round circuit reported in [25] under the same operating environment.

This tessellation enables us to partition the entire circuit into smaller units which are obviously the strands. Since these are interconnected, it gives rise to a natural tree structure among them in the following way: a strand j is a child node of strand i, if the output of j is one of the inputs of i. Hereafter, by observing the variation of the power consumption in these strands, it is possible to deduce a strong correlation between the power consumed by each strand its position in the above tree, which leads to the definition of a tree-based metric that correlates the energy consumption to a wide range of stream ciphers, namely:

(a) Trivium-like constructions [61] in which the register output tap locations are chosen randomly.

(b) Trivium-like constructions proposed in the literature that have some structural differences in comparison to the original Trivium design. These include the modified Trivium proposed in [102], TriviA [54], Kreyvium [52] and Triad-SC [24].

(c) Algebraically more complex ciphers with large state update functions such as Grain-128 [77].

(d) Subterranean-like constructions [59], which do not exhibit rotating state registers.

Thus this leads to the proposal of the fist formal energy model for stream cipher constructions akin to that for block ciphers in [18].

2. *New Energy-Optimal Stream Ciphers.* By leveraging the obtained energy model, we are able to show that register tap positions significantly affect the energy efficiency. Hence, our next attempt is to design new energy-optimal ciphers, where our approach is to change the register tap positions of the original Trivium cipher. However, the change of the register tap positions also affects the security, and we carefully chose these positions without decreasing the claimed security level, i.e., the 80-bit security of Trivium. We present two candidates, which we call Trivium-LE(F) and Trivium-LE(S), that consume around 10-15% and 25% less energy than Trivium, respectively. Note that Trivium-LE(F) is conservative with enough security margin, and Trivium-LE(S) is challenging with a thin security margin. As shown in Figure 4.1, both constructions stand as the currently most energy-efficient encryption primitives in the literature when at least 24 bytes are encrypted. The energy efficiency of Trivium-LE(F) outperforms known ciphers, and the structure is also useful to design an energy-efficient message authentication code. We present Trivium-LE-MAC whose update function inherits to Trivium-LE(F) but the message is absorbed instead of key-stream generation.

   It is important to note that our model makes it, for the first time, possible to design stream ciphers for hardware environments that are specifically optimized in terms of

energy consumption as the metric is both simple and widely applicable. We also applied the same strategy to Triad-SC, which supports 112-bit security, because it seems to be the most promising for the energy efficiency due to the shorter state size of 256 bits. By altering tap locations, we present one candidate, which we call Triad-LE, that lower the energy consumption than the original Triad-SC.

At this point, we remark that previous major works in the field of energy efficiency [13, 18, 25] were limited in their approach in the sense that their findings were restricted to a 90 nm standard cell library and energy was computed at 10 MHz throughout. This was feasible as 90 nm standard cells have very low leakage and at a frequency of 10 MHz or higher the contribution of the leakage energy to the total energy consumption was minimal. Since the dynamic component of the energy is constant with respect to frequency, as a result, at all frequencies upwards of 1 MHz the energy consumption was more or less constant (see [18, Fig. 1]). However, we present our findings for four different standard cell libraries in which the underlying transistors have sizes 90 nm (TSMC), 65 nm (UMC), 45 nm and 15 nm (NanGate) respectively and therefore we do not ignore leakage energy. Although for presentability, we report results at certain fixed frequencies for each library, primarily to bring out the dynamic part of it, the energy trends that we present hold across libraries and a wide range of clock frequencies, and we argue that convincingly in the paper. When this is not possible, for space constraints, the results are reported at clock frequency 10 MHz for the TSMC 90 nm and UMC 65 nm libraries and at 1 GHz for the NanGate libraries. This is done so that the dynamic energy component is the dominant contributor of the total energy consumption (for better comparison with [13, 18, 25, 89]). All energy figures are reported for encryption of 1.28 Mbits of data and are generated after a timing simulation of several thousand test vectors on the corresponding netlist post-synthesis.

Outline. In Section 4.1, we present the effects that different compiler directives used to synthesize stream cipher circuits have on the energy consumption. Section 4.2 details the obtained heuristic energy model. In Section 4.3, we propose energy-optimal Trivium variants and an energy-efficient message authentication code. Subsequently, in Section 4.4 , we study recent Trivium-like, Grain-like and Subterranean-like constructions proposed in the literature and show that our derived energy model works for these designs too. The paper is then concluded in Section 4.5.

4.1 Restricted Circuits

Combinatorially heavy circuits, such as the increasingly complex algebraic representation of state update equations in r-round unrolled stream ciphers, induce synthesis tools to produce optimized architectures in terms of circuit area. They also introduce a gap when it comes to reasoning about the overall energy consumption, which is significantly hindered as the synthesized circuits have mutated into opaque, garbled constructions.

We find that imposing a regular structure which is exclusively composed of simple combinatorial logic gates in which the state update function is replicated unaltered across differ-

ent r in an unrolled setting yields equivalent if not better power figures for basic as well as more feature-rich standard cell libraries when compared to the highly optimized circuits of the Synopsys Design Compiler synthesis tool. In the following, we define one such structure as stated below:

Definition 2 (Strand)**.** *Recall the Trivium state update function that is composed of three independent logic blocks of the form whose inputs are tapped from the 288-bit state register $x_0, x_1, \ldots, x_{287}$ such that*

$$\begin{aligned} t_0 &\leftarrow x_{65} \oplus (x_{90} \wedge x_{91}) \oplus x_{92} \oplus x_{170} & (x'_0, \ldots, x'_{92}) &\leftarrow (t_2, x_0, \ldots, x_{91}) \\ t_1 &\leftarrow x_{161} \oplus (x_{174} \wedge x_{175}) \oplus x_{176} \oplus x_{263} & (x'_{93}, \ldots, x'_{176}) &\leftarrow (t_0, x_{93}, \ldots, x_{175}) \\ t_2 &\leftarrow x_{242} \oplus (x_{285} \wedge x_{286}) \oplus x_{287} \oplus x_{68} & (x'_{177}, \ldots, x'_{287}) &\leftarrow (t_1, x_{177}, \ldots, x_{286}). \end{aligned}$$

We define each individual logic block as a strand of the following form:

$$a \oplus b \oplus (c \wedge d) \oplus e.$$

A feature-rich library with 3-pin linear cells can implement one strand with 3 gates (1 NAND2, 1 XNOR2, 1 XNOR3), hence the entire Trivium combinatorial layer then consists of 10 gates in total (9 for the 3 strands and one 3-input XOR gate for the output function). A simpler library that only consists of 2-pin linear logic elements such as the NanGate cell library family requires 14 gates for the combinatorial layer. A full description of Trivium is given in Section 2.6.1.

In this respect, we investigate several circuit and compilation directives supported by the Synopsys Design Compiler:

- *Regular.* The entire circuit is compiled with the regular compile command which moderately attempts to optimize the synthesis result. In this setup, the synthesizer is free to choose the mapping and the corresponding optimisation. The compiler may choose to not respect the boundaries between two strands and make any optimisation it deems fit. This is actually equivalent to the implementation strategy of [25], i.e., in which the compiler has the freedom to optimize given the logical representation of the update function.

- *Restricted.* Same compilation directive as in the regular configuration, i.e., compile, however the synthesis of the state update function is restricted to the logical mapping, where the state update circuit for $r = 1$ is simply replicated for higher degrees of unrolling. Under this directive, the compiler puts together each strand separately and is forced to respect the boundaries between 2 strands. Thus when used as such, the compiled circuit consists of exactly $3r$ strands for an r-round unrolled construction.

- *Ultra.* The circuit is synthesized using compile_ultra directive which is a high-effort routine that optimizes beyond the entity boundaries and often yields the most area- and latency-efficient constructions. Here too, the compiler may choose not to respect strand boundaries.

One of our empirical findings is that for Trivium circuits compiled under the *Restricted* directive, the increase in the power consumption (for encrypting a given number of data blocks) is much slower (with respect to the degree of unrolling r) than circuits compiled under the *Regular* or *Ultra* directives.[19] Note that a more fundamental answer to the question whether the energy figures increase or decrease when a cipher is further unrolled is directly linked to its latency and power consumption.

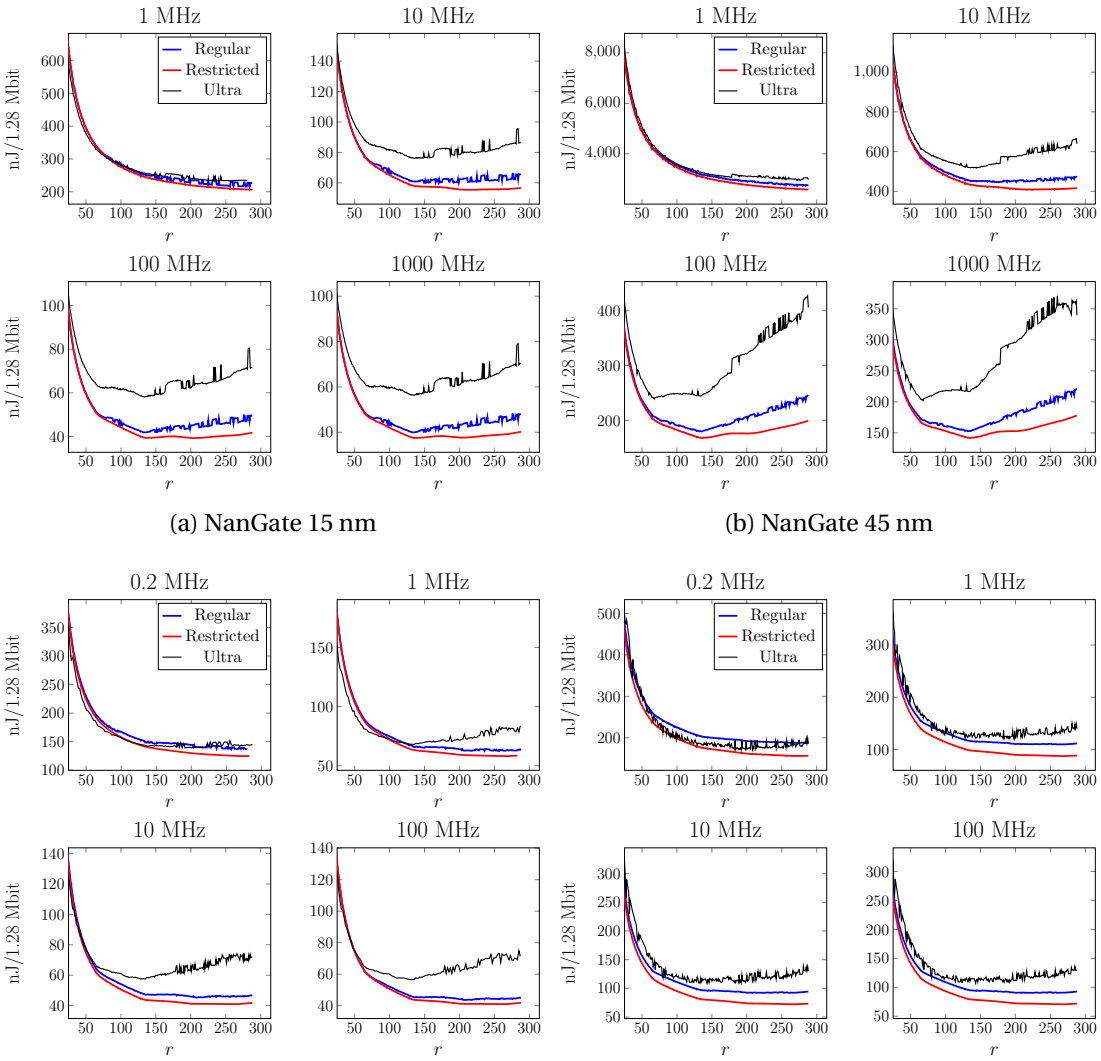
Let $L(r)$ be the total number of clock cycles required to encrypt a fixed-size plaintext block in the r-round unrolled setting and denote by $P(r)$ and $E(r)$ the power and energy values respectively. It is crucial to note that $L(r)$ will decrease and consequently $P(r)$ will increase as r increases and thus the value of r which minimizes $E(r) = P(r) \cdot L(r)$ (this is true for block ciphers too) was exactly the problem studied for block ciphers in [18] and for stream ciphers in [25]. In Figure 4.2 and Figure 4.3, we detail the energy and area simulation results for four standard cell libraries (TSMC 90 nm, UMC 65 nm and NanGate 45 and 15 nm) over a wide range of frequencies. The choice of frequencies was indeed library specific: so that the critical path of the circuit was well below the clock period even when the circuit was fully unrolled. This obviates the need for the compiler to use higher drive strength based cells just to get a positive slack in order to ensure clock period larger than critical path, which alters the basic character of the circuit for different values of r and prevents a fair evaluation. Hence for the faster NanGate library based circuits we used the frequency range 1 MHz to 1 GHz, and for the other libraries we used the range 0.2 MHz to 100 MHz. We find that the circuits compiled in the restricted mode are by far the most energy-efficient of the three. Its energy consumption more or less decreases monotonously for $r \geq 150$, which suggests that if r is allowed to vary up to 288, then the fully unrolled cipher, i.e., $r = 288$, is the best setup for energy constrained environments (though not always). This empirical observation naturally allows us to segue into the next round of results in Section 4.2 where we look more closely at the circuits compiled under the restricted mode.

4.2 Perfect Tree Energy Model

For the remaining experiments, we look to investigate unrolled Trivium circuits with $r = 288$ since they achieve maximum throughput and deliver close to the best energy efficiency for all libraries across a wide range of frequencies. Though it was theoretically possible to unroll more, it would require more silicon area and improve energy efficiency only fractionally more. Since the circuits are compiled in restricted mode, it is possible to see how much power each strand consumes. We commence by introducing some notations and definitions that will help us formalise the write-up better.

Definition 3 (i-th Strand)**.** *Denote by $t_i(r)$ the strand for equation t_i in the r-th unrolled round*

[19] One reason for this is that with the other compiler directives, the main optimisation effort goes behind reducing area of the circuit and meeting timing slack requirements. And the result it blurs the boundaries between individual strands and is thus not necessarily power-optimal.



(a) NanGate 15 nm

(b) NanGate 45 nm

(c) UMC 65 nm

(d) TSMC 90 nm

Figure 4.2: Trivium energy measurements for the three synthesis settings for different frequencies and libraries. Note that energy graphs are noisier for the regular/ultra modes which indicates that the synthesizer chooses different mapping strategies for varying r.

with $i \in \{0,1,2\}$ *and* $r \in \{0,\ldots,287\}$ *such that each successive* $t_i(r)$ *can be recursively defined as:*

$$t_1(r) = t_3(r-66) \oplus t_3(r-93) \oplus [t_3(r-91) \wedge t_2(r-92)] \oplus t_0(r-78)$$
$$t_2(r) = t_1(r-69) \oplus t_1(r-84) \oplus [t_1(r-82) \wedge t_0(r-83)] \oplus t_1(r-87)$$
$$t_3(r) = t_2(r-66) \oplus t_1(r-111) \oplus [t_1(r-109) \wedge t_1(r-110)] \oplus t_2(r-69),$$

where $t_1(r) = x_{94-r}$, $t_2(r) = x_{178-r}$ *and* $t_3(r) = x_{1-r}$ *whenever* $r \leq 0$.

Figure 4.4 shows the power consumed in each of the strands $t_i(r)$ for increasing values of r for two of the libraries we experiment with in this paper. We had expected the power

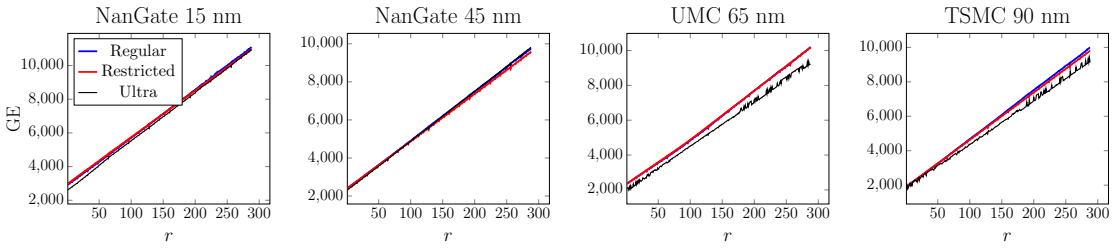
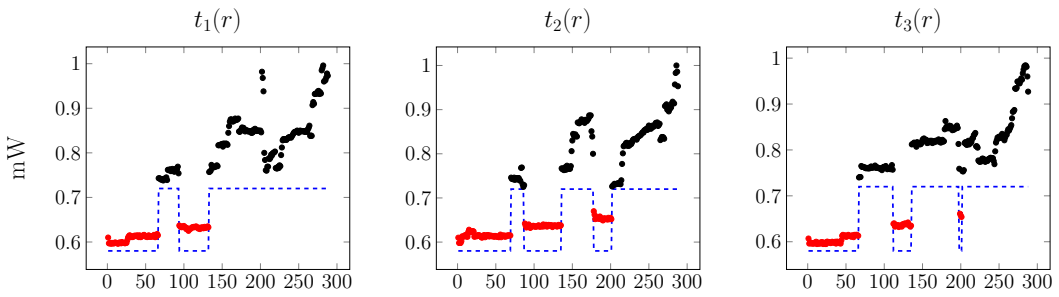


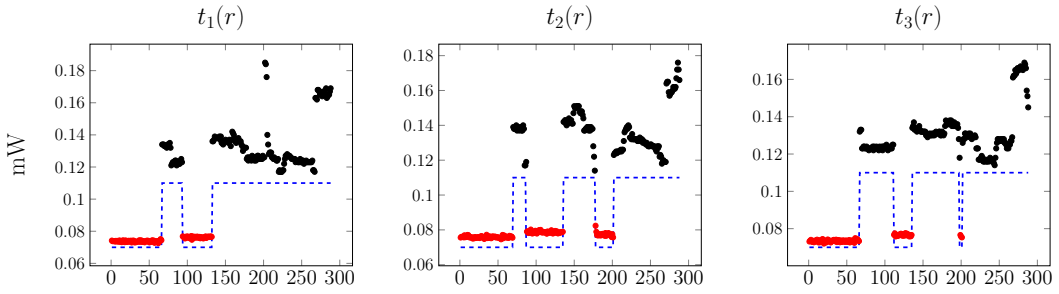
Figure 4.3: Trivium area measurements (Gate Equivalent) for the three synthesis settings for all unrolling factors r and cell libraries. Note that the number of clock cycles that are required in order to encrypt x bits of data is given by $\lceil \frac{1152}{r} \rceil + \lceil \frac{x}{r} \rceil$, hence the encryption of 1.28 MBits of data for $r = 288$ has a latency of 4449 cycles.

in the strands to increase monotonously with r as in block ciphers, but the figure clearly suggests that the increase is far from monotonous. The red marks represent the strands whose power consumption experiences a sudden dip. This observation seemed at first to be counter-intuitive, and so we set about trying to understand this curious phenomenon. We first observed that all $t_1(r)$'s (for $1 \leq r \leq 66$) consume the same power until $t_1(67)$ whose power consumption is considerably larger (note the red to black jump in Figure 4.4 around $r = 66$ for $t_1(r)$ for all the libraries). All inputs to $t_1(r)$ (for $1 \leq r \leq 66$) come directly from the register. Thus in some sense their input nodes are all at a distance zero from the register. However, one of the inputs of $t_1(67)$ comes from the output of $t_3(1)$ and thus not all its inputs are at distance zero from the register. This delay imbalance in the input wires gives rise to more glitches in the internal circuitry of $t_1(67)$ and this hints at one of the reasons why it consumes more. Further consider the boundary around $r = 93$. At $r = 94$, the power consumption of $t_1(94)$ drops. It is easy to see that all the inputs of $t_1(94)$ are at distance two from the register, whereas the inputs of $t_1(93)$ are still unbalanced with respect to the delay from the register. This led us to believe that delay imbalance plays a major role in determining how much power the strands consume.

Through the Looking Glass. In order to verify the above phenomenon, we looked at the internal timing diagrams of both the strand pairs (a) $t_1(66)$ and $t_1(67)$, and (b) $t_1(93)$ and $t_1(94)$, presented in Figure 4.5 (the circuit was synthesized using NanGate 45 nm cell library and clocked at 1 GHz). Let us examine $t_1(66)$. The first 2 input pins x_1, x_{28}, according to the circuit synthesizer, have an average delay of 0.09 ns from the clock edge at which the new inputs are written on to the registers. As a result, the output of the first XOR gate in the strand i.e., $x_1 \oplus x_{28}$ is only moderately glitchy. Over 4450 clock cycles this net switches logic only 2271 times, as found by a post-synthesis timing simulation on the netlist. On the other hand, in $t_1(67)$, x_{27} is at a delay 0.09 ns whereas the other input $t_3(1)$ is at an average delay 0.25 ns. The output of the corresponding XOR gate $x_{27} \oplus t_3(1)$ is glitchier as compared to $x_1 \oplus x_{28}$, it switches 4512 times in the same interval. This clearly indicates that $t_1(67)$ consumes more power. Conversely, consider $t_1(93)$. The first 2 input pins x_1, $t_3(27)$ have delays 0.09 ns and 0.25 ns from the clock edge. Hence the net $x_1 \oplus t_3(27)$ switches around 4665 times in the same interval. However, in $t_1(94)$, the pins $t_3(1)$, $t_3(28)$ have delays 0.24 ns and 0.25 ns. Hence,



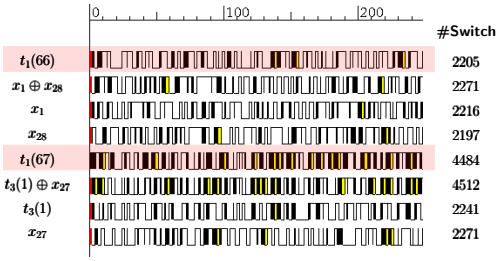
(a) NanGate 45 nm



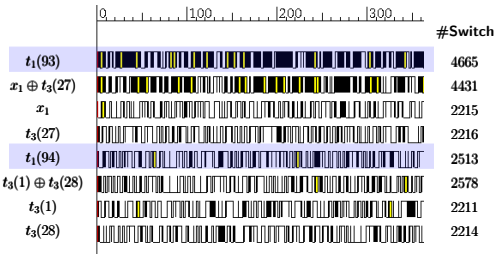
(b) TSMC 90 nm

Figure 4.4: Power measurements for all the perfect unrolled strand trees for two cell library processes. The red data points indicate unrolled strand equations which correspond to perfect trees. The dashed blue line signifies the transition boundaries between perfect and imperfect trees, i.e., low points represent perfect unrolled strand trees while high points correspond to imperfect trees.

many of the glitches produced by them cancel out and the XOR net $t_3(1) \oplus t_3(28)$ switches 2551 times in this interval. This indicates that depth-balanced strands consume less power than unbalanced ones.



(a)



(b)

Figure 4.5: Timing diagrams for internals in (a) $t_1(66)$, $t_1(67)$, and (b) $t_1(93)$, $t_1(94)$.

4.2.1 Circuit to Tree

In order to formalise the above phenomenon, we found that the circuit strands are connected naturally in a well-defined graphical topology. Each unrolled strand can be translated into a

5-ary tree with the root node as the output bit whose subtrees are other unrolled strand trees or leaf nodes.

Definition 4 (Unrolled Strand Tree)**.** *Let $T_i(r)$ be the 5-ary unrolled strand tree corresponding to the unrolled strand equation $t_i(r)$. The child nodes of the strand $T_i(r)$ are therefore all the 5 nodes $T_j(u)$ for which the corresponding terms $t_j(u)$ are present in its recursive definition as per Definition 3.*

Example 5. *To make the link between unrolled strand equations, and their respective trees clearer, we give 3 examples of varying complexity. The unrolled strand trees $T_3(1)$, $T_3(100)$ alongside $T_3(200)$ are displayed in Figure 4.6. Note that terms that appear several times in an unrolled strand equation result in duplicate nodes in the corresponding unrolled strand tree. This is to ensure that the equations are a one-to-one representation of the actual circuit.*

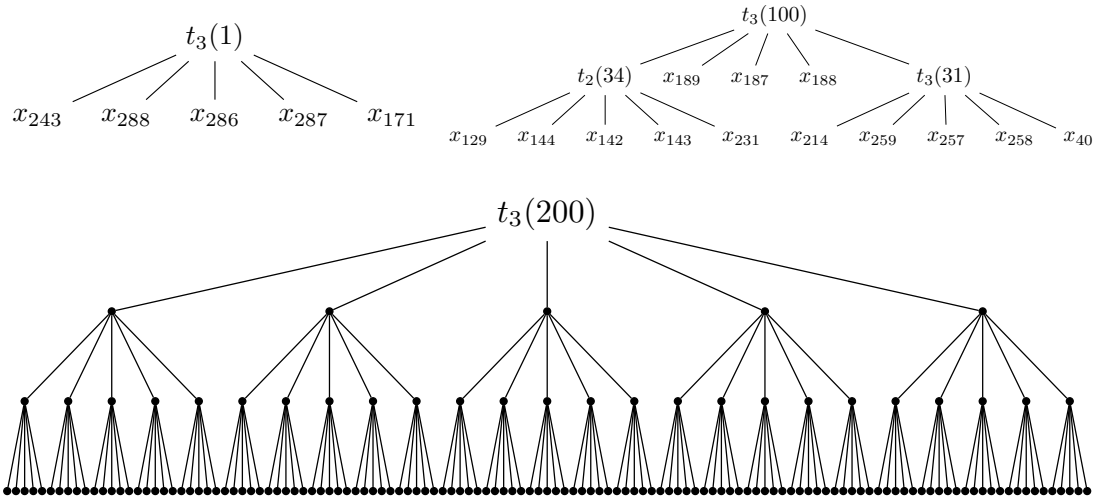


Figure 4.6: The strand trees $T_3(1)$, $T_3(100)$ and $T_3(200)$. $T_3(1)$, $T_3(200)$ are perfect.

We can further classify our unrolled strand trees as either perfect or imperfect according to the following definitions.

Definition 5 (Perfect m-ary Tree)**.** *A perfect m-ary tree is a tree in which all non-leaf nodes have m children and all leaf nodes are at the same depth.*

Clearly, the unrolled strand trees in Trivium are 5-ary. Further, remark that in Figure 4.6, $T_3(1)$ and $T_3(200)$ are perfect unrolled strand trees while $T_3(100)$ is imperfect due to having leaf nodes at different depths. In the example from the previous section clearly, $T_1(66)$, $T_1(94)$ were perfect trees whereas $T_1(67)$, $T_1(93)$ were not. This gives us a very good understanding of the power consumption of strands vis-à-vis the position of the corresponding nodes in the circuit tree graph. A strand evidently consumes less power if the node it occupies in the circuit graph houses a perfect tree.

Let us try to argue this inductively. A tree is 5-ary perfect if and only if all of its five child nodes are also perfect. Thus it is easy to see that in a perfect tree all its input nodes are at

approximately the same average delay from the register. This being so all perfect trees tend to consume less power. On the other hand a tree is imperfect if and only if one of its child nodes is also imperfect, due to which the gate output corresponding to this imperfect child node is considerably more glitchy. This excess glitch from the child node would naturally be carried forward in the parent strand making its output glitchier and thus causing it to consume more dynamic power. This observation naturally leads us to the next question: is it possible to have a general Trivium-like stream cipher (with tap locations perhaps different from the original Trivium specifications) that is more energy-efficient and also secure at the same time? The translation of circuit to an equivalent algebraic topology may have given us a quick way to check this. Since perfect trees consume less dynamic power, a variant of Trivium (with different tap locations) is likely to consume less energy if its circuit tree graph has a larger total number of perfect trees.

Let us provide more arguments as to why the above makes sense. Consider two configurations of Trivium: Trivium-A and Trivium-B with different tap locations (both synthesized in restricted mode). At a degree of unrolling equal to 288, the circuits of both these variants consist of exactly the same amount of gates and flip-flops. Since the leakage power in a circuit depends directly on the total silicon area, both these circuits are likely to consume the same leakage power. Furthermore, the circuit graphs of both these variants have exactly the same amount of nodes. If for example Trivium-A has more perfect trees in the graph than Trivium-B, then it automatically implies that Trivium-A has fewer imperfect trees than Trivium-B, which more or less implies that Trivium-A is likely to be the variant that consumes less dynamic power. Since the leakage power is the same, this means that the Trivium-A consumes less total power and hence less total energy. This of course should hold irrespective of the standard cell library used to synthesize the circuit or the frequency of signal used to clock the circuit.

We can estimate the total number of perfect trees in a generic Trivium configuration. To ease notation we will denote the total number of perfect trees among all strands $t_i(r)$ as $S(T_i)$ such that the total number of perfect trees in the circuit is $S(T) = \sum_i S(T_i)$. More formally, let f be a function from the set of all trees to $\{0,1\}$ such that $f(T_i(r)) = 1$ if and only if $T_i(r)$ is a perfect tree, and is 0 otherwise: then $S(T_i) = \sum_r f(T_i(r))$. Below, we report the distribution of perfect unrolled strand trees in the original Trivium. In Trivium, we have $S(T_1) = 105$, $S(T_2) = 144$, $S(T_3) = 93$, and hence $S(T) = 339$. Note that there are no perfect unrolled strand trees of depth four or larger.

4.2.2 Enumerating Perfect Trees

In the following, let us consider a generic Trivium layout in order to determine configurations that yield a high number of perfect trees and consequently lower the power consumption.

Definition 6. *Denote by Trivium(X, n) a generic Trivium configuration composed of n chained registers $(X_1, \ldots, X_n)$ such that X_j^{ℓ} is the j^{th} register's leftmost forward tap, X_j^{f} is the feedback tap and X_j^{op} is the output tap. See Figure 4.7 for a schematic depiction. Note that X_j^{op} is essentially the final tap location of the j^{th} register (this is required to ensure the one-to-one*

nature of the Trivium update). The figure does not explicitly show the taps for the AND gates, as we will show that if both the AND taps are between X_j^ℓ and X_j^{op} then it does not affect the total number of perfect trees in the circuit graph.

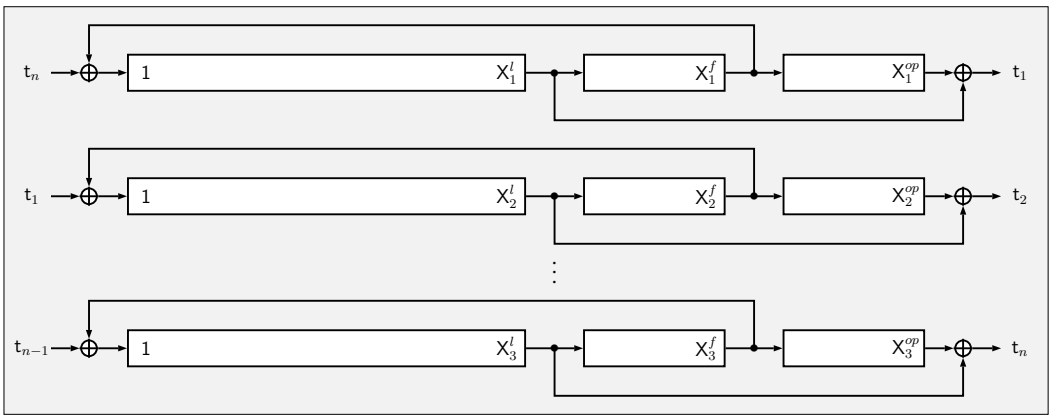


Figure 4.7: Generic Trivium(X,n) configuration of n chained registers.

Note that this notation corresponds to n update function strands hence the unrolled strand tree of $t_j(r)$ is $T_j(r)$.

Example 6. *The original Trivium specification composed of three update function strands is congruent to Trivium$(X,3)$ where $X_1^\ell = 66$, $X_1^f = 69$, $X_1^{op} = 93$, $X_2^\ell = 69$, $X_2^f = 78$, $X_2^{op} = 84$, $X_3^\ell = 66$, $X_3^f = 87$, $X_3^{op} = 111$ with an additional non-linear gate between the leftmost and output tap in each register.*

Finding configurations that lead to an increased number of perfect trees seems non-trivial as the search space is enormous. Additionally, a closed-form solution that evaluates the exact number of perfect trees for a given circuit Trivium(X,n) appears equally hard. A brute-force solution consists of individually creating the unrolled strand tree for each equation and checking that all leaf nodes are at the same distance from the root. However, this approach is expensive and hard to optimize apart from ordinary parallelisations. Nevertheless, transcribing the problem into a recurrence relation offers some remedy to this issue.

Lemma 1. *Given an arbitrary, generic Trivium(X,n) circuit composed of n registers, the total number of perfect unrolled strand trees $S(T)$ in the fully unrolled setting is given by $S(T) = \sum_{j=1}^{n} S(T_j) = \sum_{j=1}^{n}\sum_{l=1}^{n}\left(g_l(X_j) - f_l(X_j)\right)^+$, where $y^+ = \max\{y,0\}$ and $f_l(X_j)$, $g_l(X_j)$ are recursively defined functions for $1 \le l \le n$ of the form*

$$
\begin{aligned}
f_l(X_j) &= \max\left\{f_{l-1}(X_{j-1}) + X_j^{op},\ f_{l-1}(X_j) + X_{j+1}^{f}\right\}\\
g_l(X_j) &= \min\left\{f_{l-1}(X_{j-1}) + X_j^{op} + \left[\left(g_{l-1}(X_{j-1}) - f_{l-1}(X_{j-1})\right) - (X_j^{op} - X_j^{\ell})\right]^+,\right.\\
&\qquad\left. g_{l-1}(X_j) + X_{j+1}^{f}\right\},
\end{aligned}
$$

such that $f_1(X_j) = 0$ *and* $g_1(X_j) = \min\left\{X_j^{\ell}, X_{j+1}^{f}\right\}$. *The number of perfect trees of depth* t *for the* j*-th strand is* $S(T_j)|_{depth=t} = \left(g_t(X_j) - f_t(X_j)\right)^+$. *Hence the total number of trees of all depths is* $S(T_j) = \sum_{l=1}^{n}\left(g_l(X_j) - f_l(X_j)\right)^+$ *and thus the lemma follows.*

We remark that since there are n registers indexed 1 to n the value of $j+1$ (resp. $j-1$) refers to addition (resp. subtraction) mod n in the set $\{1,2,\cdots,n\}$. Further note that a tree is perfect if and only if all its subtrees are perfect.

Proof. (Intuition) From Figure 4.8, we can see that there are certain values of r for which the circuit for $t_j(r)$ produces a perfect depth-1, depth-2 tree etc. We define two families of functions f_t, g_t such that $f_t(X_j)+1$ is the minimum value of r for which $t_j(r)$ corresponds to a perfect depth t tree, and similarly $g_t(X_j)$ is the maximum such value of r. It stands to reason that the total number of depth t trees produced in this range of r is $g_t(X_j) - f_t(X_j)$. Note that, obviously if $g_t(X_j) \leq f_t(X_j)$ for some t then there do not exist any depth t trees. It remains to show that f_t and g_t can be recursively defined. The full proof is considerably involved and is given in Appendix B.1.

To conclude let us argue why the number of perfect trees is independent of the AND gate taps as long as they are to the right of the leftmost tap X_j^{ℓ}. It is intuitively not difficult to reason why and let us argue with the help of our previous example: $t_1(66)$ corresponds to a perfect tree but $t_1(67)$ is imperfect. This is because in the process of unrolling $X_1^{\ell}+1$ is the first value of r at which $t_1(r)$ no longer takes inputs directly from the register. Thereafter, it does not matter where exactly the AND taps are as long as they are to the right of X_1^{ℓ}: all subsequent values of r until X_1^{op} continue to produce imperfect trees. □

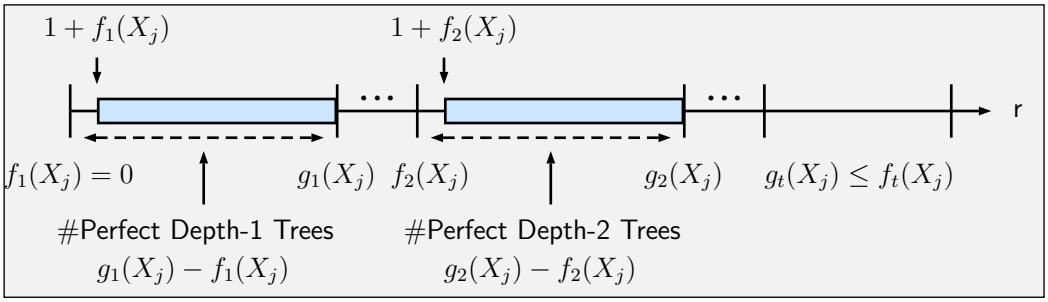


Figure 4.8: Illustration of finite depth trees in Trivium circuit.

Verification. In order to verify our hypothesis (at least empirically) that (a) the number of perfect trees is actually a good indicator of the energy consumption of a generalized Trivium circuit, and (b) that the above holds irrespective of the cell library used to construct the circuit or frequency of the signal used to clock it, we performed an extensive simulation experiment. We generated a large number of Trivium circuits with random taps and calculated the number of perfect trees with the help of the recursion formula given above. We synthesized each circuit in restricted mode using the four cell libraries used in all of our experiments and computed the total power consumed at a wide range of frequencies. The results are plotted in

Figure 4.9. Not only is there a strong negative correlation between the power consumed (and hence energy) and the number of perfect trees, the results hold across libraries and clock frequencies as claimed in Section 4.2.1. For each cell library the same trend is visible across all frequencies. Similarly, since the leakage power of each random Trivium instance is the same and frequency independent (say it is equal to P_l), and since decreasing the frequency (alt. increasing the clock period by ΔT) only increases the physical time required for encrypting a fixed size plaintext block by an amount proportional to ΔT, hence it follows that the leakage energy of each Trivium instance increases by an amount proportional to $P_l \cdot \Delta T$ when the frequency is decreased. Since the dynamic energy is frequency independent, hence all other things remaining the same, when only the frequency is varied, it is equivalent to translating each energy scatter plot by a constant amount along the Y(energy)-axis.

Note that even for configurations with same number of perfect trees, there may be a slight variation in energy consumption, but this variation is negligible as the number of perfect trees increase. This really depends on how badly the imperfect trees are configured in the graph, i.e., configurations with large number of trees with wide variation of delays at their input nodes tend to consume more energy. To model such situations when the number of perfect trees is small, one can think of secondary metrics like the distribution $D(x)$ of number of trees where the absolute difference of the maximum and minimum depths of leaves in the tree is equal to x (note $D(0)$ is the number of perfect trees). It is easy to see that configurations for which $D(x)$ is lower for higher values of x (i.e. lesser number of highly imbalanced trees) are better for energy. Also note that the graph tells us that to get any significant decrease in energy consumption over the original specifications of Trivium (around 10-20%) one needs at least 500 perfect trees.

4.3 Energy-Optimal Variants of Trivium

Before we start to look for more energy-efficient Trivium configurations with more perfect trees, let us once again look at the recursion relationship we have just stated. Note that most perfect trees are at depth 1. In order to increase the number of degree-1 perfect trees, it is obvious that we need to have higher values of $g_1(X_j) = \min\{X_j^\ell, X_{j+1}^f\}$, i.e., each tap location should be chosen towards the end of the register. Naturally, it is not possible to choose each tap location only energy efficiency reasons as the new configuration must be as secure as the original Trivium. Since the search space is large, we decided to follow the following criteria, inherited from the original specification:

1. The linear tap locations X_i^ℓ, X_i^f and X_i^{op} for all i, are chosen from the multiple of 3. In other words, X_i^ℓ, X_i^f, and X_i^{op} are divisible by 3 for all i.

2. The locations of AND gates are fixed such that these two inputs are not divisible by 3. In Trivium, $X_i^{op} - 1$, $X_i^{op} - 2$ are chosen for all i. However, as discussed in the previous section, the impact on the energy consumption is negligible as long as the number of perfect trees is the same. Therefore, we change the AND location to $X_i^\ell + 1$, $X_i^\ell + 2$. Then, the number of perfect trees never changes, and the number of times that AND

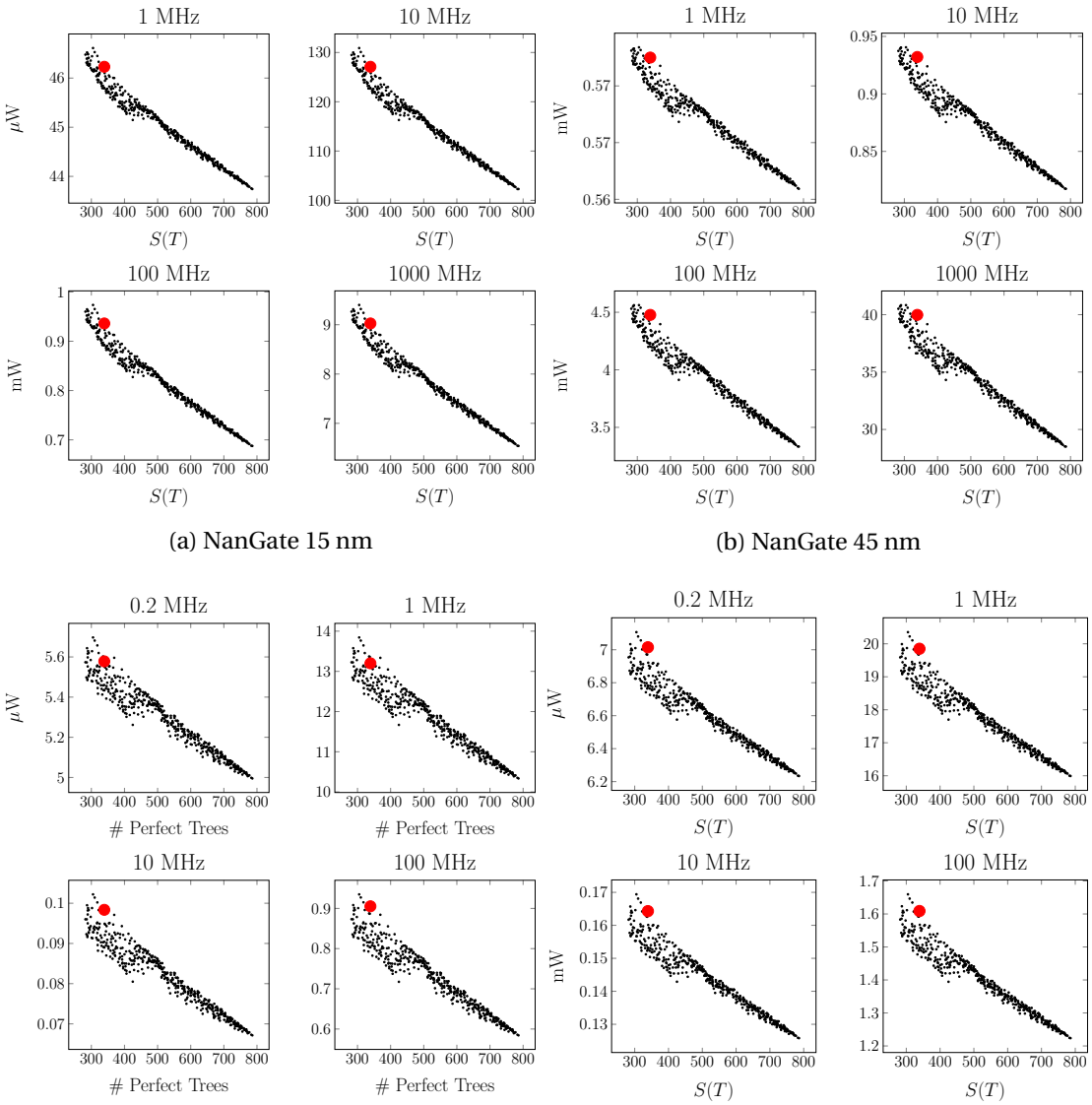


Figure 4.9: Power measurements of several Trivium$(X,3)$ circuits vs $S(T)$ for different libraries and frequencies. The red data points signify the power consumption of original Trivium.

gates are applied increases according to the increase of the number of rounds. Thus, this choice is profitable for the security without increasing the energy consumption.

3. Each tap location for X_i^ℓ and X_i^f is larger than 64 such that a 64× parallel implementation is possible in the software.

4. Under the condition where the output of each AND gate is approximated to 0, we denote by ϵ the maximum correlation in a linear combination of keystream bits. In Trivium, $\epsilon = 2^{-72}$, but it is quite robust against linear attacks because at least 2^{144} keystream bits are required. For a cipher targeting 80-bit security, $\epsilon \leq 2^{-40}$ is necessary.

In particular, 1. and 2. are two of the most important criteria in the design philosophy of Trivium. Thanks to them, we can expect that ϵ in criterion 4. is the highest correlation even when the condition where the output of each AND gate is approximated to 0 is removed. It is primarily because of the following reason. Under parameters following 1. and 2., the whole cipher is divided into three sub-ciphers, and each sub-cipher is only connected non-linearly. In 4., we first evaluate the correlation under the restriction, where the output of AND gate is always approximated to 0. In other words, only one sub-cipher is active, and the other two are inactive. Of course, this restriction is not exhaustive. However, intuitively, we are unlikely to find a better distinguisher beyond this restriction. Because, if at least one output of AND gate $x \cdot y$ is approximated to x, y, or $x + y$ instead of 0, it implies at least two sub ciphers are active. It intuitively increases the number of active AND gates and makes constructing linear distinguishers with high correlation much harder.

Table 4.1: List of configurations and associated security parameters. ϵ represents maximum linear bias. T is the complexity of guess-and-determine attack. c represents an additional cost required to do Gaussian elimination to solve a set of linear equations to recover the internal state. The first row represents the parameters for the original Trivium.

#	Parameters									# Perfect Trees	Max Bias	G&D Complexity
	X_1^ℓ	X_1^f	X_1^{op}	X_2^ℓ	X_2^f	X_2^{op}	X_3^ℓ	X_3^f	X_3^{op}		$-\log_2 \epsilon$	$\log_2 T$
1	66	69	93	69	78	84	66	87	111	339	72	$\log_2 c + 83.5781$
2	87	78	93	66	90	99	75	87	96	495	72	$\log_2 c + 81.3796$
3	87	81	90	81	96	102	69	87	96	534	68	$\log_2 c + 80.0959$
4	75	84	87	84	81	105	81	90	96	570	64	$\log_2 c + 79.0486$
5	75	90	93	90	87	96	78	93	99	591	60	$\log_2 c + 78.8501$
6	81	90	93	90	84	96	78	93	99	624	56	$\log_2 c + 77.8853$
7	81	90	93	90	84	99	81	93	96	642	52	$\log_2 c + 77.2073$
8	75	90	93	96	87	99	87	93	96	666	48	$\log_2 c + 77.0503$
9	84	90	96	90	87	96	87	93	96	699	40	$\log_2 c + 75.8174$

As a result, three criteria 1., 2., and 3. allow us to reduce the number of candidates to $28534800 \approx 2^{24.8}$, and exhaustive search is possible. We exhaustively searched for the best candidates, i.e., the number of perfect trees is maximized, for correlation

$$\epsilon \in \{2^{-72}, 2^{-68}, 2^{-64}, 2^{-60}, 2^{-56}, 2^{-52}, 2^{-48}, 2^{-44}, 2^{-40}\}.$$

In Table 4.1, we list the best candidates for each ϵ. In addition, we also applied Maximov and Biryukov's Guess-and-Determine attack [102] on each of the candidates and list the result. In this attack, the weakness of the multiple-of-3 choice is exploited, and this attack shows Trivium has 80-bit security but it does not have 128-bit security even if the key length is simply extended to 128 bits. Note that this attack has many parameters and scenarios. The complexity listed in Table 4.1 is the so-called scenario T1, i.e., the time complexity is minimized

under the condition that solving only a linear system is enough to recover the key.

It is clear from the table that an increase in the number of perfect trees is generally accompanied by an increased maximum linear bias and decrease in the complexity of the Guess-and-Determine attack. Considering $c \approx 2^{16}$, all parameters would have 80-bit security, but the security margin is very marginal for parameters whose ϵ is close to 2^{-40}. The parameter in row 2 is the best one whose correlation is as low as the original Trivium, but the number of perfect trees is not over 500.

4.3.1 Trivium-LE(F)

As a good alternative which is almost equivalently secure as the original Trivium, the second row of Table 4.1 gives us the parameter set

$$(X_1^\ell, X_1^f, X_1^{op}\ ;\ X_2^\ell, X_2^f, X_2^{op}\ ;\ X_3^\ell, X_3^f, X_3^{op}) = (87,78,93\ ;\ 66,90,99\ ;\ 75,87,96).$$

A graphical depiction of those parameters is given in Figure 4.10.

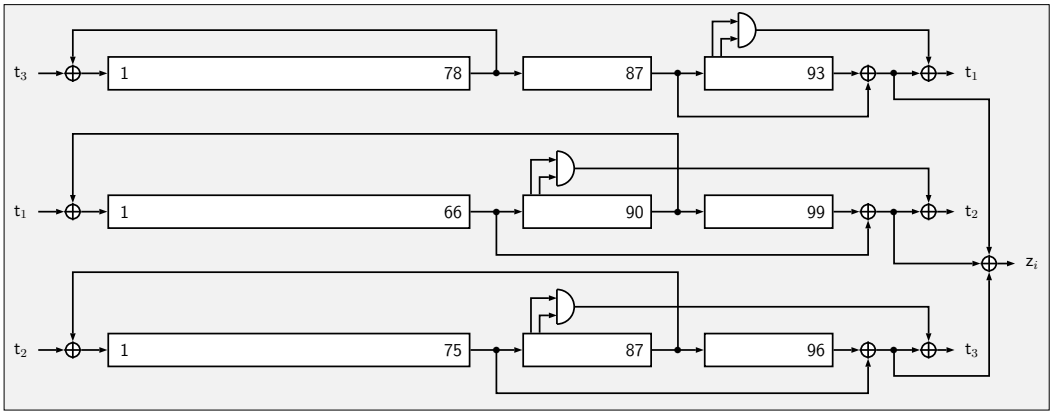


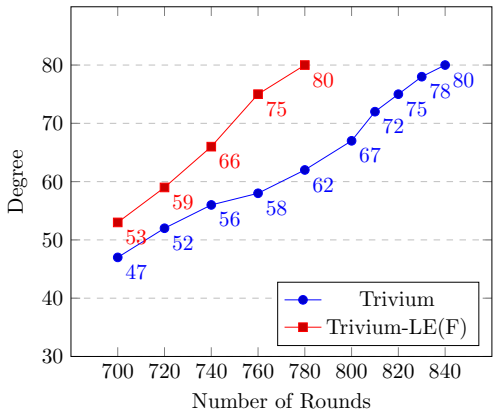
Figure 4.10: Update function and key generation functions of Trivium-LE(F).

This choice gives us a decrease in energy of around 15% over the original Trivium and still provides us with some headway over the margins of security. We therefore propose this parameter set as a more energy-efficient variant of Trivium and call it Trivium-LE(F). We keep the key-IV setup and initialisation routines for Trivium-LE(F) same as Trivium. For completeness, we round off this section with a preliminary security analysis. Since only the tap locations are modified, all types of attacks against Trivium can be applied against Trivium-LE(F). Three important attacks against are discussed below:

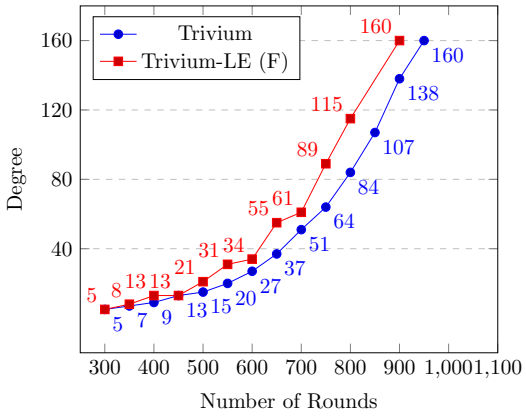
Linear Distinguishing Attack. In order to achieve 80-bit security, there should not be linear distinguishers whose correlation is higher than 2^{-40}. As we already discussed in the section before, the best correlation is 2^{-72} when outputs of AND gates are approximated to 0. While it is unlikely to find better distinguishers due to the multiple-of-3 property, we heuristically evaluated the case where these outputs are not approximated to 0. As we expected, we could not find better linear distinguishers with correlation higher than 2^{-72}.

Maximov and Biryukov's Guess-and-Determine Attack. This attack technique primarily exploits the multiple-of-3 property of Trivium, and it should be effective because Trivium-LE(F) also inherits the multiple-of-3 property. This attack first divides the internal state into three sub states and consists of two phases. In the first phase, we first guess one of three sub states at some time. In the second phase, assuming that the sub state is guessed correctly, we next recover the rest of the bits, in other words $288 \times 2/3 = 192$ bits. Then, we guess outputs of any AND gates and collect keystream bits, which are linearly represented by the internal state. In the so-called scenario T0, no output of any AND gates is guessed. When we use T0 to attack Trivium-LE(F), the time complexity is $c \cdot 2^{74.0}$, which is the same as the attack against Trivium in the same scenario. However, only 96 linear equations are collected for the second phase and it is not enough to recover the remaining 192 bits. Thus, we need to solve a nonlinear system but an efficient algorithm is not known. In scenario T1, outputs of some AND gates are guessed to collect enough linear equations to recover the remaining 192 bits. When we use T1 to attack Trivium-LE(F), the time complexity is $c \cdot 2^{81.3796}$, where 48, 45, and 44 outputs of AND gates are guessed for each register. Then, we can collect 192 linear equations for the second phase, and an efficient algorithm such as the Gaussian elimination is available. Considering $c \approx 2^{16}$, Trivium-LE(F) is secure enough against this attack.

Cube Attack. Unlike the attacks above, the target of the cube attack is the initialisation phase of the cipher. The cube attack was initially introduced in [64]. The original attack was experimental and its aim was to find linear or quadratic superpolies. However, after the division-property based cube attack was proposed [122], the theoretical security estimation is possible, and nowadays, the best cube attacks against Trivium are based on the division-property based method [76, 125]. Cube attacks exploit low algebraic degree in the initialisation. The first keystream bit is regarded as the output of the Boolean function $f_K(\mathrm{IV})$. To execute cube attacks, the superpoly has to be recovered, and it becomes impossible several rounds after the degree of $f_K(\mathrm{IV})$ reaches 80. In practice, the best cube attack against Trivium is 842 rounds [76, 83], and the degree reaches 80 in 840 rounds. Therefore, we first investigated the algebraic degree on $f_K(\mathrm{IV})$ by using the bit-based division property [121, 124] and the left plot in Figure 4.11 shows the increase in the upper bound of the algebraic degree. Thanks to changing the location of AND gates, the algebraic degree of Trivium-LE(F) increases faster than Trivium, and the degree reach 80 in 780 rounds. Moreover, to conservatively evaluate the degree of the superpoly to be high enough, we also investigated the upper bound of the algebraic degree on $f(K,\mathrm{IV})$ by using the bit-based division property. The right plot in Figure 4.11 shows the increase in the upper bound of the algebraic degree. About 900 rounds show the upper bound is full, i.e., 160, and it implies that the degree of the superpoly is unlikely to be lower even if we use the 80-dimensional cube. In both cases, $f_K(\mathrm{IV})$ and $f(K,\mathrm{IV})$, Trivium-LE(F) is more secure than Trivium against cube attacks. Thus, we conclude that Trivium-LE(F) has a large security margin against cube attacks.



(a) Upper bound on the degree of $f_k(iv)$



(b) Upper bound on the degree of $f(k,iv)$

Figure 4.11: Increase in the algebraic degree in Trivium-LE(F) with respect to the number of initialisation rounds.

4.3.2 Trivium-LE(S)

We suggest another variant Trivium-LE(S) that results in around 25% lower energy when compared with Trivium.[20] This variant is based on the 8th row of Table 4.1, and uses the parameter set

$$(X_1^\ell, X_1^f, X_1^{op}\ ;\ X_2^\ell, X_2^f, X_2^{op}\ ;\ X_3^\ell, X_3^f, X_3^{op}) = (96,87,99\ ;\ 87,93,96\ ;\ 75,90,93).$$

We have found that this cipher is algebraically weaker than Trivium, in as much as the algebraic degree of its output bit increases more slowly. It needs 1050 and 1200 rounds to reach the upper bounds of the degree of $f_K(\mathrm{IV})$ and $f(K,\mathrm{IV})$ be the full, respectively. Compared to the original Trivium, the increase of the degree is about 25% slower. Therefore we suggest that for a safe security margin, the number of initialisation rounds used with this variant is $288 \times 5 = 1440$. Note that in terms of an 288 times unrolled circuit, this variant only takes 1 extra clock cycle to initialize, and so asymptotically speaking the energy consumption does not increase due to this extra cycle. For space constraints we defer the security analysis to Appendix B.2.

4.3.3 Trivium-LE-MAC

In addition to the stream ciphers Trivium-LE(F) and Trivium-LE(S), we also propose a message authentication code (MAC) scheme with the name Trivium-LE-MAC, which is designed by slightly modifying the round function of Trivium-LE(F). In order to realise a MAC scheme whose energy consumption is competitive with the stream cipher Trivium-LE(F), it should absorb a 1-bit message into the internal state every round function. While the easiest method is simply XORing the 1-bit message with any 1 bit in the internal state, it is not secure enough

[20] Note that the (F) and (S) stand for Fast and Slow respectively. This is because the (S) variant uses a larger number of initialisation rounds.

against forgery attacks. To guarantee forgery security, we evaluated the lower bound in the number of active AND gates with an MILP-based method when two different messages are absorbed. After exhaustive experimentation we found that a 1-bit message has to be XORed to at least 3 positions of the internal to be secure against forgery attacks. For example, one possible choice is to XOR the 1-bit message with three output bits of state update function, i.e., t_1, t_2, t_3.

From an energy perspective, it is advisable that these injections take place as close as possible to the registers inputs, i.e., to locations $a_{u_1}, b_{u_2}, c_{u_3}$ for smaller values of u_1, u_2, u_3. If we model the message inputs as zero-depth nodes, then it makes each strand $t_i(r)$ correspond to 6-ary trees. It is, for example, easy to see that the first $1 \leq r \leq X_i^\ell - u_i$ strand trees for $t_i(r)$ are all depth 1 perfect 6-ary trees. Hence lower values of u_i intuitively make sense. On the other hand, we chose the injected positions by respecting the multiple-of-3 property to efficiently evaluate the resistance against forgery attacks with an MILP-based method. Specifically, in the constructed model, the message difference is only allowed to be injected at clock cycles $3j_1 + j_0$ ($j_1 \geq 0$) when the non-zero difference is first introduced at the clock j_0. Moreover, the output difference of the active AND gate is always assumed to be 0. The goal is to minimize the number of active AND gates in a trail available for the forgery attack, i.e., the difference of the whole internal state becomes zero after a certain number of clocks. We evaluated all possible candidates of the three injected positions $(a_{1+3i_0}, b_{1+3i_1}, c_{1+3i_2})$ where $0 \leq i_0 \leq 30$, $0 \leq i_1 \leq 32$ and $0 \leq i_2 \leq 31$. When the total distance from the first bit of each register is smaller than 69, among all the candidates, the maximal number of active AND gates is 72. Thus, we choose the best candidate (a_1, b_7, c_1) which reaches 72 active AND gates while achieving the smallest total distance of 6.

Algorithm 5 shows the specifications. Note that Trivium-LE-MAC inherits the security level of Trivium-LE(F) against any key-recovery attack, i.e., 80-bit security. However, the tag length is at most 64 bits. In other words, the security level of the integrity is at most 64 bits.

4.4 Generalisation to Other Stream Ciphers

In this section, we study four Trivium-like ciphers. We will show that the circuit tree phenomenon translates seamlessly onto these more complex Trivium variations.

Trivium-MB. This construction was proposed by Maximov and Biryukov [102] and adds an additional noise term, i.e., a two-input AND gate, to each strand equation t_1, t_2, t_3 which is then backward connected to the register's input. The keystream output function remains unchanged. We denote it by Trivium-MB. Each register update function t_i is of the type

$$x_a + x_b + (x_c \cdot x_d) + (x_e \cdot x_f) + x_g,$$

implying that each $T_i(r)$ has seven child nodes instead of five.

TriviA. The stream cipher by Chakraborti et al. [54] was used in the authenticated encryption scheme of the same name. It also features a key size of 128-bit alongside a 96-bit initialisation vector but exhibits an increased 384-bit internal state partitioned into three chunks of sizes

Algorithm 5 Trivium-LE-MAC

function $\mathtt{R}(A,B,C,m)$
  $z \leftarrow a_{87} \oplus a_{93} \oplus b_{66} \oplus b_{99} \oplus c_{75} \oplus c_{96}$
  $t_1 \leftarrow a_{87} \oplus a_{93} \oplus a_{88} \cdot a_{89} \oplus b_{90}$
  $t_2 \leftarrow b_{66} \oplus b_{99} \oplus b_{67} \cdot b_{68} \oplus c_{87} \oplus m$
  $t_3 \leftarrow c_{75} \oplus c_{96} \oplus c_{76} \cdot c_{77} \oplus a_{78} \oplus m$
  $b_6 \leftarrow b_6 \oplus m$
  $(a_1, a_2, \ldots, a_{93}) \leftarrow (t_3, a_1, \ldots, a_{92})$
  $(b_1, b_2, \ldots, b_{99}) \leftarrow (t_1, b_1, \ldots, b_{98})$
  $(c_1, c_2, \ldots, c_{96}) \leftarrow (t_2, c_1, \ldots, c_{95})$
  return (A,B,C,z)

function $\overline{\mathtt{P}}(A,B,C)$
  $(A,B,C,z) \leftarrow \mathtt{R}(A,B,C,1)$
  for $i = 2$ to 1152 **do**
    $(A,B,C,z) \leftarrow \mathtt{R}(A,B,C,0)$
  return (A,B,C)

function $\mathtt{Load}(N,K)$
  $A \leftarrow (k_1, \ldots, k_{80}, 0, 0, \ldots, 0)$
  $B \leftarrow (n_1, \ldots, n_{80}, 0, 0, \ldots, 0)$
  $C \leftarrow (0, \ldots, 0, 1, 1, 1)$
  return (A,B,C)

function $\mathtt{Trivium\text{-}LE\text{-}MAC}(N,K,M)$
  $(A,B,C) \leftarrow \overline{\mathtt{P}}(\mathtt{Load}(N,K))$
  $len \leftarrow$ the bit length of M.
  for $i = 1$ to len **do**
    $(A,B,C,z) \leftarrow \mathtt{R}(A,B,C,m_i)$
  $(A,B,C) \leftarrow \overline{\mathtt{P}}(A,B,C)$
  for $i = 1$ to 64 **do**
    $(A,B,C,t_i) \leftarrow \mathtt{R}(A,B,C,0)$
  return $\boldsymbol{T}$

132, 105 and 147. Unlike Trivium and Maximov and Biryukov's construction, it adds a non-linear term to the keystream function in the form of a two-input AND gate. Each node in the circuit graph of TriviA also has five child nodes as in Trivium.

Kreyvium. Kreyvium is a stream cipher designed by Canteaut et al. [52] explicitly for the use in fully homomorphic encryption schemes. The cipher has the same structure and tap locations as Trivium, however a 128-bit security is achieved by additionally XORing bits from the key and IV to the update functions. It was shown in [25] that Kreyvium circuits are most energy-efficient at degrees of unrolling that are multiples of 128. This is because the circuit does not require additional shift registers to rotate the key and IV bits to produce the required bits in the update function. In our experiments, the cipher is unrolled 256 times.

Triad-SC. This construction was proposed by Banik et al. [24] as a low-energy alternative to Trivium. It has a much smaller state size (256 bits) and aims to provide 112-bit security. It counters the guess-and-determine attacks by using one additional AND gate compared to the original architecture of Trivium. The update functions in Triad-SC are asymmetric: t_1 is of the form $x_a + x_b + (x_c \cdot x_d) + (x_e \cdot x_f) + x_g$ whereas t_2, t_3 are of the form $x_a + x_b + (x_c \cdot x_d) + x_e$. Hence, $T_1(r)$'s are 7-ary trees and T_2, $T_3(r)$'s are 5-ary trees.

We performed a similar experiment for all the above ciphers as for the original Trivium: (a) We synthesized the circuit in restricted mode and record the power consumed in each strand. Results presented in Figure 4.13 show that strands associated with perfect trees consume much less power that the strands with imperfect trees. And (b) we generated numerous random instances of these ciphers with different tap locations. For every instance we plot power consumed vs number of perfect trees in the circuit tree graph. The results are plotted in Figure 4.12. The results are indeed on expected lines: there is strong negative correlation between energy consumed and number of perfect trees. For space constraints, in the fig-

ure we plot results only for the TSMC 90 nm cell library (with power measured at 10 MHz), however extrapolating the results from Section 4.2 we can make a similar argument that the results are neither library nor frequency specific.

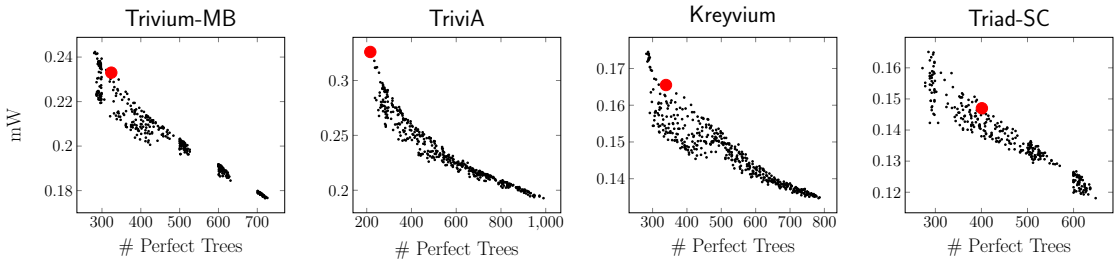


Figure 4.12: Power consumption figures as a function of the number of perfect trees for Trivum-like schemes. All data was obtained using the TSMC 90 nm process at a clock frequency of 10 MHz. Red data points mark the original schemes.

4.4.1 Applicability to Grain-128

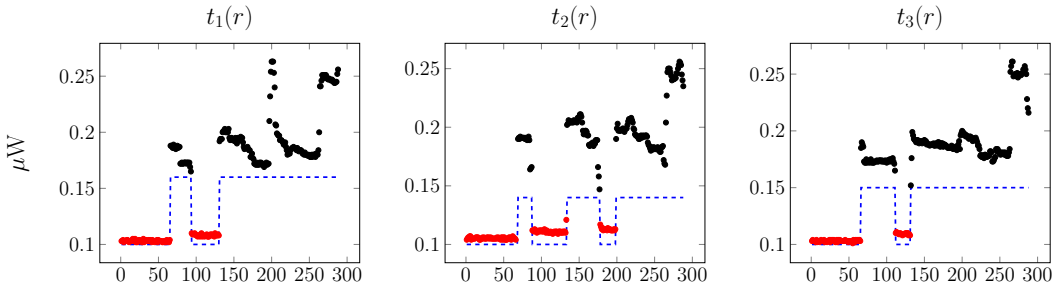
The concept of a circuit strand seamlessly translates over to Grain-128 [77] in which the round function consists of two distinct strands that update two registers $b_1, b_2, \ldots, b_{128}$ and $s_1, s_2, \ldots, s_{128}$ such that

$$(x_1, \ldots, x_{128}) \leftarrow (f, x_1, \ldots, x_{127})$$
$$(y_1, \ldots, y_{128}) \leftarrow (g, y_1, \ldots, y_{127}),$$

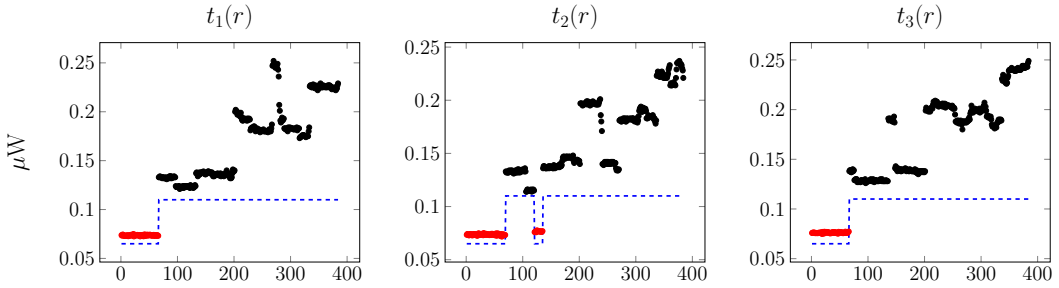
where f and g are linear and non-linear functions respectively defined as follows:

$$\begin{aligned} f &= x_{128} + x_{121} + x_{90} + x_{58} + x_{47} + x_{32}, \\ g &= x_{128} + y_{128} + y_{102} + y_{72} + y_{37} + y_{32} + y_{44}y_{60} + y_{61}y_{125} \\ &\quad + y_{63}y_{67} + y_{69}y_{101} + y_{80}y_{88} + y_{110}y_{111} + y_{115}y_{117} \end{aligned}$$

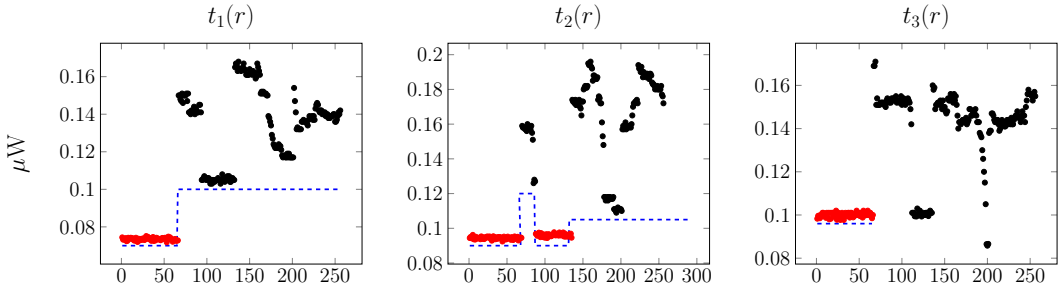
Evidently, the complexity of the update function in this family is higher than in Trivium and so finding a sensible restricted circuit configuration for these complex strands is a harder task. Even if we define the strand as a sub-circuit for the f, g functions, it is not immediately clear which configuration of gates is the best way to construct each strand. We can however delegate this responsibility to the circuit compiler, so that in the restricted mode it still respects the boundary between the strands, but chooses the internal structure of the strand independently. In Figure 4.14, we repeat the experiments from Section 4.1 by letting the synthesizer choose the circuit for each individual strand of f and g in *Restricted* mode instead of imposing a predefined set of logic gates. The results in Figure 4.14 show that for Grain-128 suggests that restricted mode performs at least on par with other synthesis modes indicating that further optimisations for the restricted mode are possible by finding better circuit configurations. Additionally, by repeating the experiments from Section 4.2, we observe



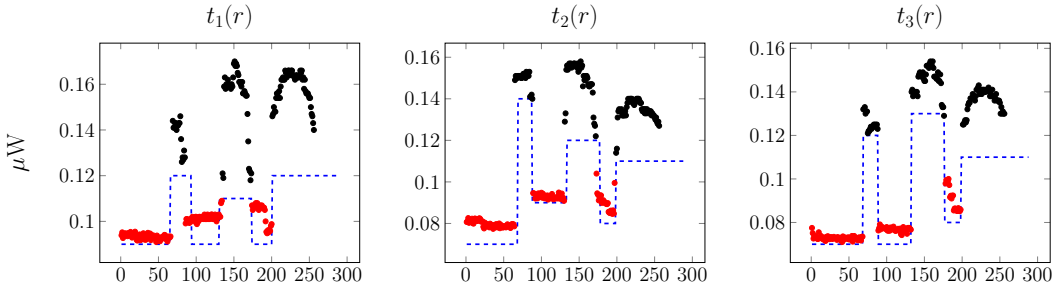
(a) Trivium-MB [102]



(b) TriviA [54]

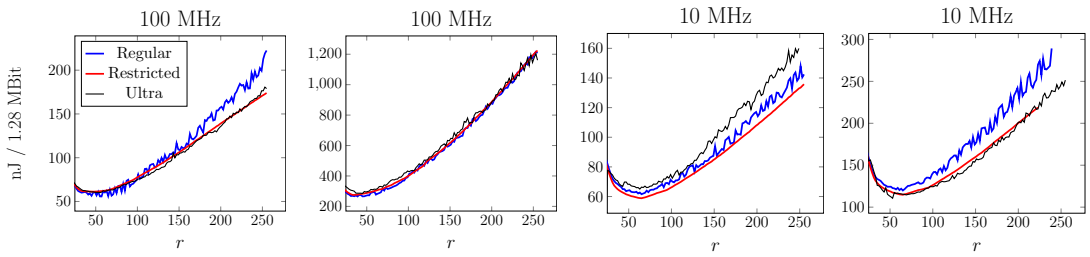


(c) Kreyvium [52]



(d) Triad-SC [24]

Figure 4.13: Power consumption measurements for all the unrolled strand trees for the 128-bit key size variation of Trivium proposed by Maximov and Biryukov [102], the state update function of the TriviA stream cipher [54], Kreyvium [52] and Triad-SC [24]. The Measurements were obtained using the TSMC 90 nm process at a frequency of 10 MHz.



(a) NanGate 15 nm (b) NanGate 45 nm (c) UMC 65 nm (d) TSMC 90 nm

Figure 4.14: Grain-128 energy measurements for three synthesis settings and cell libraries. The complete set of plots for different frequencies is given in Appendix B.3.

that increasing the number of perfect unrolled strand trees also correlates with the power consumption although in a weaker form, hence our results are also applicable to stream ciphers whose state update functions are significantly more complicated than in Trivium-like ciphers.

4.4.2 Applicability to Subterranean-Deck

Unlike Trivium and Grain-128, the Subterranean-Deck [59] stream cipher does not feature a rotating register but in each round each state bit $x_1, x_2, \ldots, x_{257}$ is replaced by the output of a single strand that is replicated 257 times such that

$$(x_1, \ldots, x_{257}) \leftarrow (T_{\pi(1)}, T_{\pi(2)}, \ldots, T_{\pi(257)})$$
$$T_i \leftarrow (x_i + (x_{i+1} + 1)x_{i+2}) + (x_{i+3} + (x_{i+4} + 1)x_{i+5}) + (x_{i+8} + (x_{i+9} + 1)x_{i+10}),$$

for some permutation π. This strand can be realized in 3 NOT, 3 NAND, 1 XNOR3 and 1 XOR4 gates for feature-rich cell libraries and with 3 NOT, 3 NAND and 4 XOR2 for the more basic libraries. Denote $t_i = (x_i + (x_{i+1} + 1)x_{i+2})$. Each T_i consists of 3 sub-strands t_i, t_{i+3}, t_{h+8} of exactly equal circuit complexity. In restricted mode, if each T_i is compiled separately, then the sub-strand t_i is replicated 3 times in the strands T_i, T_{i-3}, T_{i-8}. This increases the circuit size three times, and also adds to unnecessary power consumption. Instead, if we choose t_i (in place of T_i) as the minimal unit whose compilation is restricted, then this replication can be avoided, and this is precisely what we do here. This simplicity lends itself well to the restricted mode of synthesis as shown in Table 4.2 and this mode is decidedly better energy-wise.

A unique property of this structure is that all the sub-strands t_i that constitute the round-function at any level are perfect trees in the corresponding circuit graph. As a result, it is expected that all the sub-strands at a given level consume similar energy. This is borne out by simulations performed on 4-round unrolled Subterranean-Deck as shown in Figure 4.15.

Table 4.2: Subterranean-Deck energy measurements for $r = 4$ for four cell libraries.

	nJ/1.28 MBit		
	Regular	Ultra	Restricted
NanGate 15 nm (100 MHz)	162.9	153.0	**150.5**
NanGate 45 nm (100 MHz)	341.2	389.1	**326.6**
UMC 65 nm (10 MHz)	129.9	156.1	**107.7**
TSMC 90 nm (10 MHz)	222.3	206.5	**192.1**

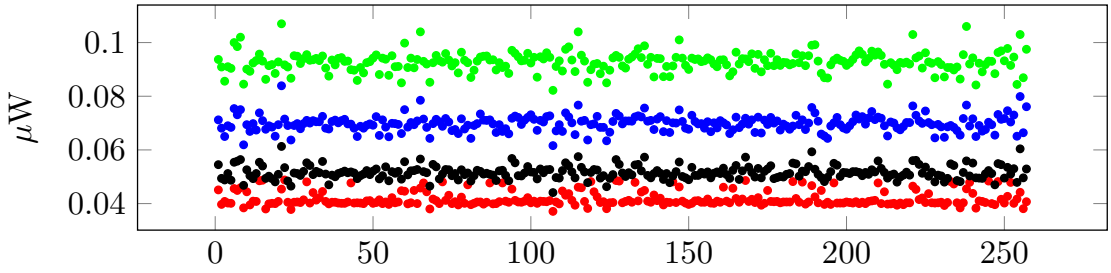


Figure 4.15: Power plot for 4-round unrolled Subterranean-Deck (TSMC 90 nm at 10 MHz). The points in red, black, blue and green represent the power consumed by the strands t_i in the first, second, third, and fourths levels of the round function.

4.5 Summary

In this chapter, we make some fundamental observations about the energy consumption of hardware-targeted stream ciphers and propose the first heuristic energy model that is based on the novel perfect tree metric. Our model is both simple and widely applicable to a wide range of stream ciphers and thus enables designers of future algorithms to specifically optimize for the energy consumption. The perfect tree energy model finds direct application in Trivium-LE(F) and Trivium-LE(S) that stand as the most energy-efficient encryption primitives known in the literature with a 10-15% (resp. around 25 %) margin to the next best cipher. A complete summary of all measurements is given in Table 4.3. Finally, we extend the reach of our model beyond stream ciphers and propose a novel, energy-efficient MAC Trivium-LE-MAC that can then be used to bootstrap an energy-efficient AEAD mode.

[...] To be praised for my plastic lack of inspiration [...]

Table 4.3: Measurements summary for all investigated stream ciphers.

Scheme	Library	Area (GE)	Power (μW)					Energy (nJ/1.28 Mbit)				
			0.2 MHz	1 MHz	10 MHz	100 MHz	1 GHz	0.2 MHz	1 MHz	10 MHz	100 MHz	1 GHz
Trivium	NanGate 15 nm	10834	–	46.23	127.1	936.1	9026	–	205.7	56.39	41.64	40.15
($r = 288$)	NanGate 45 nm	9521	–	577.5	932.1	4477	39980	–	2569	414.6	199.2	177.2
	UMC 65 nm	9911	5.579	13.15	98.36	905.1	–	124.1	58.51	43.75	42.83	–
	TSMC 90 nm	9757	7.015	19.85	164.3	1609	–	155.8	88.29	73.08	71.57	–
Trivium-LE(F)	NanGate 15 nm	10834	–	45.35	118.3	848.4	8147	–	201.7	52.62	37.73	**36.24**
($r = 288$)	NanGate 45 nm	9521	–	574.5	901.0	4166	36863	–	2555	400.8	185.3	**163.9**
	UMC 65 nm	9911	5.391	12.21	88.97	856.5	–	119.3	54.34	39.57	**38.10**	–
	TSMC 90 nm	9757	6.659	18.12	147.0	1436	–	148.1	80.60	65.39	**63.89**	–
Trivium-LE(S)	NanGate 15 nm	10834	–	44.42	108.2	746.3	7127	–	197.6	48.13	32.68	**31.70**
($r = 288$)	NanGate 45 nm	9521	–	568.9	845.1	3608	31247	–	2531	376.1	160.5	**138.9**
	UMC 65 nm	9911	5.238	11.30	80.51	767.2	–	115.9	50.23	35.81	**34.13**	–
	TSMC 90 nm	9757	6.411	16.08	132.1	1311	–	142.58	71.53	58.76	**58.33**	–
Triad-SC	NanGate 15 nm	10834	–	44.88	121.6	889.1	8564	–	224.6	60.84	44.49	42.85
($r = 256$)	NanGate 45 nm	9199	–	561.8	918.8	4488	40075	–	2811	459.7	224.6	201.5
	UMC 65 nm	9487	5.035	11.79	87.79	847.5	–	126.0	59.01	43.39	42.42	–
	TSMC 90 nm	9350	6.422	17.90	147.1	1438	–	160.7	89.61	73.62	72.02	–
Trivium-MB	NanGate 15 nm	13635	–	57.93	185.5	1460	14218	–	257.7	82.51	64.99	63.25
($r = 288$)	NanGate 45 nm	12132	–	768.9	1470	8487	75790	–	3402	654.1	377.6	335.9
	UMC 65 nm	12287	6.886	15.95	118.0	1138	–	153.2	70.99	52.49	50.66	–
	TSMC 90 nm	12783	9.798	28.02	233.0	2283	–	217.9	124.6	103.6	101.6	–
TriviA	NanGate 15 nm	15340	–	71.15	231.0	1829	17813	–	237.4	77.07	61.02	59.43
($r = 384$)	NanGate 45 nm	13440	–	839.1	1565	8835	59172	–	2799	522.5	294.7	196.1
	UMC 65 nm	13892	8.869	23.24	184.9	1809	–	147.9	77.55	61.69	60.12	–
	TSMC 90 nm	13867	11.94	37.59	326.1	3210	–	199.3	125.4	108.8	107.2	–
Kreyvium	NanGate 15 nm	11043	–	50.69	143.5	1078	10425	–	250.5	71.81	53.95	52.16
($r = 288$)	NanGate 45 nm	9703	–	594.8	1015	5220	47308	–	2976	508.0	261.2	236.7
	UMC 65 nm	10083	5.333	12.10	93.22	900.4	–	133.4	62.59	46.65	45.06	–
	TSMC 90 nm	9927	7.128	20.06	165.5	1620	–	178.4	100.4	82.83	81.07	–
Subterranean-Deck	NanGate 15 nm	12344	–	58.51	199.6	1505	15722	–	585.2	199.6	150.6	157.2
($r = 4$)	NanGate 45 nm	11170	–	706.6	940.1	3327	–	–	7067	940.2	332.2	–
	UMC 65 nm	11620	5.756	14.05	107.6	1003	–	575.7	140.1	107.7	103.1	–
	TSMC 90 nm	11125	8.741	24.05	192.2	1930	–	874.2	240.6	192.3	193.0	–
Grain-128	NanGate 15 nm	8565	–	31.89	51.12	457.1	4401	–	645.1	103.5	92.65	89.71
($r = 64$)	NanGate 45 nm	7592	–	456.3	748.6	3671	–	–	9238	1516	743.1	–
	UMC 65 nm	7544	3.636	8.356	61.45	592.4	–	294.3	169.7	124.4	119.9	–
	TSMC 90 nm	7512	5.411	15.30	126.6	1239	–	438.4	309.6	256.4	250.3	–

5 Atom

[...] Mummified circuitry [...]

We remain in the domain of stream ciphers, rather than exploring the feasibility of potential energy consumption optimisations through the lens of existing algorithms, we attempt to conceive a novel energy-efficient by specifically reducing the internal state size and thus the corresponding register footprint. In hardware-oriented stream ciphers that produce a single keystream bit per clock cycle, the register that hosts the cipher state occupies the largest chunk of circuit area and consumes the most energy of all components hence a small state size is beneficial to both metrics.

Crucially, however, the state size cannot be arbitrarily small due to the existence of Time-Memory trade-off attacks popularized Hellman [79] and subsequently applied to stream ciphers as Time-Memory-Data (TMD) trade-off attacks by Biryukov and Shamir [37] where it was proven that for a stream cipher to be secure against generic TMD trade-off attacks, the size of its internal state needed to be at least twice the size of its secret key in bits. The key strategy lies in mounting a table based attack that stored functional iterates of a function that mapped the internal state of the cipher to a keystream prefix of equal length.

In 2015, the stream cipher Sprout [6] was proposed that had a Grain-like architecture and surprisingly an internal state and key both of 80 bits. Yet the cipher seemed to be immune to any of the above TMD attacks. The reason for that is that although the internal state of Sprout it of 80 bits, the state update algorithm required a key input. Consequently, it is impossible to construct any function that maps the internal state to keystream without the secret key. Nevertheless, Sprout fell under heavy cryptanalytic scrutiny [10, 69, 96, 128] with the attack in [69] being the decisive from a design point of view. The function of the key that was used to update the state in Sprout was non-linear and so the authors were able to construct a table using some special states for which the key input was 0 for 40 consecutive cycles: hence a function mapping state to a 40-bit keystream prefix was possible for these special states. Using this the authors constructed a probabilistic attack that sampled random keystream prefix until they found one that was present in the tables. After this key recovery was possible in practical time. The attack underscored the fact that any key function that was used for state update had to be linear. This was exactly the approach used in later Grain-like designs like Lizard [75] and Plantlet [105]. Over the years there have been attacks reported against Lizard and Plantlet as well. In [23], the authors reported various distinguishing attacks against the

full round version cipher and a key recovery attack against 223 of the 256 initialisation rounds of the cipher. Later, Banik et al. [12] proposed a differential attack against Plantlet that finds the key in around 2^{70} encryptions. Ultimately, Todo et al. [123] proposed a correlation attack was proposed on Plantlet, albeit on a version that allowed more keystream bits to be extracted from a single key-IV pair than mentioned in the specifications.

Contributions. Almost all stream ciphers with the Grain structure, including Grain-v1 and Grain 128, have had some weaknesses or undesirable properties reported against them. Thus from a design point of view it is an important question whether it is possible to design stream ciphers securely in the long run. In this chapter, we take up this very challenge. Firstly, our goal is to devise a circuit component that is able to prevent some of the attacks that have been proposed in literature. Secondly, we aim to minimize the hardware footprint of the cipher and thus its energy consumption, and keep it ideally less than other stream ciphers proposed in literature that offer a similar security level. With respect to this, propose the stream cipher Atom that offers 128-bit security and has an internal state of around 159 bits which is less than 25% more than the secret key size. On the face of it, Atom has the same Grain-like structure adopted by Plantlet, Sprout and Lizard. However, we do add a circuit-level novelty to the cipher specification that costs only around 150 Gate-Equivalents (GE) in hardware but guarantees immunity against all attacks proposed against small state ciphers in the recent past. In fact, we prove its security in the context of specific attacks that have been proposed against similar designs in literature and present implementation results that establish its competitiveness among stream ciphers with Atom being both the most area and energy-efficient[21] construction among other contemporary design that also feature key size of 128 bits. Atom was included in the fourth issue of Transactions on Symmetric Cryptology (IACR-ToSC) in 2021 [20].

Outline. In Section 5.1, we present the algebraic specifications of the cipher. In Section 5.2, we argue about some design choices and why they make sense for a short state cipher. Section 5.3, presents a comprehensive security analysis. Section 5.4 details the implementation results. Section 5.5 concludes this chapter.

5.1 Specification

Atom is a stream cipher inspired by the Grain family composed of a linear feedback shift register (LFSR) and a non-linear feedback shift register (NFSR). An high-level overview of Atom is depicted in Figure 1. The size of the secret key K of Atom is 128 bits. Similar to most stream ciphers, the algorithm is divided into an initialisation phase and a keystream generation phase. The LFSR and NFSR are connected through a XOR gate and have a length of 90 and 69 bits respectively. At clock cycle $t = 0, 1, \ldots$, denote the contents in NFSR and LFSR by $B^t = (b_0^t, \ldots, b_{89}^t)$ and $L^t = (l_0^t, \ldots, l_{68}^t)$, respectively. The definitions of the output function,

[21] Energy efficiency is taken in the context of single-bit circuits that produce a single keystream bit per clock cycle. In Chapter 4, we exclusively worked with fully unrolled ciphers in the slipstream of the work by Banik et al. [25] that demonstrated the competitiveness of unrolled stream ciphers in the encryption of larger amounts of data.

LFSR and NFSR are detailed below:

Output Function. The output function $O(B^t, L^t)$ of Atom is a sum of linear terms, a quadratic bent function and another 9-variable function h such that

$$\begin{aligned} O(B^t, L^t) = & b_1^t \oplus b_5^t \oplus b_{11}^t \oplus b_{22}^t \oplus b_{36}^t \oplus b_{53}^t \oplus b_{72}^t \oplus b_{80}^t \oplus b_{84}^t \\ & \oplus l_5^t l_{16}^t \oplus l_{13}^t l_{15}^t \oplus l_{30}^t l_{42}^t \oplus l_{22}^t l_{67}^t \oplus h(l_7^t, l_{33}^t, l_{38}^t, l_{50}^t, l_{59}^t, l_{62}^t, b_{85}^t, b_{41}^t, b_9^t), \end{aligned}$$

where $h(X) = h(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ is defined as

$$\begin{aligned} h(X) = & x_0x_1x_2x_7x_8 \oplus x_0x_1x_2x_7 \oplus x_0x_1x_2x_8 \oplus x_0x_1x_2 \oplus x_0x_1x_3x_7x_8 \\ & \oplus x_0x_1x_3x_7 \oplus x_0x_1x_4x_7x_8 \oplus x_0x_1x_4x_7 \oplus x_0x_1x_4x_8 \oplus x_0x_1x_4 \\ & \oplus x_0x_1x_5x_7x_8 \oplus x_0x_1x_5x_7 \oplus x_0x_1x_6x_7x_8 \oplus x_0x_1x_6x_8 \\ & \oplus x_0x_1x_7x_8 \oplus x_0x_1x_8 \oplus x_0x_2x_3x_7x_8 \oplus x_0x_2x_3x_7 \oplus x_0x_2x_3x_8 \\ & \oplus x_0x_2x_3 \oplus x_0x_2x_4x_7x_8 \oplus x_0x_2x_4x_8 \oplus x_0x_2x_5x_7x_8 \oplus x_0x_2x_5x_7 \\ & \oplus x_0x_2x_5x_8 \oplus x_0x_2x_5 \oplus x_0x_2x_6x_7x_8 \oplus x_0x_2x_6x_8 \oplus x_0x_2x_7x_8 \\ & \oplus x_0x_2x_8 \oplus x_0x_3x_4x_7x_8 \oplus x_0x_3x_4x_7 \oplus x_0x_3x_5x_7x_8 \oplus x_0x_3x_5x_7 \\ & \oplus x_0x_3x_6x_7x_8 \oplus x_0x_3x_6x_7 \oplus x_0x_3x_8 \oplus x_0x_3 \oplus x_0x_4x_5x_7x_8 \\ & \oplus x_0x_4x_5x_7 \oplus x_0x_4x_6x_7x_8 \oplus x_0x_4x_6x_8 \oplus x_0x_4x_7 \oplus x_0x_4 \\ & \oplus x_0x_5x_6x_7x_8 \oplus x_0x_5x_6x_7 \oplus x_0x_5x_7x_8 \oplus x_0x_5x_7 \oplus x_0x_6x_7 \\ & \oplus x_0x_6x_8 \oplus x_0x_7x_8 \oplus x_1x_2x_3x_7x_8 \oplus x_1x_2x_4x_7x_8 \oplus x_1x_2x_4x_8 \\ & \oplus x_1x_2x_5x_7x_8 \oplus x_1x_2x_6x_7x_8 \oplus x_1x_2x_6x_8 \oplus x_1x_2x_7 \oplus x_1x_2x_8 \\ & \oplus x_1x_2 \oplus x_1x_3x_4x_7x_8 \oplus x_1x_3x_5x_7x_8 \oplus x_1x_3x_6x_7x_8 \oplus x_1x_3x_7 \\ & \oplus x_1x_4x_5x_7x_8 \oplus x_1x_4x_5x_8 \oplus x_1x_4x_6x_7x_8 \oplus x_1x_4x_7 \oplus x_1x_4 \\ & \oplus x_1x_5x_6x_7x_8 \oplus x_1x_5x_6x_7 \oplus x_1x_5x_7x_8 \oplus x_1x_5x_7 \oplus x_1x_5x_8 \\ & \oplus x_1x_6x_7 \oplus x_1x_8 \oplus x_1 \oplus x_2x_3x_4x_7x_8 \oplus x_2x_3x_5x_7x_8 \oplus x_2x_3x_6x_7x_8 \\ & \oplus x_2x_4x_5x_7x_8 \oplus x_2x_4x_5x_8 \oplus x_2x_4x_6x_7x_8 \oplus x_2x_4x_7x_8 \oplus x_2x_4x_8 \\ & \oplus x_2x_5x_6x_7x_8 \oplus x_2x_5x_6x_8 \oplus x_2x_5x_8 \oplus x_2x_6x_7x_8 \oplus x_2x_6x_8 \\ & \oplus x_2x_7x_8 \oplus x_2 \oplus x_3x_4x_5x_7x_8 \oplus x_3x_4x_5x_7 \oplus x_3x_4x_6x_7x_8 \\ & \oplus x_3x_4x_6x_7 \oplus x_3x_5x_6x_7x_8 \oplus x_3x_5x_7x_8 \oplus x_3x_6x_7x_8 \oplus x_3x_6x_7 \\ & \oplus x_3x_7 \oplus x_3 \oplus x_4x_5x_6x_7x_8 \oplus x_4x_5x_6x_8 \oplus x_4x_6x_7x_8 \oplus x_4x_6x_8 \\ & \oplus x_4x_7 \oplus x_5x_7x_8 \oplus x_5 \oplus x_6 \oplus x_7x_8 \oplus x_7 \oplus x_8 \oplus 1 \end{aligned}$$

Note that h is a $(9, 5, 3, 240)$ function, i.e., it is of 9 variables, algebraic degree 5, its correlation immunity is 3 and its non-linearity is 240. It has one of the highest non-linearities among all 9 variable Boolean functions (the highest known non-linearity among 9 variable Boolean functions is 242). Thus the output is a sum of linear terms, a quadratic bent function and h. Since bent functions are known to have highest non-linearity for even variable functions, this provides us adequate protection from correlation attacks.

NFSR. The definition of the update function $G(B^t)$ used in NFSR is specified as follows:

$$\begin{aligned} G(B^t) =& b_0^t \oplus b_{24}^t \oplus b_{49}^t \oplus b_{79}^t \oplus b_{84}^t \oplus b_3^t b_{59}^t \oplus b_{10}^t b_{12}^t \oplus b_{15}^t b_{16}^t \\ & \oplus b_{25}^t b_{53}^t \oplus b_{35}^t b_{42}^t \oplus b_{55}^t b_{58}^t \oplus b_{60}^t b_{74}^t \oplus b_{20}^t b_{22}^t b_{23}^t \\ & \oplus b_{62}^t b_{68}^t b_{72}^t \oplus b_{77}^t b_{80}^t b_{81}^t b_{83}^t. \end{aligned}$$

LFSR. The update function $F(L^t)$ used in LFSR is based on a primitive polynomial over $GF(2)$ and hence always ensures a period of $2^{69}-1$ such that

$$F(L^t) = l_0^t \oplus l_5^t \oplus l_{12}^t \oplus l_{22}^t \oplus l_{28}^t \oplus l_{37}^t \oplus l_{45}^t \oplus l_{58}^t.$$

Initialisation Phase. Denote the 128-bit secret key by $K = (k_0, \ldots, k_{127})$ and the 128-bit initial value by $\text{IV} = (\text{iv}_0, \ldots, \text{iv}_{127})$. In addition, there will be extra 22-bit padding denoted by $\text{PD} = (\text{pd}_0, \ldots, \text{pd}_{21}) = 1^{22}$ (all one string). The NFSR and LFSR registers are initialised as follows:

$$b_i^0 = \text{iv}_i \text{ for } 0 \le i \le 89, \quad l_i^0 = \begin{cases} \text{iv}_{i+90} & \text{if } 0 \le i \le 37; \\ \text{pd}_{i-38} & \text{if } 38 \le i \le 59; \\ 0 & \text{otherwise.} \end{cases}$$

After the two registers are initialized, the state at clock t ($0 \le t \le 510$) is updated through the routine specified below:

$$\begin{aligned} z^t &= O(B^t, L^t), \\ \text{cnt} &= l_{62}^t || l_{63}^t || l_{64}^t || l_{65}^t || l_{66}^t || l_{67}^t || l_{68}^t, \\ b_{89}^{t+1} &= G(B^t) \oplus l_0^t \oplus k_{\text{cnt}} \oplus z^t, \\ b_i^{t+1} &= b_{i+1}^t \ \ (0 \le i \le 88), \\ l_{59}^{t+1} &= F(L^t) \oplus z^t, \\ l_i^{t+1} &= l_{i+1}^t \ \ (0 \le i \le 58), \\ l_{i+60}^{t+1} &= ((t+1) >> (8-i)) \wedge 1 \ \ (0 \le i \le 8). \end{aligned}$$

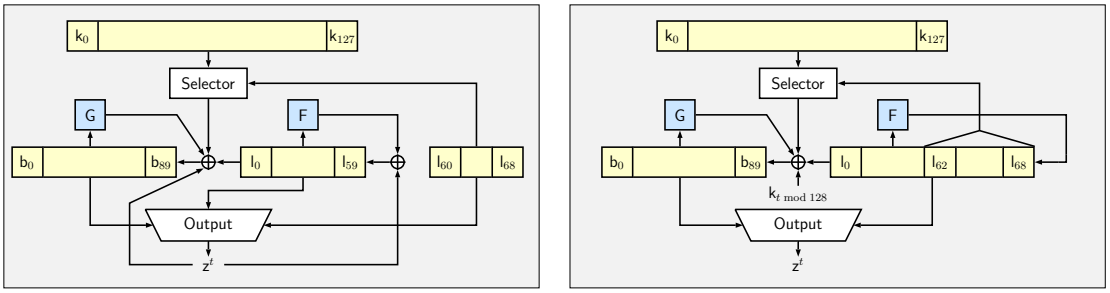
The corresponding illustration can be found in Figure. Thus in the initialisation phase the LFSR is partitioned as two 60 and 9-bit registers. While the 60-bit part is updated using the LFSR logic, the 9-bit part operates as a decimal up-counter. Since the last 7 bits of the LFSR when interpreted as a decimal number (denoted by cnt) also forms the index of the key bit used in the NFSR update, this ensures that all the key bits influence the initialisation function. Note that $l_{i+60}^{t+1} = ((t+1) >> (8-i))$ for $i \in [0,8]$ simply means that in the $(t+1)^{th}$ cycle the LFSR bits 60 to 68 are updated with the 8 bits of the decimal up-counter $t+1$. The 60th LFSR location gets the most significant bits of the counter and 68th location gets the least significant bits.

Keystream Generation. After 511 rounds of the initialisation phase, the state in NFSR becomes $B^{511} = (b_0^{511}, \ldots, b_{89}^{511})$ and the state in LFSR is $L^{511} = (l_0^{511}, \ldots, l_{68}^{511})$. Note that since the last 9

LFSR bits worked as a decimal up-counter during the initialisation phase, and since the initialisation phase had 511 rounds, the last 9 bits of the LFSR at the beginning of the keystream generation phase will always be the all 1 vector. The first keystream used for plaintext encryption is z^{511}. At the keystream phase, the state at clock t ($t \geq 511$) is updated as follows and the keystream bit z^t will be output. Note that we limit the amount of keystream extractable from a single key-IV pair to 2^{64} bits, which should be sufficient for most applications. The exact keystream generation routine is as follows:

$$\begin{aligned} z^t &= O(B^t, L^t), \\ \text{cnt} &= l_{62}^t || l_{63}^t || l_{64}^t || l_{65}^t || l_{66}^t || l_{67}^t || l_{68}^t, \\ b_{89}^{t+1} &= G(B^t) \oplus l_0^t \oplus k_{\text{cnt}} \oplus k_{t\%128}, \\ b_i^{t+1} &= b_{i+1}^t \ \ (0 \leq i \leq 88), \\ l_{68}^{t+1} &= F(L^t) \\ l_i^{t+1} &= l_{i+1}^t \ \ (0 \leq i \leq 67). \end{aligned}$$

The corresponding illustration can be found in Figure. The circuit level novelty that we were referring to the previous section is actually the additional key filter k_{cnt} that we use. As we have seen earlier, the cnt sequence is derived from interpreting the last 7 bits of the LFSR as a decimal number. During the initialisation phase it is simply the i mod 128 sequence with i ranging from 0 to 510. However during the keystream generation phase the cnt sequence depends on the evolution of the LFSR. Since the LFSR has a period of $2^{69}-1$, so does the cnt sequence. This effectively garbles the order of key bits that is used to update the NFSR. Along with the $k_{t \bmod 128}$ filter, this not only guarantees a high period of the keystream, but as we will show in the following sections, it also provides immunity against many known cryptanalytic techniques used to attack small state stream.



(a) Initialisation Phase

(b) Keystream Phase

Figure 5.1: High-level schematic of the Atom initialisation and keystream phase.

5.2 Design Rationale

Every design decision taken on a macro level has been predicated by the need to prevent previous known attacks on short state stream ciphers. In this section we try to reason why

we took some of the steps to adopt the current algebraic structure of the cipher. Our aim is to clearly establish what the design challenges were and how we attempted to address them. Note that most of the attacks against Lizard leveraged the fact that the Key/IV mixing function was not injective. This is not the case for Atom as such we seek to prevent other attacks reported in literature.

5.2.1 Preventing Banik's Key-Recovery Attack on Sprout

In [10], it was first pointed out that there exist around 2^{30} IVs for every key in Sprout such that the LFSR becomes all 0 after the Key-IV mixing phase. In the keystream phase the LFSR does not receive feedback from the output bit and hence remains in the zero state throughout the evolution of the cipher, which made the algebraic structure of the cipher weak. The work in [10] leveraged this fact to report the following:

- Key-IV pairs were found in practical time that produced keystream bits with period equal to 80.
- A key recovery attack was reported in the multiple IV mode. The attacker queries keystream for a fixed secret key and multiple secret IVs and waits till an IV is queried such that the LFSR falls into the all 0 state after the key-IV mixing. Once this happens key recovery was shown to be very efficient, since the algebraic structure of the cipher weakened, simple equations on the key bits could be obtained which could be solved to find the secret key.

The zero state problem was tackled quite elegantly in the design of Plantlet. In the key-IV mixing phase Plantlet fixes the final LFSR bit to 1, so that when the cipher finally enters the keystream generation mode the LFSR state is never all zero on the account of the fact that the final bit is 1. Since the Plantlet LFSR connection polynomial is primitive and has length 61, it has a period of $2^{61}-1$ and it never falls into an all zero state. This effectively prevents all the above attacks. In Atom, the solution we adopt is indeed inspired by Plantlet. During the key-IV mixing phase the last 9 bits of the LFSR effectively work as a decimal counter. Since the number of rounds in the initialisation phase is 511, the last 9 bits of the LFSR are all 1 when we enter the keystream generation phase. The Atom LFSR connection polynomial is also primitive guaranteeing a period of $2^{69}-1$ and the fact that it never falls into the all 0 trap.

In the keystream generation phase the keybit-sum to update the NFSR is $k_{\text{cnt}} \oplus k_{t \bmod 128}$. cnt is basically the decimal number formed by interpreting the last 7 LFSR bits as a decimal number. Since the LFSR has a period of $2^{69}-1$, it is to be expected that the cnt sequence would also have the same period. Therefore it is clear that both the cnt and t mod 128 sequence taken together would repeat only after $\text{lcm}(2^{69}-1, 128) = 2^{76}-128$ clock cycles. This also guarantees that the Atom keystream, produced by most key-IV pairs, has a minimum period of $2^{76}-128$ bits (unless we have some degenerate cases like the all one/zero key for which the minimum period is given by the period of the LFSR i.e., $2^{69}-1$).

5.2.2 Preventing Banik-Barooti-Isobe Attacks on Plantlet

This attack on Plantlet demonstrated in [12] can be summarised as follows:

- The attackers observe that if two internal states of Plantlet differ by $0^{40}||0^{42}||1||0^{18}$, i.e., the 43rd LFSR bit, then they produce keystream vectors whose difference is 0 with probability 1 in 41 clock cycles. The difference is 1 also with probability 1 in 4 other clock cycles.

- They try to obtain two internal states with single bit difference in the 43rd LFSR location by querying some fixed key and random IVs. The keystream extractable with a single key-IV pair is limited to 2^{30} bits. It was calculated that the probability that such difference in states will occur during the course of a single key-IV query is around $2^{-54.6}$. Hence to get the required difference at least once on average the attackers needed to query around $2^{54.6}$ different IVs.

- They inspect the keystream blocks extracted with the $2^{54.6}$ IVs. They can with some probability guess that when 2 segments of keystream blocks possess the 45 bit difference just mentioned, they have been produced by two internal states that differ only in the 43rd LFSR location.

- Thereafter by solving a system of polynomial equations given by the keystream bits, the attacker can find the secret key if his guess was indeed correct, or reach some kind of contradiction if his guess was incorrect. In the latter event, they repeat the procedure for other keystream blocks with the given difference. The process when repeated a finite number of times, does indeed yield the value of the secret key.

For Atom, we experimented with differences of up to hamming weight 4. The best differential characteristic was obtained when two Atom states differ at the 55th LFSR location: they are guaranteed to produce keystream that are either equal or unequal in only 18 clock cycles with probability 1. This number is quite small, compared to Plantlet where 45 keystream bits are guaranteed to be equal/unequal for a well chosen difference in the LFSR state. The reason for this is that, in Atom because of the fact that the last 7 bits of the LFSR provide the index of the key bit which is used to update the NFSR. Thus any LFSR difference will after a short amount of time be fed back into the 68th LFSR location through the LFSR taps. When this happens different key bits are used to update the NFSR and the so a difference is propagated into the NFSR relatively quickly. In any case, when we generate N keystream bits, the probability that there exists two clock instances t_1, t_2 at which the internal states differs only in the 55th LFSR bit is given by birthday bound considerations and is around $p = N^2 \cdot 2^{-160}$. For $N = 2^{64}$, this probability is around 2^{-32}. The attacker would therefore need to query around $V = p^{-1} = N^{-2} \cdot 2^{160}$ IVs to get one difference state on average. The attacker then filters out all internal state pairs for which the keystream segment does not produce the 18 bit difference pattern. The number of such pairs is around $U = V \cdot \frac{N(N-1)}{2} \cdot 2^{-18} \approx 2^{141}$. The attacker would have to then solve polynomial systems arising from these 2^{141} state pairs. This is well above

the complexity of exhaustive search. Note that the main reason that the attack is prevented is that the use of k_{cnt} term used in the state update depending on the last 7 LFSR bits. This prevents us from getting a large enough differential pattern of equal/unequal bits that can bring down the value of U to less than 2^{128}.

5.2.3 Preventing Todo-Meier-Aoki Attacks on Plantlet

The attacks reported in [123] are against a variation of Plantlet that allows up to 2^{54} keystream bits per key-IV pair and are correlation attacks of a similar type reported against the Grain family. The attack can be summarised in a few steps:

- The attackers try to formulate probabilistic equations of the following form:

$$\bigoplus_{t\in\mathbb{T}_Z} z_t \oplus \bigoplus_{t\in\mathbb{T}_K} k_t \oplus \bigoplus_{t\in\mathbb{T}_L} l_i^t = \epsilon \tag{5.1}$$

  where $\mathbb{T}_Z$, $\mathbb{T}_K$, $\mathbb{T}_L$ are linear masks that denote subsets of keystream, key and LFSR state bits that add up to form the probabilistic equations. ϵ denotes a variable such that $Pr[\epsilon = 0] = 1/2 \cdot (1+\eta)$, where η is the correlation term whose value determines the complexity of the attack algorithm.

- The value of η depends on the linear biases of the NFSR update function and the output function.

- In Plantlet, the keybit used in the state update function repeats every 80 clock cycles. To apply the correlation attack to Plantlet, the attackers need $\bigoplus_{t\in\mathbb{T}_K} k_t$ to be a constant. Therefore the attackers build up a bank of equations at intervals of 80 cycles each. For example if the original probabilistic equation was $z_t \oplus z_{t+2} \oplus k_t \oplus k_{t+45} \oplus k_{t+51} \oplus l_{10}^t + l_{14}^t \oplus l_{31}^t = \epsilon$, the attackers build equation banks of the form:

$$\begin{aligned} z_0 \oplus z_2 \oplus k_0 \oplus k_{45} \oplus k_{51} \oplus l_{10}^0 + l_{14}^0 \oplus l_{31}^0 &= \epsilon_1 \\ z_{80} \oplus z_{82} \oplus k_0 \oplus k_{45} \oplus k_{51} \oplus l_{10}^{80} + l_{14}^{80} \oplus l_{31}^{80} &= \epsilon_2 \\ z_{160} \oplus z_{162} \oplus k_0 \oplus k_{45} \oplus k_{51} \oplus l_{10}^{160} + l_{14}^{160} \oplus l_{31}^{160} &= \epsilon_3 \\ &\vdots \end{aligned}$$

  Note that the value of $\bigoplus_{t\in\mathbb{T}_K} k_t$ remains constant in these equations because in Plantlet the keybits used in the state update repeat every 80 cycles.

- Once the attacker has around N such equations and the correlation of each ϵ_i is η, then for the correct value of the LFSR internal state the difference between the number of correct and incorrect equations will be distributed according to $\mathcal{N}(N\eta, N\eta^2)$ if $\bigoplus_{t\in\mathbb{T}_K} k_t = 0$ or $\mathcal{N}(-N\eta, N\eta^2)$ if $\bigoplus_{t\in\mathbb{T}_K} k_t = 1$. This distribution when the LFSR internal state is incorrect is given by $\mathcal{N}(0, N\eta^2)$, where $\mathcal{N}(\mu,\sigma)$ denotes the normal distribution with mean μ and variance σ. Note that around $N = O(\eta^2)$ equations are needed to reliably distinguish the distributions $\mathcal{N}(\pm N\eta, N\eta^2)$ and $\mathcal{N}(0, N\eta^2)$ and mount the attack.

- The authors use a maximum likelihood decoding algorithm such as the Fast Walsh Hadamard transform (FHWT) to find the LFSR state efficiently. Thereafter the key and NFSR state can be found by solving polynomial equations on the keystream bits.

The reason why this attack can not be used against Atom is as follows. In Atom, we can also derive probabilistic equations as given in Equation (5.1). However, the key bits used in the state update depend on both the LFSR and the time counter and as such is not known completely to the attacker. Since the LFSR is guaranteed to begin the keystream phase with a non zero state and that its connection polynomial is primitive over GF(2), it will only repeat after $2^{69}-1$ cycles. The mod 128 counter repeats only after 128 cycles. Hence the keybits repeat only after $\tau = \text{lcm}(2^{69}-1, 128) = 2^{76}-128$ clock cycles. This greatly increases the data complexity of any correlation attack on Atom. Since we limit the amount of keystream to 2^{64} per key-IV pair, this type of attack seems infeasible. Nevertheless, we heuristically searched different linear combinations to mount a linear attack. However, we did not find any efficient linear combination of keystream bits that has a high enough correlation with the sum of LFSR bits and keybits.

5.2.4 Preventing Esgin-Kara Attacks on Sprout

One of the reasons Sprout was immune to the TMD trade-off attacks, despite having state size equal to the size of the key, was that the key was used to update the state update transitions, and so it was impossible to construct a function that mapped state to any length of keystream prefix, without using the key. Thus the effective size of the internal state was twice the size of the key as required. The attack against Sprout proposed in [69] can be summarised below:

- The authors showed that in spite of all the above, there existed special states of Sprout for which it is possible to map the state to keystream without requiring the secret key. This is because the round key function used to update the state at time t was of the form $(k[t \bmod 80]) \cdot (\sum n_i^t + \sum l_i^t)$, where $k[i]$ is the i-th key bit. Now if the expression $(\sum n_i^t + \sum l_i^t)$ is 0 for 40 consecutive cycles then the contribution of the key to the state update is 0, which in other words means that it is now possible for these special states to produce keystream without requiring the key and so a function mapping state to a 40-bit keystream segment is now possible, which does not additionally require the secret key.

- The authors showed that it was possible to efficiently enumerate these special states and tabulate these states along with the 40 bit keystream segment produced by them.

- In the online stage of the attack, the authors examined every available keystream segment and look for the segment in the precomputed tables and try to extract the state. It was shown that if done sufficient number of times, Sprout will degenerate into a special state and the attacker can extract the corresponding state from the table. After this the attacker solves some algebraic equations to find the secret key.

- The authors showed that both the online and offline stages of the attack had practical time, memory and data complexities.

Ever since the publishing of [69], all subsequent stream cipher designs like Plantlet and Lizard have been proposed with strictly linear key bit contributions to the state update function, i.e., of the form $k[t \bmod \text{keysize}]$. We too adopt linear key addition to the state update, but the key bit to be used is not indexed by a counter but by the last seven bits of the LFSR. This gives rise to some unique opportunities. To understand better, let us consider an altered variant of Atom in which the state update function does not include the $k_{t\%128}$ term, in other words

$$b_{89}^{t+1} = G(B^t) \oplus l_0^t \oplus k_{\text{cnt}},$$
$$l_{68}^{t+1} = F(L^t).$$

Note that the LFSR runs autonomously during keystream generation, and since initialisation lasts 511 cycles, the cipher always enters keystream phase with last 9 bits all 1. Since the connection polynomial is primitive, it is well known, that the LFSR has a period of $2^{69}-1$ and never enters the all zero state. Now let us say that for some t, the last seven bits of the LFSR is 000 0001 which occurs with probability 2^{-7}. We want that for T consecutive cycles the last seven bits of the LFSR take the following values `01,02,04,08,10,20,40,01`... and so on. This way the only keybits involved in the update of the NFSR state will be from the 7-element set $\mathbb{K} = \{k_1, k_2, k_4, k_8, k_{16}, k_{32}, k_{64}\}$. For this to happen the following three equations need to hold:

1. $l_{62}^t, l_{63}^t, l_{64}^t, l_{65}^t, l_{66}^t, l_{67}^t, l_{68}^t = [000\ 0001]$
2. $l_{68}^{t+i} = F(L^{t+i-1}) = 0$ for $i = 1,2,3,4,5,6 \bmod 7$ and $1 \leq i < T$
3. $l_{68}^{t+i} = F(L^{t+i-1}) = 1$ for $i = 0 \bmod 7$ and $1 \leq i < T$.

There are a total of $T+6$ conditions to fulfill and it is not difficult to see that the solution space of the above linear system of equations has dimension $69-(T+6) = 63-T$. Since the system is linear it is easily possible to enumerate all such states. Note that for all such 2^{63-T} LFSR states, we could easily define a function that maps the entire internal state + $\mathbb{K}$ to a T-bit keystream segment. Our attack unfolds as specified below into phases:

Offline.

1. For all 2^{90} NFSR states, 2^{63-T} LFSR states enumerated above and all 2^7 values of $\mathbb{K}$ (a total of 2^{160-T} iterations)
   (a) Compute the T-bit keystream segment Z.
   (b) Store in a table Tab(Z)= State$||Q$, where Q are the seven key bits that produced Z.
   (c) Each table cell will store around 2^{160-2T} entries (a total of 2^{160-T} entries are divided among 2^T table cells).

The offline phase comes with a time complexity of 2^{160-T} and a memory complexity of $160 \cdot 2^{160-T}$.

Online.

1. Take any keystream segment Z.
2. Extract all 2^{160-2T} entries S from $\mathrm{Tab}(Z)$.
3. For all such S:
   (a) Solve for the keybits not in $\mathbb{K}$.
   (b) If a solution exists then output key else try another value of S.

In terms of the time complexity of the online phase, note that we have stored a fraction $1/2^{T+6}$ of all internals states. Under standard randomness assumptions, the online stage will extract the correct state in $2^{T+6} \cdot 2^7$ attempts when both the state and key are correctly extracted. Hence the complexity of the online phase is $2^{T+13+160-2T} = 2^{173-T}$ attempts to solve for the remainder of keybits. Assuming that finding the rest of the key bits is efficient, taking $T \approx 60$, would put the complexity of both the online and offline phase below the complexity of exhaustive search. Now when we include the $k_{t \bmod 128}$ term in the update function it ensures that we can never get a relation between the state and keystream segment that only involves a few key bits. More precisely any function mapping into $T \leq 128$ keystream bits must involve at least $159 + T$ input bits of the state and key. Thus this attack is prevented.

5.3 Security Evaluation

We start the security evaluation by showing the resistance of Atom against various TMD trade-off attacks before shifting to differential strategies.

5.3.1 TMD Trade-Off Attacks

TMD trade-off attacks aim to invert a one way function f at a single point in the range of function. The attack is probabilistic and the attacker may need access to multiple points in the range of f. For stream ciphers, the one way function is typically the map between the internal state and the prefix of the keystream bits produced by the internal state. We have already seen in the previous section that any function mapping into $T \leq 128$ keystream bits must involve at least $159 + T$ input bits of the state and key. In this context, let us look at some of the common TMD trade-off attacks reported against stream ciphers:

Biryukov-Shamir Attack [37]. Given that N is the cardinality of the set of internal states, the attacker chooses m, t, D so that $mt^2 = N$ and $t \geq D$, where D will be the data complexity required for the attack. The attacker builds $\frac{t}{D}$ tables of size $m \times t$ in the following manner: he randomly chooses m initial states. For each initial state, he forms a chain of length t by iteratively applying the stream cipher function f and using the keystream as the state for the next point. For each table some unique reordering of the bits after applying the function f

is used so that the tables do not store the same set of states. For the attacker to be able to do this he must be able to formulate the stream cipher function in such a manner that it maps equal length input and output bit vectors. The only choice the attacker has is to choose $T = 159 + 128 = 287$, which enables him to formulate the function f as mapping the key and state to the first 287 keystream bits produced by the generator. In the process, $mt \cdot \frac{t}{D} = \frac{N}{D}$ of the state space is covered by all the chains. This also happens to be the offline complexity T_{offline} of this stage. Also only the start and endpoints of each chain are stored in tables, and so $M = m \cdot \frac{t}{D}$ bits of memory is used. In the online phase, the attacker has access to D segments of keystream. For each target keystream segment y, he applies f on y upto t times checks if y is present as an endpoint in any table. If yes, he goes back to the starting point and retrieves the state just before y in the chain. The total time complexity is thus $T_{\text{online}} = D \cdot t \cdot \frac{t}{D} = t^2$. This gives us the trade-off curve $T_{\text{online}} M^2 D^2 = N^2$, with the limitation that $T_{\text{online}} \geq D^2$. For Atom, $N = 2^{287}$ and there is no point in the trade-off curve for which T_{online} and T_{offline} are both less than 2^{128}.

Sampling Resistance of Atom. The main idea of sampling is to find an efficient way to generate and enumerate special cipher states, for which the first few keystream bits generated by the cipher is a fixed string. If this happens for a run of x bits, the sampling resistance of the cipher is defined to be $R = 2^{-x}$. This leads to improved trade-off attacks when the value of x is significant. For Atom, the taps are extremely densely packed. The sampling resistance is quite large and around 2^{-5}. Observe the following set of equations in the generator:

$$
\begin{aligned}
b_{22}^t &= z_t \oplus b_1^t \oplus b_5^t \oplus b_{11}^t \oplus b_{36}^t \oplus b_{53}^t \oplus b_{72}^t \oplus b_{80}^t \oplus b_{84}^t \\
&\quad \oplus l_5^t l_{16}^t \oplus l_{13}^t l_{15}^t \oplus l_{30}^t l_{42}^t \oplus l_{22}^t l_{67}^t \oplus h(l_7^t, l_{33}^t, l_{38}^t, l_{50}^t, l_{59}^t, l_{62}^t, b_{85}^t, b_{41}^t, b_9^t) \\
b_{23}^t &= z_{t+1} \oplus b_2^t \oplus b_6^t \oplus b_{12}^t \oplus b_{37}^t \oplus b_{54}^t \oplus b_{73}^t \oplus b_{81}^t \oplus b_{85}^t \\
&\quad \oplus l_6^t l_{17}^t \oplus l_{14}^t l_{16}^t \oplus l_{31}^t l_{43}^t \oplus l_{23}^t l_{68}^t \oplus h(l_8^t, l_{34}^t, l_{39}^t, l_{51}^t, l_{60}^t, l_{63}^t, b_{86}^t, b_{42}^t, b_{10}^t) \\
&\vdots \\
b_{26}^t &= z_{t+4} \oplus b_5^t \oplus b_9^t \oplus b_{15}^t \oplus b_{40}^t \oplus b_{57}^t \oplus b_{76}^t \oplus b_{84}^t \oplus b_{88}^t \\
&\quad \oplus l_9^t l_{20}^t \oplus l_{17}^t l_{19}^t \oplus l_{34}^t l_{46}^t \oplus l_{26}^t l_{71}^t \oplus h(l_{11}^t, l_{37}^t, l_{42}^t, l_{54}^t, l_{64}^t, l_{66}^t, b_{89}^t, b_{45}^t, b_{13}^t)
\end{aligned}
$$

This means that given the value of 282 particular state and key bits of Atom and the first 5 keystream bits produced from that state, another 5 internal state bits may be deduced efficiently. The equation for b_{27}^t involves b_{90}^t which already contains the b_{25}^t term. This helps us define a function $f : \{0,1\}^{282} \rightarrow \{0,1\}^{282}$. We fix a specific 5 bit string say 1^5. For any 282-bit string x we expand it to a 287-bit string by interpreting it as a partial state of Atom and calculating the remaining 5 bits $b_{22}^t, b_{23}^t, \ldots, b_{26}^t$ by assuming that $z_t, z_{t+1}, \ldots, z_{t+4} = 1^5$. Generate the remaining 282 bits $y = z_{t+5}, z_{t+6}, \ldots, z_{t+286}$ by clocking the Atom generator with the full state. We define $f(x) = y$. Using this technique, in the online stage we wait till we observe the 1^5 vector in the keystream sequence. If so we try to invert f using the subsequent 282 bits of the keystream. It can be shown that the trade-off curve resulting thereof is $T_{\text{online}} M^2 (RD)^2 = (RN)^2$, with the condition $(RD)^2 \leq T$. Again there is no point in the trade-off curve for which T_{online} and T_{offline} are both less than 2^{128}.

Hong-Sarkar [81]. This attack is exactly same as the TMD technique by Biryukov and Shamir [37], except that the definition of the underlying one-way function is different. In this attack, f maps the string consisting of the Key and IV to an equal length keystream bits. Thus if K and V refer to the size of the Key space and IV space respectively, then $N = KV$, and we will have the new trade-off curve $T_{\text{online}} M^2 D^2 = K^2 V^2$ with the limitation that $T_{\text{online}} \geq D^2$. This strategy becomes applicable if $V \ll K$, as in the case of the A5/3 cipher (in which the size of the secret key in 64 bits, and the size of the IV is 22 bits). In our case $K = V$, and $N = KV = 2^{256}$, so there is no point in the trade-off curve for which T_{online} and T_{offline} are both less than 2^{128}.

Dunkelman-Keller [66]. This TMD attack, that is, a multiple IV routine, in which the attacker obtains keystreams from multiple IVs and the same key in order to perform the attack. The definition of the underlying one way function f is slightly different from the Hong-Sarkar attack. Given a fixed IV, the function f maps the secret key to the keystream sequence of equal length. The attacker chooses $\frac{V}{D}$ random IVs. For each IV he constructs t tables as before, by iterative application of the function f from m random starting points, with $mt^2 = K$. Again only the start and end points are stored and so for each IV the storage required is $M_{\text{single}} = mt$, and the total storage is therefore $M_{\text{single}} \cdot \frac{V}{D}$, and the total offline complexity is $T_{\text{offline}} = K \cdot \frac{V}{D}$. In the online phase, the attacker waits until he receives keystream for one of the $\frac{V}{D}$ IVs he had made tables for. This happens in roughly D IV resynchronisations. Once he gets such keystream from such an IV, he retrieves the t tables he had constructed for the particular IV and tries to find the inverse image of the keystream string in each of the tables. Therefore the online complexity is given by $T_{\text{online}} = D + t^2 = D + \frac{K^2}{M^2_{\text{single}}} = D + \frac{K^2 V^2}{M^2 D^2}$ with the constraints $T_{\text{online}} \geq D, V \geq D$. In the case of Atom, $KV = 2^{256}$ and again this attack is not feasible.

5.3.2 Differential Cryptanalysis

Differential cryptanalysis was first introduced to analyse the block ciphers DES and Feal and the hash function N-Hash by Biham and Shamir [34]. Since then, it has been applied to many symmetric cryptographic primitives, not limited to block ciphers. To evaluate the resistance against differential attacks for block ciphers, one way is to obtain the lower bound of the number of active S-boxes, which is a nonlinear operation. For Atom, there are two nonlinear components of NFSR and output function h, which include AND operations as nonlinear operation. Therefore, instead of an active S-box, we search the lower bound of the number of sum of active AND and $h(X)$, which means having at least one active bit as input. In our evaluation, since the maximum differential probability of AND is 2^{-1} and $h(X)$ consists of AND and XOR, we count the maximum differential probability of $h(X)$ as 2^{-1}. Hence, it is sufficient to guarantee the security against the differential attack if there are 128 sum of active AND and $h(X)$. We present this security evaluation with a MILP-based method [108], which is well known as the efficient search method to obtain the lower bound of the number of sum of active AND and $h(X)$ (active S-boxes). Our evaluation uses the Gurobi optimizer as a MILP solver, and searches all bit-wise differential characteristics. Note that our evaluation assumes that each active AND is independent in a differential characteristic. Thus, it might include invalid differential characteristics. However, since our search can cover all valid ones

at the same time, we believe that our evaluation is sufficient for obtaining lower bounds of the number of active AND.

Table 5.1: Lower bounds of the sum of AND and $h(X)$ in the related IV setting.

Number of Rounds	40	45	50	60
Number of Active AND & $h(X)$	18	24	30	43

Table 5.1 shows the minimum number of the sum of active AND and $h(X)$ for 40, 45, 50, 60 clocks at the initialisation phase in the related IV setting. In our evaluation, we can search it for up to 60 clocks with a computer equipped with 48 cores and 256 GB RAM. Form this result, Atom achieves 128 sum of active AND and $h(X)$ for more than 180 clocks. Thus, we expect that the full rounds of Atom can resist differential cryptanalysis.

5.3.3 Conditional Differential Cryptanalysis

Conditional differential cryptanalysis was first introduced in [31]. The technique allows for improved key recovery and distinguishing attacks against a group of ciphers such as Trivium, Katan, Grain-v1 and Grain-128. For example, the authors of [91] demonstrated a 961-round distinguisher for Trivium for a large class of weak keys. Attacks against reduced-round variants of the Grain family were reported in [90].

In its core, conditional differential cryptanalysis is a refinement of ordinary differential attacks where the longevity of differential trails is extended by imposing some conditions on public parameters such as initialisation vectors. Denote by $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ an n-bit initialisation vector and let $\Delta x = (\Delta x_1, \ldots, \Delta x_n) \in \{0,1\}^n$ be an IV difference such that $x + \Delta x = (x_1 + \Delta x_1, \ldots, x_n + \Delta x_n)$. Furthermore, let $t_i(k,x)$ be the newly generated state bit in round i based on some secret key k and the public parameters x. The IV difference Δx propagates to $t_i(k,x)$ whenever

$$\Delta t_i(k,x) + t_i(k, x + \Delta x) = 1.$$

In order to attain a simple high-round differential trail the attacker can impose conditions on the public parameters to prevent the propagation of the differential into the state in certain rounds. More specifically, these conditions are categorized into two types:

- *Type 1:* Conditions that only involve IV variables, i.e., $w_1(x) \in \{0,1\}$.
- *Type 2:* Conditions that involve both IV and key variables, i.e., $w_2(x,k) \in \{0,1\}$. w_2 should be of the form $w(x,k) = f(x) + g(k)$, where the function $f(x)$ only depends on the IV bits and and the function $g(k)$ only depends on the key bits.

For correctly chosen conditions that prevent the propagation of the differential an adversary hopes to find a biased keystream bit in some round. Then by leveraging this distinguisher the attacker partitions the IV space into 2^N subsets (where N is the total number of type 2 conditions) one for each type 2 condition. A bias should then occur for the one subset for which $g(k)$ is correctly guessed.

The derivation of these conditions can be achieved through computer algebra systems. Evidently, the algebraic expressions of a large number of rounds can be explicitly evaluated when the state update function of the cipher is simple. For example, in Trivium it is effortlessly possible to compute the algebraic equations for more than 200 rounds. This does not hold true for Atom, in fact, due to the complex nature of its state update function, it is not possible to compute the equations of than a dozen rounds, which naturally limits the applicability of conditional differential attacks.

We searched for single-bit differentials heuristically. The best single-bit input differential trail in Atom is found when a difference is introduced in bit IV_{67}. We proceed by stopping its propagation into subsequent rounds with the following conditions:

$$\begin{aligned}
t = 5: &\quad \mathrm{IV}_{73} = 0 \\
t = 6: &\quad \mathrm{IV}_{81} = 0 \\
t = 7: &\quad \mathrm{IV}_{11} = 0 \\
t = 8: &\quad \mathrm{IV}_{64} = 0 \\
t = 11: &\quad \mathrm{IV}_{70} = 0 \\
t = 13: &\quad \mathrm{IV}_{39} = 1 \\
t = 14: &\quad \mathrm{IV}_{22} = 0,\ \mathrm{IV}_{56} = 1,\ \mathrm{IV}_{112} = 0, \\
&\qquad f_1(\mathrm{IV} \setminus \{\mathrm{IV}_0\}) + \mathrm{IV}_0 + k_0 = 1, \\
&\qquad f_2(\mathrm{IV} \setminus \{\mathrm{IV}_8\}) + \mathrm{IV}_8 + k_1 + k_2 + k_4 + k_5 + k_{10} = 1.
\end{aligned}$$

Here f_1, f_2 are polynomials on both key and IV bits. At this point, the propagation of the differential is prevented during the first 15 rounds. Two of the conditions are of type 2, however the polynomials f_1 and f_2 are infeasible to enumerate unless more IV variables are set to zero, i.e.,

$$\mathrm{IV}_i = 0,\ i \in [10,38] \cup [46,55] \cup [57,66] \cup [68,89] \cup [100,127]$$

This means that in total 98 IV bits have to bit set to either 0 or 1, which leaves 30 free variables. After this f_1, f_2 become polynomials defined only on the IV bits. We note that IV_0 only occurs linearly in f_1, the same is true for IV_8 in f_2. As we have two type 2 conditions we partition the IV variables into $2^2 = 4$ sets T_U of the form

$$\begin{aligned}
T_U = \{\mathrm{IV} \in \{0,1\}^{128} \mid &\ \mathrm{IV}_i = 0,\ i \in [10,38] \cup [46,55] \cup [57,66] \cup [68,89] \cup [100,127], \\
&\ \mathrm{IV}_{39} = 1,\ \mathrm{IV}_{56} = 1, \\
&\ \mathrm{IV}_0 = f_1 + k_0,\ \mathrm{IV}_8 = f_2 + k_1 + k_4 + k_5 + k_{10}\},
\end{aligned}$$

where $U = [k_0, k_1 + k_2 + k_4 + k_5 + k_{10}]$. These conditions then produce a detectable bias in the keystream bit of round 36 for the IV set T_U where U has been guessed correctly. We give the full polynomials f_1 and f_2 in Appendix.

Note a bias may linger longer in the cipher state. For example a difference in IV bit IV_{18}

produces a difference in b_{12}^6 under the conditions

$$\mathrm{IV}_{17} = 0,\ \mathrm{IV}_{19} = 0.$$

This conditional differential trail exhibits a bias in state bits b_0^{117} and l_0^{117}, i.e., in round 117. However, it is not clear how such a biased state bit can be exploited given the complicated nature of the keystream function. Consequently, we believe that Atom resists conditional differential attacks with a large security margin.

5.3.4 Integral/Cube Attacks

The integral attack was first proposed by Daemen et al. [58] as a dedicated attack against the block cipher Square, and then it was formalized to the integral property by Knudsen and Wagner [93] We define the four states for a set of 2^n n-bit cell:

(A) If $\forall i, j\ i \neq j \Leftrightarrow x_i \neq x_j$.

(C) If $\forall i, j\ i \neq j \Leftrightarrow x_i = x_j$.

(B) $\bigoplus_i^{2^n-1} x_i$.

(U) Everything else.

In our evaluation, we search the integral distinguisher on clock-reduced Atom. To find the integral distinguisher, we explore the propagation of the division property as proposed by Todo [121], which can search the integral distinguisher in more detail than the integral property, with an MILP-based search method proposed by Xiang et al. [127], which can efficiently explore the propagation of the bit-based division property. When we search the integral distinguisher, we give IV having (A), which denotes that all bits in IV are active, as the division property at the input and then we check whether the output of the keystream bit is balanced after r clocks or not. As a result, we found the integral distinguisher after 67 clocks and we could not find the integral distinguisher after 68 clocks. Thus, we expect that Atom can resist against the integral attack. In addition, the division property was introduced to evaluate cube attacks [64], i.e., it examines the set of key bits J involved in the superpoly given a certain cube I [122]. Therefore, this result shows that the full rounds of Atom has a sufficient security level against cube attacks.

5.3.5 Algebraic Attacks

The output functions in Atom were chosen to be sufficiently complicated to prevent advances due to algebraic attacks. In [32], an algebraic attack was proposed against Grain-like ciphers in which the NFSR variables add only linearly to the expression for the keystream bit. For example it was shown that if a modified version of the Grain v1 cipher was conceived in such a way that it contained only the non-linear register from which the keystream was obtained only by linearly adding specific bits of the inner state, then each updated NFSR bit would be a linear function of initial state of NFSR bits. For example if $v_0, v_1, v_2, \ldots, v_{n-1}$ is the initial

state of the NFSR such that each updated bit is calculated as $v_{n+t} = G(v_{n+t-1}, v_{n+t-2}, \ldots, v_t)$ and each keystream bit $z_t = L(v_{n+t-1}, v_{n+t-2}, \ldots, v_t)$, where G, L are non-linear and linear boolean functions respectively on n bits, then updated bit v_{n+t} can be written as

$$v_{n+t} = G(v_{n+t-1}, v_{n+t-2}, \ldots, v_t) = \text{Lin}(v_0, v_1, \ldots, v_{n-1},\ z_0, z_1, \ldots, z_t),\ \forall t \tag{5.2}$$

Here, Lin is another linear function. The above is not difficult to show and requires simple mathematical induction based arguments: if t_0 is the smallest integer for which the expression for z_{t_0} contains the term v_n, then from the linearity of z_{t_0}, we can see that v_n can be written as a linear expression in $v_0, v_1, \cdots, v_{n-1}$ and z_{t_0}. The argument carries forward in a similar manner for any subsequent value of $t = t_0 + 1, t_0 + 2, \ldots$ etc. Now we can multiply Equation (5.2) on both sides by H which is the annihilator of G to get equations of lower algebraic degree. The authors of [32] showed that for the modified of Grain v1, one could generate degree 4 equations using the annihilator of the non-linear update function. The system could then be linearized and solved using Gaussian elimination using 2^{49} operations. Consider what happens when we have a Grain-like structure in which the attacker somehow gets to know the entire LFSR state. If the output equation for the keystream bit only contains terms from the NFSR which are linearly added along with non-linear terms from the LFSR, then due to the fact that the LFSR is completely known, the expression for the keystream bit becomes a purely linear expression in the initial NFSR state variables and we arrive at a situation that is similar to the one described in the previous paragraph. For Atom, we made sure that the expression for the kesytream bit contains higher degree terms with NFSR bits. Hence the attack of [32] does not apply to Atom.

In [10, 12, 100] algebraic attacks via the method of SAT solvers are proposed against Sprout and Plantlet. The idea is to formulate equations relating the key and the internal state variables to the keystream bits and forward the resulting equation bank to a suitable solver. This approach is in itself slightly problematic against Atom since a part of the LFSR directly decides which key bit is used to update the NFSR. If the LFSR is variable or unknown, the attacker would find it difficult to enumerate any equation bank as it is not known which key bit was used in the state update. Hence to use this approach the attacker has to guess correctly the entire LFSR state in order to proceed with the attack (this imposes a multiplicative complexity 2^{60} to begin with, since the attacker knows that the last 9 bits of the LFSR at the beginning of the keystream phase is always 1). After this, we tried to follow a similar approach as in the above papers and present an equation bank to the solver. On a machine running with an Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz and 12 GB RAM, we could not solve the equations in reasonable time unless we additionally guessed correctly the values of NFSR bits too. A detailed analysis of the distinguishing attack presented in [10] is given in Appendix.

5.4 Hardware Implementation

In Table 5.2, we compare our implementation results with current state of the art hardware-oriented stream ciphers providing 80-bit security Grain-v1, Trivium, Sprout, Plantlet, Lizard

and the 128-bit stream cipher Grain-128. For completeness, we also include AES-CTR in our benchmarks. There are numerous hardware implementations of AES presented in literature: from fully unrolled which performs encryption in one clock cycle, to round based which takes 10 clock cycles per encryption to various serialized circuits which although smaller in hardware size, utilize much more clock cycles for the same purpose. We choose the byte-serial AES implementation presented in which takes around 176 clock cycles to encrypt one plaintext and so when used in counter mode, this circuit can encrypt 128 bits every 176 clock cycles. This is close to the one bit/cycle implementations of the other stream ciphers. We remark the strategy of reducing having a small state size is beneficial for both the circuit area footprint and the overall energy consumption effectively making Atom the most efficient choice among stream ciphers with the same key length of 128 bits.

For a more detailed inspection of the Atom circuit components, we present a break-up of the area shares taken by the various components of the circuit in the design of Atom using the standard cell library of the STM 90nm logic process. As can be seen, a major part of the circuit area (around 56%) is occupied just by the registers. The two key filters occupy 303 GE in total, which means that the additional innovation cost us around 150 GE. As can be seen our design is quite competitive in performance as compared with Grain 128. A high-level diagram of the circuit is present in Figure 5.2. As can be seen, Init represents the signal which is high only during the initialisation phase. The LFSR is indeed partitioned in to two sections of 60 and 9 bits each. We also have a dedicated decimal counter that updates the last 9 bits of the LFSR during initialisation which is discontinued as soon as the initialisation phase is completed. The key filters are arranged so that the $k_{t \bmod 128}$ component is added to the NFSR update only after the initialisation phase is completed.

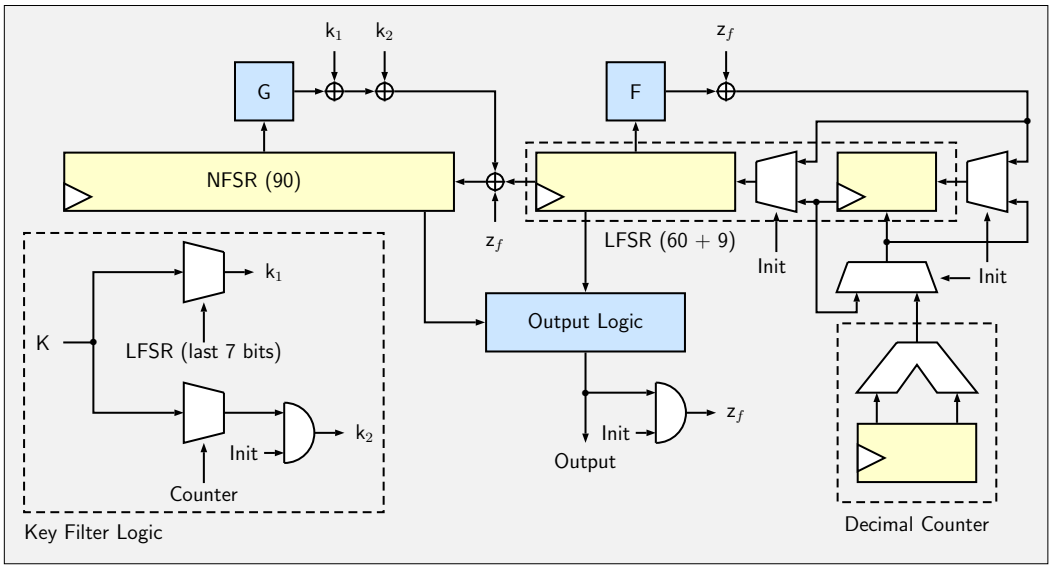


Figure 5.2: High-level circuit diagram. Note that the logic required to load the IV and constant pad on to the registers is not explicitly shown.

If any application requires higher throughput relative to hardware area, Table 5.2 shows

that this can be achieved with Grain or Trivium. Hence for some speeds, these ciphers offer a much better area/performance trade-off. Also note that, for Grain/Trivium it is possible to compute a lot of bits in parallel relatively easily, because these designs deliberately leave last few bits in the registers untapped. While these designs can afford to do this, because their register sizes are 2-3 times the key size, for small-state designs this is undesirable, for following reasons: (1) It decreases the sampling resistance of the cipher and so a TMD trade-off attack via BSW sampling may become feasible, (2) prevent situations like [10, Section 5.2] in which if part of the LFSR with some probability is zero for few clock cycles, the keystream can be expressed as a function of a much smaller part of the internal state, and (3) another reason why parallelizing is counterproductive in Atom-like with key-filter, is that for any x-times parallelized implementation, we will need $2x$ keybits to update the state, which requires $2x$ key-filters. Since each filter is a 128-to-1 multiplexer, this increases the area of the design multiple folds.

5.5 Conclusion

In this chapter, we proposed the Atom stream cipher with internal state only around 25% larger than the secret key. Since all such attempts in the past had some cryptanalytic advances reported against them, our aim was to see if there were any high level architectural modifications that could make such designs immune against common cryptanalytic methods. As a result we adopted a Grain-like structure with an additional key filter that seems to protect against most cryptanalytic advances reported against small state ciphers. Atom currently stands as the most area and energy efficient construction among stream ciphers of the same key size of 128 bits.

[...] I get that this is heinous [...]

Table 5.2: Hardware measurements of Atom and other hardware-oriented stream cipher algorithms. The power and energy figures were obtained at a clock frequency of 10 MHz. We list the energy metric for the encryption of small message batch of 128 bits and a large one of 1.28 Mbits. All constructions were synthesized using the *compile_ultra* directive. Note that the AES-CTR circuit a byte-serial construction.

Library	Scheme	State/Key	Area		Init. Latency	Critical Path	Max. TP	Power	Energy (nJ)	
		Bits	μm^2	GE	Cycles	ns	MBits/s	μW	128 Bits	1.28 Mbits
NanGate 15 nm	Grain-v1	160/80	320.3	1629	160	0.28	3558.7	90.6	0.261	1159.9
	Trivium	288/80	514.9	2619	1152	0.23	4386.0	144.8	1.853	1855.1
	Sprout	80/80	209.9	1068	320	0.17	6044.9	58.9	0.264	753.7
	Plantlet	101/80	245.4	1248	320	0.17	6002.0	69.2	0.310	886.1
	Lizard	121/120	379.0	1927	256	0.17	5744.2	90.1	0.346	1153.4
	Grain-128	256/128	482.2	2453	256	0.24	4130.0	141.9	0.545	1816.7
	Kreyvium	288/128	989.7	5034	1152	0.21	4829.3	246.2	3.151	3154.2
	Trivium-MB	288/128	537.4	2734	1152	0.21	4678.1	148.6	1.902	1903.8
	Trivia	384/128	686.1	3489	1152	0.19	5270.6	191.5	2.451	2453.4
	AES-CTR	128/128	648.8	3300	176	0.33	2184.6	283.2	0.997	4984.8
	Atom	159/128	446.0	2268	511	0.21	4774.6	129.2	0.826	1654.4
NanGate 45 nm	Grain-v1	160/80	1161.9	1456	160	0.63	1587.3	342.5	0.986	4384.5
	Trivium	288/80	1857.6	2328	1152	0.51	1960.8	548.3	7.018	7024.6
	Sprout	80/80	768.7	963	320	0.62	1612.9	227.0	1.017	2906.3
	Plantlet	101/80	873.3	1094	320	0.76	1315.8	242.1	1.085	3099.7
	Lizard	121/120	1326.0	1662	256	0.81	1234.6	386.9	1.486	4953.3
	Grain-128	256/128	1685.6	2112	256	0.87	1149.4	453.5	1.741	5806.0
	Kreyvium	288/128	3441.6	4313	1152	0.76	1315.8	824.9	10.559	10568.2
	Trivium-MB	288/128	1947.9	2441	1152	0.51	1960.8	557.9	7.140	7146.9
	Trivia	384/128	2482.6	3111	1152	0.73	1369.9	725.1	9.281	9289.6
	AES-CTR	128/128	2340.3	2933	176	2.05	354.8	729.5	2.568	12840.5
	Atom	159/128	1613.3	2022	511	1.25	800.0	431.8	2.759	5529.2
UMC 65 nm	Grain-v1	160/80	1836.0	1275	160	2.73	366.3	112.5	0.324	1440.2
	Trivium	288/80	2852.6	1981	1152	2.92	342.5	187.8	2.404	2406.0
	Sprout	80/80	1265.8	879	320	3.12	320.5	73.9	0.331	945.5
	Plantlet	101/80	1452.6	1009	320	3.19	313.5	84.9	0.380	1086.4
	Lizard	121/120	2272.0	1578	256	3.90	256.4	104.1	0.400	1332.7
	Grain-128	256/128	2824.2	1961	256	3.57	280.1	173.4	0.666	2220.0
	Kreyvium	288/128	5613.5	3898	1152	3.68	271.7	314.1	4.020	4024.1
	Trivium-MB	288/128	3031.6	2105	1152	2.83	353.4	194.2	2.486	2488.0
	Trivia	384/128	3871.0	2688	1152	3.79	263.9	246.7	3.158	3160.6
	AES-CTR	128/128	3960.7	2751	176	8.97	81.1	284.9	1.003	5014.7
	Atom	159/128	2835.4	1969	511	6.22	160.8	145.0	0.927	1856.7
TSMC 90 nm	Grain-v1	160/80	3512.5	1246	160	1.98	505.1	316.2	0.911	4047.9
	Trivium	288/80	5508.6	1953	1152	1.92	520.8	535.0	6.848	6854.2
	Sprout	80/80	2371.5	841	320	2.68	373.1	184.6	0.827	2363.5
	Plantlet	101/80	2738.4	971	320	2.61	383.1	221.5	0.992	2835.9
	Lizard	121/120	4125.6	1463	256	2.86	349.7	269.6	1.035	3451.6
	Grain-128	256/128	5239.1	1858	256	1.56	641.0	486.2	1.867	6224.6
	Kreyvium	288/128	10318.0	3659	1152	2.69	371.7	924.2	11.830	11840.4
	Trivium-MB	288/128	5702.7	2022	1152	2.28	438.6	499.7	6.396	6401.9
	Trivia	384/128	7353.1	2607	1152	2.25	444.4	709.7	9.084	9092.3
	AES-CTR	128/128	7545.7	2676	176	5.71	127.4	723.4	2.546	12733.1
	Atom	159/128	4790.3	1699	511	3.20	312.5	297.5	1.901	3809.5

...and Other Optimisations Book II

6 Area: Serial Encryption Circuits

[...] Skin graft machinery [...]

We inaugurate the first chapter of the second book with a treatise on serialised block cipher circuits. In an allegorical interpretation of the matter, serialisation epitomises the zenith of cryptographic optimisation in hardware, a sacrifice of Iphigenia as an *exitus acta probat* for the supreme overarching goal of minimising the area footprint of a construction. More specifically, a serialised block cipher trades a reduced data path and thus lighter combinatorial layers with an increase in latency. Algorithms that adhere to this methodology are aplenty. Most notably, in the context of this chapter, the Atomic-AES circuits by Banik et al. [16, 17] that are eponymous for AES designs with a data path width of 8 bits that compute a single encryption in 226 and 246 clock cycles respectively. The Atomic-AES proposal was decisively inspired by the seminal 8-bit serial AES Threshold Implementation by Moradi et al. [107] from a few years prior. The striking feature of Atomic-AES lies in the particularity of only using a single S-box circuit S-box circuit. Any AES circuit that deploys one S-box can not perform one AES round in fewer than twenty cycles. i.e., sixteen in the Substitution layer and four in the Key schedule. In this regard, the Atomic-AES circuit is close to the optimum when it comes to the overall latency. A straightforward way to convert a byte-serial circuit to a bit-serial equivalent is to chain the flip-flops in a byte register sequentially, so that it takes eight instead of one cycle to transfer one byte of information. Consequently, a trivial conversion of the Atomic-AES circuit into a bit-serial equivalent would require $21 \cdot 8 = 168$ cycles to execute one AES round. This was subsequently improved to 160 cycles by Jean et al. [86] in their bit-sliding work where 128 clock cycles are used to rotate bits across the state pipeline and perform AddRoundKey and SubBytes operations simultaneously. Precisely eight cycles are used for ShiftRows and 32 more for the MixColumns layer. The key insight elaborated throughout this chapter lies in a novel way of enhancing the flexibility of scheduling operations, which is to say neither is it necessary to wait for the AddRoundKey and SubBytes operation to be completed on the entire 128-bit state to begin ShiftRows, nor wait for the ShiftRows to complete to begin MixColumns. When a group of bits in the state have undergone AddRoundKey and SubBytes, we can trigger the ShiftRows calculation on those bits immediately and the same holds for the scheduling of MixColumns vis-à-vis ShiftRows. In the process of developing this technique, we find that not all operations of a round is finished in the time allocated for the round, and so we improvise and try to get them done in the next round, while trying to

maintain functionality at all times. A preliminary version of the aforementioned technique was applied by Banik et al. [11] to the permutation layers of GIFT-64 and PRESENT, but lacks generalisation to other families of block ciphers or wider data paths.

Contributions. None of the implementations reported proposed in [86] and [11] achieved a latency per round figure equal to the block size of the underlying block cipher. This leads naturally to the question of the lowest possible number of cycles in which, for example, one AES round can be computed in a bit-serial circuit. Needless to say that it is impossible to execute a full round in fewer than 128 cycles which is the block size of AES, since each bit in the state needs to be rotated around the state pipeline at least once, and this requires 128 cycles. In this chapter, we investigate strategies that allow us to devise bit-serial block cipher circuits for AES, SKINNY and GIFT that compute one round function in exactly n cycles where n is the block size of the algorithm at hand. We proceed in two parts. Firstly, we propose $\{1,4,8\}$-bit serial architecture for the 128-bit block size variants of AES, SKINNY and GIFT. The advantages of our designs are listed below:

- In terms of circuit area, each of our block cipher implementations is an evident contender to be the smallest implementation. Each implementation fully utilizes both the state and the key pipelines. With single-bit data paths, each round consisting of 128-bit is executed exactly in 128 clock cycles. This ensures that we get the maximum throughput from 1-bit serial implementation leading to an approximately twenty percent reduction in latency (in clock cycle units) over the circuits reported in [26, 86]. The AES, SKINNY and GIFT circuits in these papers report a latency of 168, 168, 160 cycles per round respectively. Our circuit design is novel in the sense that both pipelines are continuously active. A preliminary comparison chart for our bit-serial constructions is given in Table 6.1.

- For AES and SKINNY, we further provide byte-serial circuits that execute each round in sixteen clock cycles and thus reduce that latency compared to existing constructions [17, 86] by twenty percent that in the most cases have a smaller area footprint. Ultimately, a 4-bit serial circuit is presented for GIFT in its bitsliced representation as detailed in Section 2.6.2. A preliminary comparison chart for our $\{4,8\}$-serial constructions is given in Table 6.2.

- Each implementation respects the standard ordering of input and output bits. We do not make a non-standard assumption on the ordering of the bits to reduce the area and latency. Namely, we ensure that an implementation from our paper is readily usable from a NIST LWC candidates without having to modify and deal with the ordering the bits.

The second part of this chapter, we devote to the implementation of serial lightweight AEAD schemes as part of the NIST LWC. Using the serial block cipher circuits from the before, we bootstrap SUNDAE-GIFT [15], SAEAES [109], Romulus [74] and SKINNY-AEAD[29]. To the best of our knowledge, these are the smallest block-cipher-based authenticated encryption

schemes reported so far in the bit-serial and 4/8-bit-serial configurations. Note we utilise the *compile_ultra* synthesis directive for all circuits as part of this chapter.

The content of this chapter was distilled in a research paper titled *The Area-Latency Symbiosis: Towards Improved Serial Encryption Circuits* published in the first volume of the IACR Transactions on Cryptographic Hardware and Embedded Systems in 2021 (IACR-TCHES-2021) [9].

Table 6.1: Bit-serial circuit comparison with the state of the art for the NanGate 45 nm cell library process. More measurements for our designs using other libraries are provided in the respective sections. The source code for the circuits proposed in [86] is not public and thus the given area figures are not verifiable.

Cipher	Area	Latency (Cycles)		Reference
	GE	Round	Full	
AES-128[22]	1982	168	1776	[86]
AES-128	2029	168	1776	[86]
AES-128	1974	128	1408	Section 6.2
SKINNY-128-128	1740	168	6976	[86]
SKINNY-128-128	1748	128	5248	Section 6.3
SKINNY-128-256	2501	168	8448	[86]
SKINNY-128-256	2502	128	6272	Section 6.3
SKINNY-128-384	3260	168	9920	[86]
SKINNY-128-384	3263	128	7296	Section 6.3
GIFT-128	1848	160	6528	[26]
GIFT-128	1791	128	5248	Section 6.4

A Note on the Power/Energy Consumption of Serial Circuits. Serialized implementations generally have poor energy efficiency as explained in [18]. There are two principal reasons for this:

- These circuits generally require larger latency than round-based circuits. For example, Atomic-AES [17], has a data path of eight bits, occupies of roughly 2000 GE, and requires 226 clock cycles to execute one full encryption. The latency is twentyfold compared to eleven clock cycles it takes for the round-based circuit to do the same, but it occupies around four times less area than the round-based circuit, as the latter costs approximately 8000 GE. Due to this, the byte-serial circuit takes around 7-8 times more energy than a round-based circuit.
- Any serialized circuit is required to perform many more register write operations. A round-based circuit overwrites its internal register eleven times, once per clock cycle,

[22] This serial AES implementation uses the non-standard row-major bit ordering instead of the usual column-major fashion.

[23] See previous footnote.

Table 6.2: {4,8}-bit serial circuit comparison with the state of the art for the NanGate 45 nm cell library process. More measurements for our designs using other libraries are provided in the respective sections.

Cipher	Area	Width	Latency (Cycles)		Reference
	GE	Bits	Round	Full	
AES-128[23]	2733	8	23	246	[16]
AES-128	2535	8	16	176	Section 6.2
SKINNY-128-128	2076	8	21	872	[30]
SKINNY-128-128	2022	8	16	656	Section 6.3
SKINNY-128-256	2944	8	21	1040	[30]
SKINNY-128-256	2923	8	16	784	Section 6.3
SKINNY-128-384	3831	8	21	1208	[30]
SKINNY-128-384	3825	8	16	912	Section 6.3
GIFT-128	2183	4	33	1352	[26]
GIFT-128	2130	4	32	1312	Section 6.4

with intermediate round output. Consider on the other hand any byte-serial circuit, or more specifically the Atomic-AES circuit. A simple operation of the form

$$b_0, b_1, \ldots, b_{15} \rightarrow b_1, b_2, \ldots, S(b_0 \oplus k_0),$$

which rotates the state by one byte, and substitutes the least significant byte with the output of the AddRoundkey and S-box on the byte b_0 , requires an entire register overwrite. As is obvious, this needs to be done at least fifteen more times for each byte b_i and then again four times for the MixColumns operation for each of the ten rounds. Since each register write requires finite energy, it is obvious that the energy budget shoots up in the process. The energy consumption proportionally increases the more we try to reduce the length of the datapath. Therefore, a bit-serial datapath would by a similar logic, be even more expensive energy-wise.

The logic mentioned above does not extend to power consumption, as power is the time derivative of energy and essentially the rate of energy spent per unit of time. The fewer number of gates a circuit consists of, it is more often the case that its power footprint becomes smaller. As a consequence, serialized circuits should not be used in situations where energy efficiency is a prime concern, but can still be favored in scenarios where power is the target metric. For example, for battery-driven devices energy consumption is a top priority, whereas for applications like medical implants and passive RFID tags that typically do not use the latest CMOS technology, power consumption is very important. In implantable devices, power is a more crucial metric, as the wearer certainly can not tolerate any rise in operating temperature as a side effect of high power consumption over a number of cycles. Although energy is an important metric, our research direction is directed towards these applications

that cannot ignore area and power constraints.

6.1 Generic Approach

A block cipher based on the SPN methodology generally consists of three round function layers, namely key addition, substitution and a linear transformation for diffusion. The linear layer is often a combination of a permutation function and a matrix multiplication. A case in point is AES where the permutation function is the ShiftRows operation and matrix multiplication is performed by the MixColumns operation. In the context of lightweight circuits, we can further classify these operations into two broad classes: Firstly, swap-based operations and secondly operations that are replacement-based. In AES, the SubBytes and MixColumns operations can be seen as replacement-based operations, since they take a finite portion of the AES state and replace them with a new data block of equal length. ShiftRows can be seen as a swap-based operation because it essentially swaps some bits at two different locations of the state vector. Our technique, for implementing an SPN-based block cipher, then consists of finding a good and short sequence of swap operations that corresponds to the swap-based operation, and interleave them with the replacement-based operations. In the case of AES, the byte level, the ShiftRows is a permutation over the set $[0,15]$ which can be formulated as

$$(1,13,9,5) \circ (2,10) \circ (6,14) \circ (3,7,11,15).$$

Assuming that the AES byte order is column-major, i.e., $b_0, b_1, \ldots, b_{15}$, the above notation means that after ShiftRows, b_1 is moved to location 13, b_{13} is moved to location 9 and so on. Note that each of the four permutations correspond to a particular row of the AES state and they commute with each other, so the order of their execution is irrelevant. The above expression can be decomposed further as

$$[(9,13) \circ (5,9) \circ (1,5)] \circ (2,10) \circ (6,14) \circ [(11,15) \circ (7,15) \circ (3,15)].$$

The swaps of the first 4-cycle which decomposes as $(9,13) \circ (5,9) \circ (1,5)$ are not commutative any more, however the operation can be implemented in sixteen clock cycles. For illustration purposes, let us choose the $(11,15)$ swap for this task by placing scan-flip-flop-based byte registers in locations 10 and 14 as shown in Figure 6.1.

The first task lies in executing swap $(1,5)$. We perform the rotate operation, denoted by r, a total of six times on the circuit, and so that bits arrive to positions shown in Figure 6.1b. We proceed by invoking the scan functionality so that in the next cycle bytes 1 and 5 arrive in positions 10 and 14 as shown in Figure 6.1c. Note that in doing so we effectively execute the permutation $\theta = r \circ (11,15)$. The next swap to be executed is $(5,9)$, which corresponds to switching b_1 and b_9 in the current state. By rotating three more times, we reach to the state in Figure 6.1d, where the bytes b_1 and b_9 are in place to be swapped in the next cycle. After executing θ at this point, we reach to the state in Figure 6.1e. The final swap to be performed is $(9,13)$, which as per the previous logic is swapping bytes b_1 and b_{13}. Again it is easy to see that performing $\theta \circ r^3$ over the next four cycles yields the position in Figure 6.1g, where all the

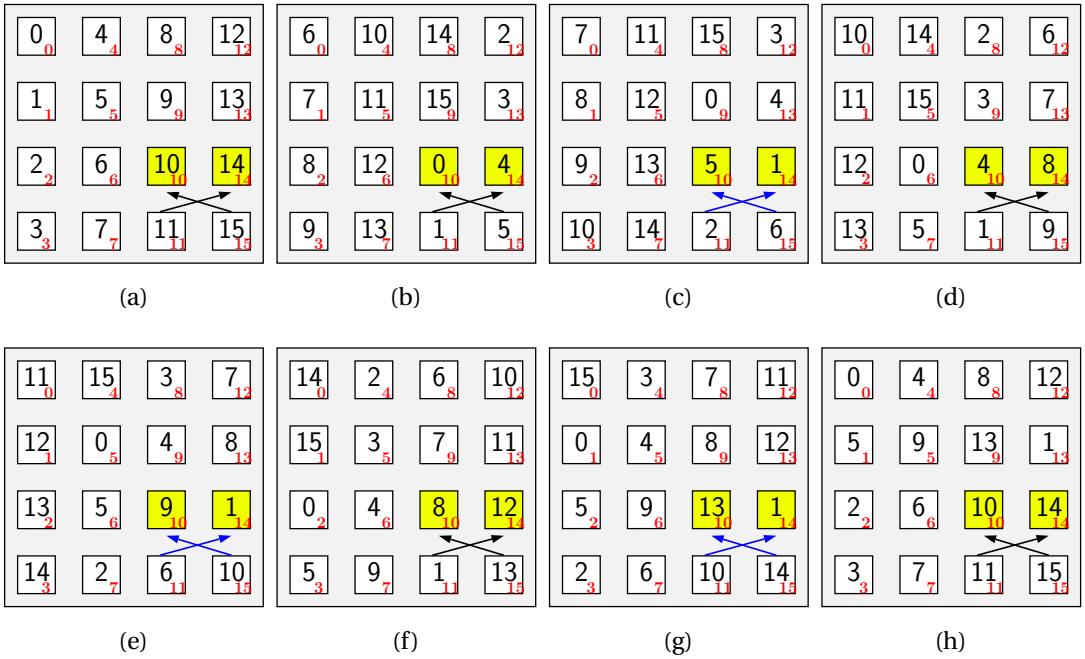


Figure 6.1: The contents of pipeline (a) initially, (b) after r^6, (c) after $\theta \circ r^6$, (d) after $r^3 \circ \theta \circ r^6$, (e) after $\theta \circ r^3 \circ \theta \circ r^6$, (f) after $r^3 \circ \theta \circ r^3 \circ \theta \circ r^6$, (g) after $\theta \circ r^3 \circ \theta \circ r^3 \circ \theta \circ r^6$, (h) finally after $r \circ \theta \circ r^3 \circ \theta \circ r^3 \circ \theta \circ r^6$. Note the numbers in black denote the byte index, i.e., corresponds to b_i, and the subscripts in red denote the fixed register positions. The yellow boxes denote the byte registers implemented with scan flip-flops. Blue and black arrows denote whether the operation θ or r is executed respectively.

bytes have now been swapped as required. We perform the rotate operation once more to get the position in Figure 6.1h, where all bytes are back to the original position and the ShiftRows operation has been executed on the first row. Effectively, the permutation we performed is akin to $r \circ \theta \circ r_3 \circ \theta \circ r_3 \circ \theta \circ r_6$, which takes 16 cycles. Note that if we had chosen any other swap location of the form $(x, x+4)$, it would still be possible to do the above sequence of operations. For instance, if we had chosen the swap $(9,13)$ instead of $(11,15)$, we would need to execute $\theta' \circ r_3 \circ \theta' \circ r_3 \circ \theta' \circ r_8$, where $\theta' = r \circ (9,13)$. This already takes 17 cycles and so all the bytes will be indeed swapped correctly, but not return to their original positions as before. Conceptually this means that if the AES round is executed in sixteen cycles, then a few of the swap operations of the current round would take place in the next round, and we would have to tailor the other operations in the pipeline accordingly.

Following the same logic, let us now try to do the swaps $(2,10) \circ (6,14)$ of the next row. This time, let us choose two swap locations eight places apart, in particular $(5,13)$. The above swaps commute and so can be done in any order, so let us do $(2,6)$ first. After r_{13}, the bytes b_2 and b_6 are in place for swapping, and in the next cycle we execute the scan functionality to perform $\alpha = r \circ (5,13)$. After three more cycles of r, the bytes b_6 and b_{14} are in place, and then we execute α again. Thus by executing $\alpha \circ r_3 \circ \alpha \circ r_{13}$, we have again already spent eighteen cycles. As explained before, this indicates that at this point, the bytes have again

been correctly swapped in terms of their relative order in the pipeline and that in terms of data flow in the circuit, some of the swaps of the current AES round overflow into the next round.

The third set of swaps for the final row is $(11,15) \circ (7,15) \circ (3,15)$. We can construct this sequence with three different swap locations also at distances four, eight and twelve apart. Let us choose the swaps $(11,15)$ and $(5,13)$ as before and $(2,14)$ as the additional swap location. We have to execute $(3,15)$ first, therefore we rotate once to bring the bytes b_3 and b_{15} in place and then execute $\beta = r \circ (2,14)$. We use the swap locations $(11,15)$ and $(5,13)$ which have already been used to perform the swaps in the previous two rows. At this point b_7 and after the previous swap b_3 are already in place and so we execute α on the location $(5,13)$ by invoking its scan functionality. For the last remaining swap $(11,15)$, we have to wait till b_{11} returns to location 11, which requires thirteen more rotations after which we can invoke θ.

We have just put together a set of swap sequences that enable the execution of the AES ShiftRows operation. We looked at each row separately and so it is conceivable that the swap sequences be performed one after the other, thereby requiring a little over 48 cycles. But in the interest of latency, we wish to do them in sixteen and if required within a few cycles of the next round. Since the k-cycles in each row that we executed commute with each other, the swaps can actually be executed concurrently. That is,

1. Invoke the scan functionality on the swap location $(2,14)$ at clock cycle 1 (cycle indexing starting with 0).

2. Invoke the scan functionality on the swap location $(5,13)$ at clock cycles 2, 13 and 17.

3. Invoke the scan functionality on the swap location $(11,15)$ at cycles 6, 10, 14 and 16.

The point is that since the k-cycles commute, we execute the swaps concurrently on the given locations in 18 continuous cycles (numbered 0 to 17) and still achieve the ShiftRows functionality. Indeed it is a matter of a simple arithmetic exercise to see that the arrangement of bytes obtained after executing the above sequence of swaps concurrently in 18 cycles results in ShiftRows off by two extra rotations.

The engineering challenge now revolves around the incorporation of substitution-based operations like SubBytes and MixColumns into a swap-and-rotate pipeline and execute them preferably when the scan functionalities of the registers are not being invoked. Naturally, a completely free placement is not achievable as, for example, the SubBytes and ShiftRows in any round must precede MixColumns. Looking beyond AES can this technique also applied on other constructions? For block ciphers that employ some kind of byte/nibble/word-based swap operations in their permutation function, the answer is affirmative. SKINNY has a permutation function given by

$$(4,5,6,7) \circ (9,11) \circ (8,10) \circ (12,15,14,13).$$

This permutation has a similar form with AES, so it takes modest effort to construct it using swaps, in the same fashion explained above. For block ciphers such as GIFT that employ bit-based permutation function, the technique becomes slightly more involved.

From Byte to Bit-Serial. When we reduce the datapath to one bit, we can no longer swap two bytes in one cycle and it would take exactly eight cycles for every byte swap. At the bit level, ShiftRows of AES is essentially the composition of the following permutations over the set $[0,127]$ for all $k \in [0,7]$ such that

$$(8+k,104+k,72+k,40+k) \circ (16+k,80+k) \circ (48+k,112+k) \circ (24+k,56+k,88+k,120+k).$$

As it can be seen from this expansion, at the bit level, everything scales by a factor of eight.

6.2 AES

Recall the AES description from Section 2.6.4 in which the round function and key scheduling algorithms are detailed. Further recall the flip-flop calling convention in a swap-and-rotate pipeline specified in Section 2.8. Our circuit consists of the following circuits in the main hierarchy: (1) a state pipeline, (2) a key pipeline, (3) controller and (4) a shared S-box.

6.2.1 State Pipeline

The state in our design uses the following components and techniques in a plug-and-play manner:

- The nibble-level MixColumns circuit from bit-sliding project by Jean et al. [86].
- The smallest, in terms of circuit area, known AES S-box proposed by Maximov and Ekdahl [103].[24]

Given that state and key bits are stored in a pipelined fashion, one can easily notice that AddRoundKey can be performed without much hassle as long as each of the state and key pipelines produces the correct bit per clock cycle. Hence, the main challenges on the state pipeline part is to (1) execute all SubBytes, ShiftRows, MixColumns operations simultaneously, (2) complete the operations in 128 clock cycles, while (3) following the standard ordering of bits for the plaintext and the key. Below, we first describe each layer separately, and show how we can fuse them into one operation that executes over the state pipeline continuously.

ShiftRows with Swaps. Assume that the 128-bit pipeline is defined in the same fashion as in Section 2.8, i.e., the bits are loaded into FF_{127} and they are flushed out by FF_0. We use three swap operations to execute the ShiftRows layer: $(80,112)$, $(56,120)$ and $(25,121)$. The timetable for scheduling these swaps is given in Table 6.3.

For simplicity, let us forget about the pipeline and shift operations for the moment, and focus on the nature of ShiftRows in the 16-byte state. We try to express ShiftRows in terms of byte swaps. Suppose that the values contained in the state are the hexadecimal characters 0, 1, ..., F. Considering the standard byte arrangement for loading the initial data, row 0 contains the values 0, 4, 8, C; row 1 contains the values 1, 5, 9, D and so on. We then devise

[24] We will encounter this S-box again in Chapter 8 in a different setting.

Table 6.3: Timetable of operations for the swap-and-rotate bit-serial AES encryption circuit.

Pipeline	Operation	Active Cycles
State	Swap (80, 112)	$[56,64) \cup [88,96) \cup [120,127] \cup [8,16)$
	Swap (56, 120)	$[88,96) \cup [120,127] \cup [0,8)$
	Swap (25, 121)	$\{127\} \cup [0,6]$
	Load S-box	$\{8k+7 : k \in [0,15]\}$
	Load MixColumns	$[32,40) \cup [64,72) \cup [96,104) \cup [0,8)$
Key	Swap (96, 128)	$[0,8)$
	Swap (40, 72)	$[56,64)$
	Load S-box	$\{112\} \cup \{120\} \cup \{0\} \cup \{8\}$
	Key XOR	$[0,96)$
	Add RC	Lookup Table

a sequence of swap operations over the rows 1, 2, 3 to perform ShiftRows. Our three distinct swaps are denoted with distinct colors in Figure 6.2. This figure shows the movement of the bytes as they arrive to their final position implied by ShiftRows. We point out two important observations: (1) each byte-swap operation can be executed by a bit-swap circuit through eight consecutive calls interleaved by shift operations, (2) the swap operations denoted with the same color can actually be executed by a single swap operation as long as it is enabled in the correct clock cycle. Therefore, the choice of swaps and the timetable in Table 6.3 are straightforward extensions of this example into the 128-bit state pipeline.

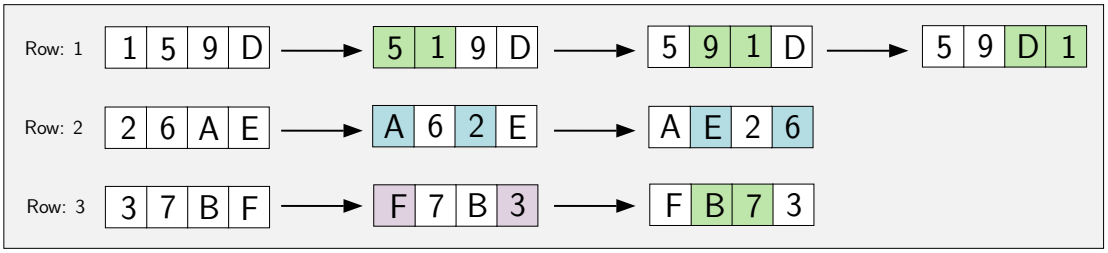


Figure 6.2: The transition diagram for rows 1, 2, 3; where the coloured cells denote the recently modified values. Note that there are three distinct swap operations, with distance 0, 1 and 2 cells in-between.

To help understand how the structure helps perform the ShiftRows operation, we note that since the pipeline is always active, the shift operation is performed in every clock cycle. To additionally perform the swap $80 \leftrightarrow 112$, in any clock cycle, we need to place scan flip-flops at locations 79 and 111 (and wire the output of 80 to the input of 111; wire the output of 112 to the input of 79). Assuming that the bits indexed 0 to 127 enter the pipelines through the location 127, at clock cycle $k \leq 127$, the pipeline stores exactly k bits. For instance, in the 56-th clock cycle, the bits indexed 8, 40 are at locations 80, 112 respectively. Enabling swaps for cycles 56 to 63 therefore swaps bits $8,\ldots,15$ with $40,\ldots,47$, which are essentially bytes indexed by 1 and 5. It can be verified without difficulty that performing the same swap in

cycles [88,96) actually swaps bytes 1 and 9. This exactly follows the explanation in Section 6.1 using (80,112) as swap locations instead of (88,120).

Nibble MixColumns. In the nibble MixColumns introduced by Jean et al. [86], a multiplication over a single column is completed over 8 clock cycles, updating each nibble at a time. To simplify, we first represent a single column of bytes as 8 vertical nibble vectors as below. Namely, from the pipeline given in Figure 6.3, the vectors M_i are defined for $0 \leq i \leq 7$ as below:

$$M_i := \begin{bmatrix} \mathrm{FF}_i \\ \mathrm{FF}_{i+8} \\ \mathrm{FF}_{i+16} \\ \mathrm{FF}_{i+24} \end{bmatrix} \qquad R(M_0) := \begin{bmatrix} \mathrm{FF}_8 \\ \mathrm{FF}_{16} \\ \mathrm{FF}_{24} \\ \mathrm{FF}_0 \end{bmatrix}$$

The nibble MixColumns architecture employs an additional set of 4 flip-flops to help with the serialized computation of this functionality. Define the vector M_8 to denote this additional internal 4-bit storage this architecture employs. During its eight clock cycle operation, these flip-flops are used to keep the value of the leftmost bit of each one of the four bytes. We define a function *upward* rotation R that rotates the elements in a given vertical matrix by one position, as exemplified above. The circuit essentially performs the following sequence of operations to derive the new value of M_i for each $i = 0, 1, \ldots, 7$, starting from $i = 0$ respectively:

- If $i = 0$, store $M_8 \leftarrow M_0$ before any of the following computation, update $M_i \leftarrow R(M_i) \oplus R^2(M_i) \oplus R^3(M_i) \oplus M_{i+1} \oplus R(M_{i+1})$,
- If $i \in \{3,4,6\}$, also update $M_i \leftarrow M_i \oplus M_8 \oplus R(M_8)$.

In other words, at each clock cycle, based on the internal 7-bit counter, we can execute a single slice of the previous computation. In total, it takes 8 clock cycles for a single column, and 32 clock cycles for the whole MixColumns layer. This serial circuit can be realized with 8 XOR, 8 NAND gates and 4 flip-flops.

Combined State Pipeline. In the controller, the circuit contains an 11-bit counter to keep both the round (4-bit) and the phase (7-bit). We split this counter into two parts and refer to them respectively by variables $0 \leq \text{round} \leq 10$ for the upper 4-bit and $0 \leq \text{count} \leq 127$ for the lower 7-bit.

In contrast to previous work [86], we follow the standard ordering of bits in our implementation. That is given a plaintext and a key, the bits are loaded into the circuit starting from the leftmost bits, and following the natural order. This becomes a crucial aspect of a block cipher implementation, if it is meant to be used in a mode of operation that needs to comply with a fixed standard. At the beginning of its operation, the 11-bit counter is reset to zero. During initialisation, i.e. round = 0, the white-colored MUXes in Figure 6.3 are configured so that the next bit s of the state is received from the plaintext input port but after the XOR is performed with the key, which is also being loaded at the same time. For round > 0, we select the state bit to be loaded from the exit of the state pipeline.

- *SubBytes.* Meanwhile, we proceed with executing the SubBytes layer, by enabling the S-box at every eighth cycle. More precisely, the S-box is configured to take the flip-flop outputs $\mathrm{FF}_{121}, \mathrm{FF}_{122}, \ldots, \mathrm{FF}_{127}$ and s as input, and the scan flip-flops $\mathrm{FF}_{120}, \ldots, \mathrm{FF}_{127}$ are instructed to load the output from the S-box whenever count mod 8 = 7.

- *ShiftRows.* Starting from count = 56, the swap operations become active. Many of the bits need to make a couple of jumps before they are located into their ultimate positions implied by ShiftRows, as demonstrated in Figure 6.2. Hence, position-wise, many bits are incorrectly located and look garbled as they pass through the chain of flip-flops $\mathrm{FF}_{24}, \ldots, \mathrm{FF}_{120}$. Nonetheless, as soon as they exit the last swap position FF_{24}, they are guaranteed to be in their final position. See Table 6.3 to notice that the last swap operation executed on a layer actually happens when count = 15 in the next round. In other words, performing ShiftRows over the i-th state uses the last 72 cycles of the round i and the first 16 cycles of the round $i+1$, and it is not aligned with the counter round.

- *MixColumns.* The input ports to the nibble MixColumns circuit are flip-flops FF_i for $i \in \{0,1,8,9,16,17,24,25\}$, and the output ports are input to the exit of the multiplexer of the pipeline and $\mathrm{FF}_7, \mathrm{FF}_{15}, \mathrm{FF}_{23}$ respectively. The MixColumns of round i is performed at round = $i+1$ and it is active during $0 \leq$ count mod $32 \leq 7$, except the last round where MixColumns must be skipped.

- *Overlaps.* There are two clock cycles, i.e., count values, during which two operations modify the same FF simultaneously in Table 6.3. First, at clock cycle 127 both S-box and swap $(25,121)$ attempts to overwrite FF_{120}. Here, the operation precedence is given to the S-box, meaning that the leftmost output bit of the S-box is fed to the swap operation (instead of FF_{120}). A second overlap occurs when count = 3, as MixColumns circuit attempts to read FF_{25} before its value is updated correctly by the swap $(25,121)$. Here, the precedence is given to the swap operations, meaning that the output of the swap operation is fed as input to MixColumns circuit instead of FF_{25}. The AES state pipeline is shown in Figure 6.3.

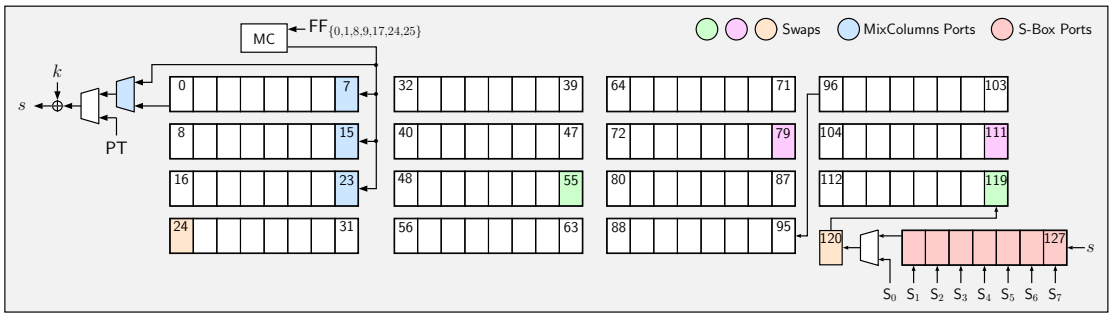


Figure 6.3: The state pipeline of the swap-and-rotate bit-serial AES encryption circuit with coloured scan flip-flops. S-box output ports are denoted with $S_0||S_1||\cdots||S_7$.

6.2.2 Key Pipeline

Recall the key schedule update function from Section 2.6.4 in the preliminaries chapter. In other words, if $K_0, K_1, \ldots, K_{15}$ represent the key bytes of a particular round. Then the next round key sequence $K_{16}, \ldots, K_{31}$ is computed as

$$\begin{bmatrix} K_{16} & K_{20} & K_{24} & K_{28} \\ K_{17} & K_{21} & K_{25} & K_{29} \\ K_{18} & K_{22} & K_{26} & K_{30} \\ K_{19} & K_{23} & K_{27} & K_{31} \end{bmatrix} \leftarrow \begin{bmatrix} K_0 & K_4 & K_8 & K_{12} \\ K_1 & K_5 & K_9 & K_{13} \\ K_2 & K_6 & K_{10} & K_{14} \\ K_3 & K_7 & K_{11} & K_{15} \end{bmatrix} \oplus \begin{bmatrix} \mathsf{S}(K_{13}) \oplus \mathsf{RC} & K_{16} & K_{20} & K_{24} \\ \mathsf{S}(K_{14}) & K_{17} & K_{21} & K_{25} \\ \mathsf{S}(K_{15}) & K_{18} & K_{22} & K_{26} \\ \mathsf{S}(K_{12}) & K_{19} & K_{23} & K_{27} \end{bmatrix}$$

The first column requires special treatment, because it involves S-box calls, and the remaining three columns can be updated smoothly (by simply XORing with a neighboring bytes). In particular, one can notice the disarrangement in the update of the first column, as it takes the current last columns bytes with a downward rotation (by one byte). If we implement this in a straightforward fashion by updating each byte when they arrive to position 0, we would have to choose the input of the S-box either from the position 13 (for computing K_{16}, K_{17}, K_{18}) or 9 (for computing K_{19}). This means that we would have to put an extra 8-bit MUX to choose which value needs to be fed to the S-box. Instead, we decided to temporarily move the byte K_{12} to position 13 before it is fed to S-box, and then return back to its original position after the S-box operation is done. Therefore the pipeline performs the following operations in sequence:

- In the first eight clock cycles, we activate the swap $(96, 128)$ so that the key byte K_{12} is temporarily moved such that it comes after K_{15}. Here, FF_{128} actually refers to the new key bit that is about to be loaded into the key pipeline. With this operation, the key pipeline contains $K_{13}, K_{14}, K_{15}, K_{12}$, in given order. Hence, it respects the order they are being used to update the first key column.

- In clock cycles 112, 120 (of the current round) and 0, 8 (of the next round); the S-box is used by the key pipeline. During these cycles, the S-box reads $K_{13}, K_{14}, K_{15}, K_{12}$ from $\mathrm{FF}_{120}, \ldots, \mathrm{FF}_{127}$ in given order. The output from the S-box is XORed with $\mathrm{FF}_{16}, \ldots, \mathrm{FF}_{23}$ and the result is loaded into $\mathrm{FF}_{15}, \ldots, \mathrm{FF}_{22}$.

- The round constant is added as the bit FF_{24} is loaded into FF_{23}. We use a lookup table to decide when the round constant bit is enabled. In total, this bit is enabled 16 times during the whole encryption.

- During the clock cycles $[56, 64)$, we activate the swap $(40, 72)$ to return K_{12} back to its original relative position. Hence the internal ordering of the bytes becomes K_{12}, K_{13}, K_{14}, K_{15} again.

- For the rest of the key bits, we handle the key scheduling by activating $\mathrm{FF}_{31} \leftarrow \mathrm{FF}_0 \oplus \mathrm{FF}_{32}$ during the clock cycles $[0, 96)$. The full key pipeline is shown in Figure 6.4.

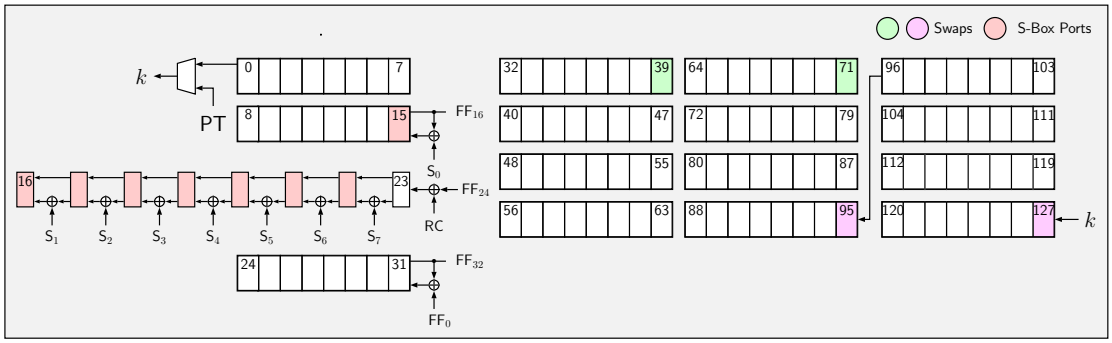


Figure 6.4: The key pipeline of the swap-and-rotate bit-serial AES encryption circuit with coloured scan flip-flops. S-box output ports are denoted with $S_0||S_1||\cdots||S_7$.

6.2.3 8-Bit Datapath

As already stated, there are several implementations of AES with a byte-serial datapath that can execute one AES round in 21 cycles [17, 107]. Since it is not possible to implement the circuit in less than twenty cycles in the presence of only a single S-box, this represents a close-to-optimal latency for this datapath. However, note that these two circuits adopt a non-standard, row first arrangement of bytes. One of our goals therefore was to design a circuit that uses standard byte ordering. Since there already exists a 21-cycles-per-round circuit that achieves close to optimal latency, we did not attempt to design one that also achieves 20 cycles per round. Instead, we focus on an implementation that closely matches our bit-serial circuit, and achieves one round in 16 cycles, by using two S-box circuits.[25] This circuit closely resembles the bit-serial circuit, all the calculations of swap locations and the time intervals when the swap functionality is invoked basically scale by a factor of eight. It is best to summarize it using the following salient points:

- The circuit has 32 byte-registers Reg_0 to Reg_{15} and Key_0 to Key_{15}, and we use the following swap operations to implement the ShiftRows operation: (1) (9,13) in cycles 7, 11, 15, 0, (2) (6,14) in cycles 11, 15, 0, and (3) (2,14) in cycle 0.
- We use a $\{0,1\}^{32} \rightarrow \{0,1\}^{32}$ MixColumns circuit for this implementation. We selected the smallest known implementation with 92 XOR gates from Maximov [101]. The operation is performed in cycles 0, 4, 8, 12. The inputs are taken from the byte registers in the first column and written in registers 1, 2, 3, 15 in the order from MSB to LSB. This closely resembles the bit-serial circuit.
- The key addition and S-box are done in every cycle.
- The key pipeline uses the swaps (11,15) in cycle 0 and (4,8) in cycle 7. The column addition in the key update is done by calculating $\text{Key}_3 \leftarrow \text{Key}_0 \oplus \text{Key}_4$, in cycles 0 to 11.

[25] In theory, it is possible to implement an AES round in 10 cycles with two S-boxes, but we found that it would be difficult to design a pipeline for such a circuit with a low area footprint. Such implementation would require many multiplexers to arrange the component operations in place and would increase the area significantly.

The synthesis results for our swap-and-rotate bit and byte-serial AES circuits are listed in Table 6.4

Table 6.4: Synthesis figures for the proposed swap-and-rotate serial AES circuits. The power and energy measurements were taken at a clock frequency of 10 MHz.

Library	Area	Latency (cycles)		Throughput	Power	Energy
	GE	Round	Total	Mbit/s	μW	nJ/128-bit
			1-bit			
NanGate 15 nm	2247	128	1408	293.89	18.36	2.58
NanGate 45 nm	1974	128	1408	45.87	142.99	20.13
UMC 65 nm	1736	128	1408	8.71	20.35	2.87
TSMC 90 nm	1663	128	1408	14.50	56.14	7.90
			8-bit			
NanGate 15 nm	2870	16	176	2160.68	23.05	0.41
NanGate 45 nm	2535	16	176	376.94	192.39	3.38
UMC 65 nm	2235	16	176	77.67	26.99	0.47
TSMC 90 nm	2256	16	176	121.89	68.91	1.21

6.3 SKINNY

For our serial SKINNY circuits, we consider the three variants with a block size of 128-bits [30] where the tweakey size is $128z$ bits with $z \in \{1,2,3\}$. For the remainder of this chapter, these variants are referred to as SKINNY-128-128, SKINNY-128-256 and SKINNY-128-384 respectively. From a high-level perspective SKINNY shares similarities with AES. However, it employs more lightweight operations as part of the round function. Prominently, S-box and MixColumns can be realized with significantly smaller circuitry compared to AES. The round function consists of SubCells, AddConstants, AddRoundTweakey, ShiftRows, MixColumns. For the finer details of these layers the reader is referred to Section 2.6.3.

6.3.1 State Pipeline

In the controller, the circuit contains an 13-bit counter to keep both the round (6-bit) and the phase (7-bit). We split this counter into two parts and refer to them respectively by variables $0 \leq \text{round} \leq 56$ for the upper 4-bit and $0 \leq \text{count} \leq 127$ for the lower 7-bit. Due to the fact that SKINNY is already designed with hardware-friendliness in mind, we load the bits into the circuit starting from the leftmost bits, by following the standard [30]. In our implementations the key blocks and the plaintext are loaded simultaneously and completed in 128 cycles. This applies to all three versions of different tweakey sizes. At the beginning of its operation, the 13-bit counter is reset to zero. Then during initialisation, i.e., round = 0, the plaintext is loaded through 1-bit input port, and the key is loaded through z-bit input port into their respective pipelines without modification. We remark that each tweakey block has

its own dedicated input port. Below, we describe the layers of operations executed on the state pipeline, in an order observed by the incoming bits.

- *SubCells.* This layer is executed by enabling the S-box at every 8-th cycle. More precisely, the S-box is configured to read $FF_{120}, FF_{121}, \ldots, FF_{127}$ as input, and the scan flip-flops $FF_{119}, \ldots, FF_{126}$ are instructed to be loaded with the S-box output if count mod 8 = 0.

- *AddConstants.* The round constants are added right after the S-box operation. An XOR gate is placed between FF_{119} and FF_{120}, and the round constant bit is added. We use a 7-bit LFSR circuit to produce the round constant bit.

- *AddRoundTweakey.* The key bits are added at the same position with the round constant bit, i.e., between FF_{119} and FF_{120}. To synchronize this with the key pipeline, the key bits k_0, k_1, k_2 are read from FF_{120} of the key pipeline. The key addition is active during $8 \leq \text{count} < 72$. This corresponds to adding the first half of each tweakey.

- *ShiftRows.* This layer is executed with the swap operations, similar to AES, and the timetable of swaps are given in Table 6.5. Position-wise, bits are incorrectly located and look garbled as they pass through flip-flops $FF_{95}, \ldots, FF_{119}$, but as soon as they exit the last swap position FF_{95}, they are guaranteed to be in their final position.

- *MixColumns.* The input ports to the nibble MixColumns circuit are flip-flops FF_i for $i \in \{0, 32, 64, 96\}$, and the output ports are input to the exit MUX of the pipeline and $FF_{31}, FF_{63}, FF_{95}$ respectively. The MixColumns is active during the first 32 clock cycles of a round.

- *Overlaps.* During the clock cycles $64 \leq \text{count} < 72$ three swaps $(112, 120)$, $(104, 120)$, $(96, 120)$ are active at the same time and overlap at the same flip-flop FF_{120}. The order of execution here is $(96, 120)$, $(104, 120)$ and $(112, 120)$ respectively. A diagram of the SKINNY of the presented state pipeline is depicted in Figure 6.5.

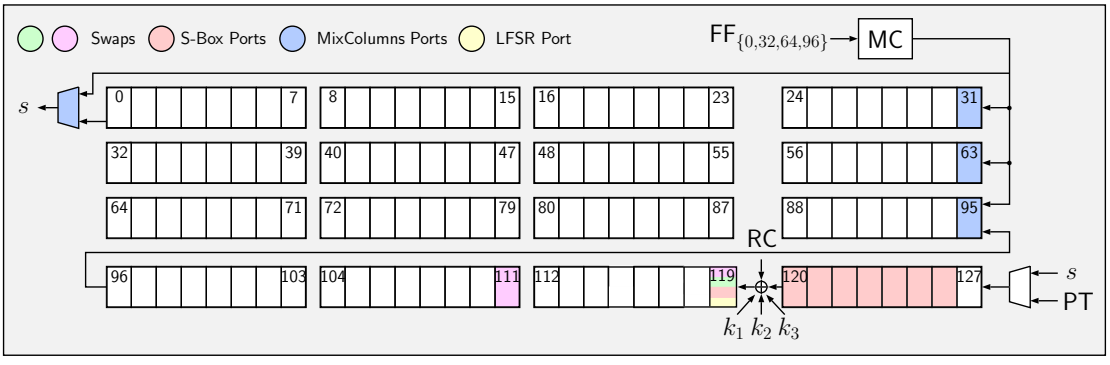


Figure 6.5: The state pipeline of the swap-and-rotate bit-serial SKINNY encryption circuit.

Table 6.5: Timetable of operations for the swap-and-rotate bit-serial SKINNY encryption circuits.

Pipeline	Operation	Active Cycles
State	Swap (112, 120)	$[112,120) \cup [120,127] \cup [0,8) \cup [64,72)$
	Swap (104, 120)	$[64,72) \cup [88,96) \cup [96,104)$
	Swap (96, 120)	$[64,72)$
	Load S-box	$\{8k : k \in [0,15]\}$
	RC Addition	(Lookup Table + LFSR)
	Load MixColumns	$[0,32)$
Tweakey 1,2,3	Swap (56, 120)	$[72,127] \cup [0,8)$
	Swap (48, 56)	$[120,127]$
	Swap (24, 56)	$[112,120) \cup [120,127] \cup [0,8)$
	Swap (8, 24)	$[120,127] \cup [0,8) \cup [24,32)$
Tweakey 2	Swap (0, 1)	$[8k, 8k+6], k \in [0,7]$
	LFSR XOR	$\{8k : k \in [0,7]\}$
Tweakey 3	LFSR (8-bit)	$\{8k : k \in [0,7]\}$

6.3.2 Key Pipeline

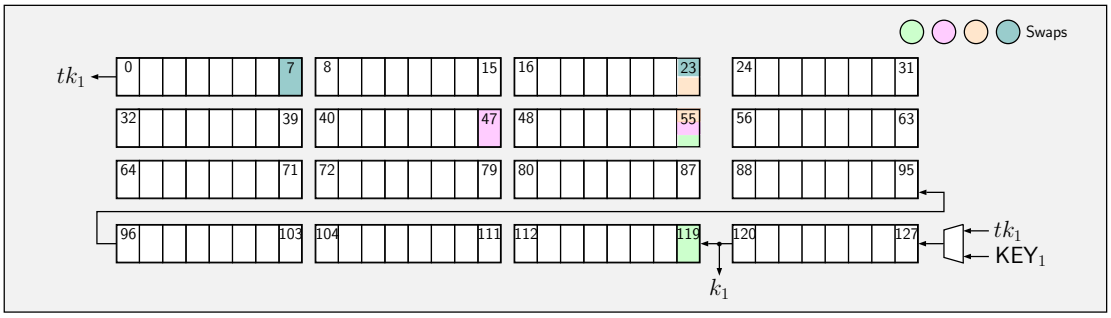
Recall that in the SKINNY key schedule, if $z = 1$ the sixteen key bytes undergo a byte-wise permutation as detailed in Section 2.6.3. Therefore, our key pipelines do the following operations in sequence. First, we swap the first and the last eight bytes by using the swap (56, 120). Then we perform the local byte permutations on the upper half, i.e., the first eight bytes of the key through swaps (48, 56), (24, 56), (8, 24). For $z = 2$ and $z = 3$, some bytes additional enter two LFSR functions F and G. F is applied through another swap (0, 1) and ultimately G is computed by a dedicated 8-bit LFSR circuit. A schematic depiction of the SKINNY key pipelines is given Figure 6.6.

6.3.3 8-Bit Datapath

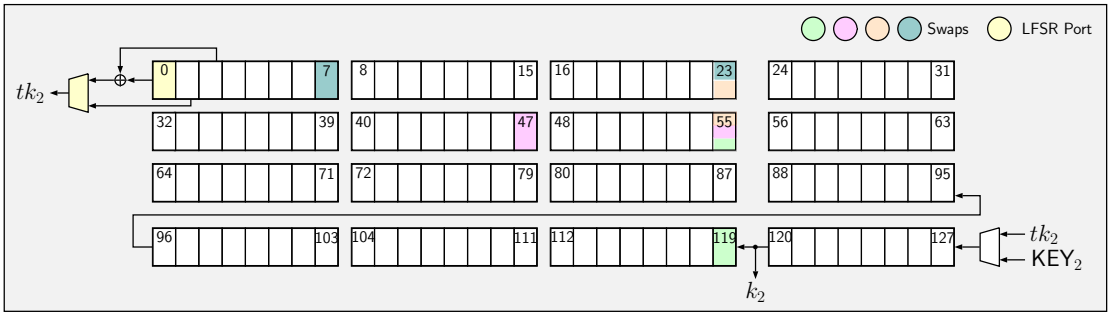
The 8-bit implementation is in fact simpler than the 1-bit equivalent, due to circuitry such as the LFSR and the S-box are already compatible with the datapath size. We only need to add extra gates for swaps, for instance, extend each single swap into byte swap, and duplicate circuit for MixColumns. The timetable is also updated so that each consecutive activity in 8 clock cycles are squeezed into one. Table 6.6 tabulates the synthesis results for the 1/8-bit circuits of SKINNY.

6.4 GIFT

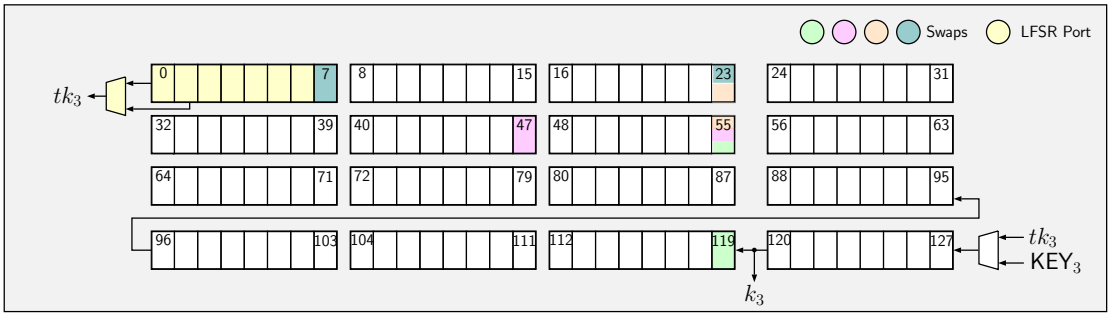
The final block cipher we investigate is the bit-sliced variant of GIFT [26] as utilized in the NIST LWC candidates GIFT-COFB [21] and SUNDAE-GIFT [15]. The bit-sliced representa-



(a) $z = 1$



(b) $z = 2$



(c) $z = 3$

Figure 6.6: The key pipelines of the swap-and-rotate bit-serial SKINNY encryption circuit.

tion of the internal cipher state is detailed in Section 2.6.2. The bit-wise nature of both the algorithm's permutation complicates matters in a swap-and-rotate setting, since each state bit needs to be moved to its designated position individually. As a consequence, a simple solution using few swaps as devised for AES and SKINNY is not achievable.

6.4.1 State Pipeline

Nonetheless, the regular structure of the four sub-permutations of the bit-sliced permutation layer is helpful in finding a sequence of swaps that implement it. As an illustration, take permutation P_0 from Table 2.6 and note that the bits of each byte are moved to positions in which the distance between each bit is 4. The same is true for the other three permutations. Using this property we can devise a set of swaps that implements each permutation independently

Table 6.6: Synthesis figures for the proposed swap-and-rotate serial SKINNY circuits. The power and energy measurements were taken at a clock frequency of 10 MHz.

Library	Area	Latency (cycles)		Throughput	Power	Energy
	GE	Round	Total	Mbit/s	μW	nJ/128-bit
1-Bit SKINNY-128						
NanGate 15 nm	1992	128	5248	277.51	15.89	8.34
NanGate 45 nm	1744	128	5248	41.63	122.06	64.06
UMC 65 nm	1508	128	5248	8.61	15.89	8.34
TSMC 90 nm	1426	128	5248	15.42	47.12	24.73
8-bit SKINNY-128						
NanGate 15 nm	2304	16	656	1051.60	18.62	1.22
NanGate 45 nm	2022	16	656	224.50	146.17	9.59
UMC 65 nm	1800	16	656	59.29	18.51	1.21
TSMC 90 nm	1706	16	656	78.05	51.67	3.38
1-bit SKINNY-256						
NanGate 15 nm	2853	128	6272	208.98	22.99	14.42
NanGate 45 nm	2496	128	6272	34.26	175.28	109.94
UMC 65 nm	2144	128	6272	7.46	23.24	14.58
TSMC 90 nm	2023	128	6272	12.90	69.25	43.43
8-bit SKINNY-256						
NanGate 15 nm	3350	16	784	928.80	27.19	2.13
NanGate 45 nm	2923	16	784	149.27	211.66	16.59
UMC 65 nm	2581	16	784	36.34	26.95	2.11
TSMC 90 nm	2434	16	784	54.28	75.21	5.89
1-bit SKINNY-384						
NanGate 15 nm	3732	128	7296	163.32	30.22	22.04
NanGate 45 nm	3255	128	7296	29.94	229.10	167.15
UMC 65 nm	2791	128	7296	5.43	30.64	22.35
TSMC 90 nm	2647	128	7296	11.44	91.38	66.67
8-bit SKINNY-384						
NanGate 15 nm	4372	16	912	574.88	35.72	3.25
NanGate 45 nm	3825	16	912	110.55	277.45	25.30
UMC 65 nm	3358	16	912	32.59	36.07	3.29
TSMC 90 nm	3155	16	912	52.84	99.29	9.05

in two steps:

1. Reorder each each 32-bit word in the same manner such that two-bit pairs are moved to position where they have a distance of 4. More specifically, we execute the permutation Q as shown below.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$Q(i)$	0	4	1	5	2	6	3	7	8	12	9	13	10	14	11	15
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$Q(i)$	16	20	17	21	18	22	19	23	24	28	25	29	26	30	27	31

This reordering of the bits can be arrived at with three swaps activated in an interval of 32 cycles for all four permutations. In our implementation, we allocated the swaps (28, 31), (29, 31) and (30, 31) for this task.

2. Having executed the permutation Q, all that is left to do is transferring pairs of bits to their correct position which differs depending for P_0, P_1, P_2 and P_3 but can be achieved with some additional swaps.

   - P_0. (24, 28), (4, 22), (10, 22), (16, 22).
   - P_1. (26, 28), (4, 22), (10, 22), (16, 22).
   - P_2. (24, 28), (4, 22), (10, 22), (16, 22).
   - P_3. (22, 28), (4, 22), (10, 22), (16, 22).

All in all, nine individual swaps are necessary to compute the full bit-sliced permutation layer of GIFT.[26] Details concerning the activation cycles can be gathered from Table 6.7 and a diagram of the corresponding state pipeline is shown in Figure 6.7.

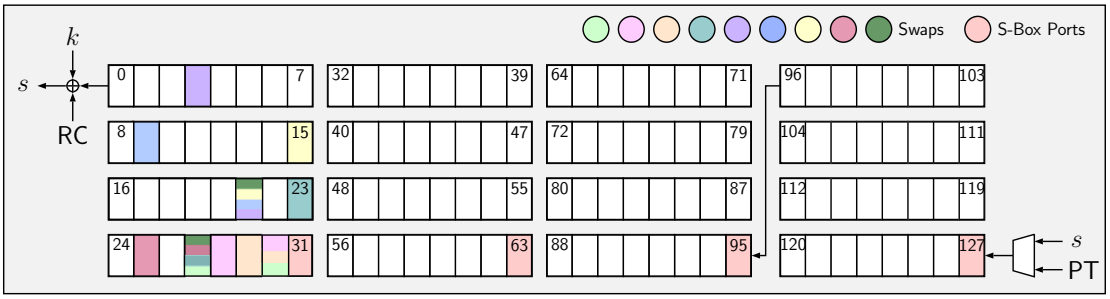


Figure 6.7: 128-cycle, bit-serial GIFT round function implementation using nine swaps. Due to the column-wise application of the substitution layer in GIFT, the S-box ports in the state pipeline are FF_{31}, FF_{63}, FF_{95} and FF_{127} which are active during the cycles 96 to 127.

6.4.2 Key Pipeline

The bit-sliced interpretation of GIFT significantly simplifies how the 64-bit round keys are extracted in each round since they are now mixed into a continuous stretch of the cipher state. For this we can assume, without loss of generality, that the master key K is loaded in

[26] We do not claim that the given sequence of swaps is the most optimal choice when it comes to implementing the bit-sliced GIFT permutation layer but no better alternative has been found after considerable efforts.

the order specified in Section 2.6.2 such that

$$K = \begin{bmatrix} K_0 \,||\, K_1 \\ K_6 \,||\, K_7 \\ K_2 \,||\, K_3 \\ K_4 \,||\, K_5 \end{bmatrix}.$$

In this scenario, the 64-bit round keys $K_2||K_3||K_6||K_7$ are added to the block cipher states during the cycles 32 to 96. The swap sequence for this key schedule is partitioned into four sub-procedures or phases.

- *Phase 1 (State Rotation).* We rotate the entire key state by 64 positions to the left. This operation can be achieved with a single swap during 64 active cycles. Preferably, the transformation should occur concurrently with the addition of the round key into the cipher state, i.e., we allocate (63, 127) to perform the rotation during the cycles 32 to 96.

- *Phase 2 (Precedence Swap).* To achieve a full emulation of the 96-bit rightward rotation of the key schedule, it is further necessary to swap the precedence of the utilized round key halves, i.e., $K_2||K_3$ and $K_6||K_7$. This again only requires a single swap during 32 cycles and can be performed subsequently to the first phase, hence we place a swap (31, 127) for this second phase.

- *Phase 3 (Rotating K_6).* This transformation can been seen as a 14-bit leftward rotation that can be achieved by composing three leftward rotations of magnitude 8, 4, and 2. The position and the interval of those three swaps can be chosen relatively freely, as $K_6||K_7$ is not a part of the current round key, as long as they occur after the second phase has terminated. To simplify the matter, we chose to perform them back-to-back during the cycles 32 and 66. More concretely, the 4-bit rotation is done during the cycles 32 to 44 using a swap at (96, 100). Subsequently, we perform the 8-bit rotation during cycles 44 to 52 with the swap (84, 92), followed by the 2-bit rotation during cycles 52 to 66 using the swap (76, 78).

- *Phase 4 (Rotating K_7).* Phase 3 is followed by a 4-bit leftward rotation of K_7 that is congruent to the 12-bit rightward rotation of the specification. This necessitates a single swap of size 4 for which we can reuse the same swap as utilized in phase 3, in other words, swap (96, 100) during – cycles 48 to 60. A graphical schema for the key schedule pipeline is depicted in Figure 6.8.

6.4.3 4-Bit Datapath

Analogous to the bit-serial implementation presented in the previous section, we now describe the 4-bit-serial architecture that completes execution of a round in 32 clock cycles. The 4-bit state pipeline is unlikely to be achieved by simple swaps and a concurrent rotation

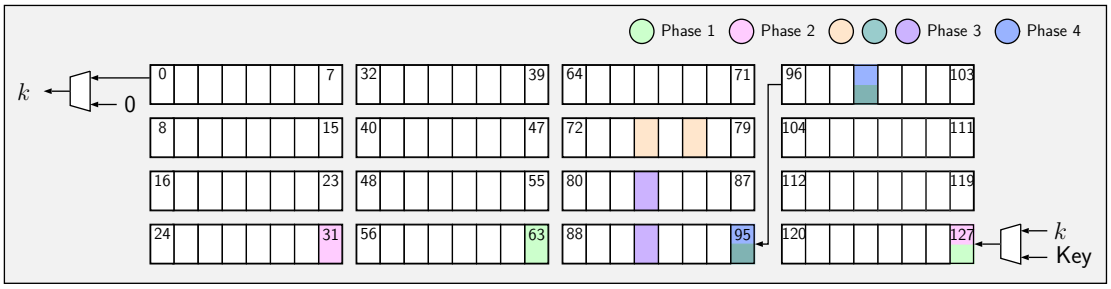


Figure 6.8: 128-cycle, bit-serial GIFT key schedule implementation using five swaps.

as the substitution layer overwrites 4 non-adjacent bits FF_{31}, FF_{63}, FF_{95}, FF_{127}, see Figure 6.7. Then it follows that the swaps performing the permutation must necessarily be placed in the most significant quarter of the state pipeline and each 32-bit row of the state has to be permuted in only 8 cycles. Furthermore since we employed 9 swaps, i.e., 18 scan flip-flops, in the bit-serial construct, we need at the least four times this amount in the 4-bit case. This requires at least 72 MUXed flip-flops which significantly complicates the placement of swaps. A second difficulty arises due to the fact that the pipeline rotates four positions at a time, thus the S-box taps are not constant but move further down the pipeline with every clock cycle, requiring a significant number of multiplexers to differentiate the different taps. In order to circumvent those complexities, we chose to equip the entire 128-bit state with scan flip-flops and execute the permutation in the last cycle of the round while using 4 S-boxes in parallel to substitute 16 bits of the state during the cycles 24 to 31. Note that a single GIFT S-box can be synthesized in fewer than 20 GE, thus the overhead of using four units is marginal and possibly still smaller than using multiplexers for the moving S-box taps.

The 4-bit key pipeline can be seamlessly adapted from the 1-bit counterpart by simply turning the single-bit swaps into nibble swaps, following the generic technique from Section 6.1. As we had 5 swaps in the single-bit version we now have $4 \times 5 = 20$ swaps, i.e., 40 scan flip-flops. In Table 6.8, we list the synthesis results for our 1-bit and 4-bit GIFT circuits.

6.5 AEAD

As standalone block ciphers are not ready-to-use primitives they are usually wrapped in a mode of operation. In this section, we investigate four NIST LWC candidates which are bootstrapped via the improved 1-bit (and 4/8-bit) implementations of AES, SKINNY and GIFT presented in the previous sections. Namely, these candidates are SUNDAE-GIFT [15], SAEAES [109], Romulus [74] and SKINNY-AEAD [29]. For all four schemes, we report the hitherto smallest block-cipher-based authenticated encryption circuits in the literature.

The choice of these four particular candidates in our work is influenced by the observation that the area of a block cipher is determined, to a large extent, by the amount of storage elements, rather than how lightweight the round operations are. This is more evident when one compares SKINNY-384, whose round function comprises lightweight operations, to AES, whose S-box and MixColumns circuits are significantly larger. The former is much larger,

Table 6.7: Timetable of operations for the swap-and-rotate bit-serial GIFT encryption circuit.

Pipeline	Function	Operation	Active Cycles
State	$Q(P_0)$	Swap(28, 31)	101, 103, 109, 111, 117, 119, 125, 127
	$Q(P_0)$	Swap(29, 31)	101, 109, 117, 125
	$Q(P_0)$	Swap(30, 31)	103, 111, 119, 127
	P_0	Swap(24, 28)	104, 105, 112, 113, 120, 121
	P_0	Swap(4, 22)	98, 99, 2, 3
	P_0	Swap(10, 22)	100, 101, 122, 123, 4, 5
	P_0	Swap(16, 22)	102, 103, 114, 115, 124, 125, 6, 7
State	$Q(P_1)$	Swap(28, 31)	5, 7, 13, 15, 21, 23, 29
	$Q(P_1)$	Swap(29, 31)	5, 13, 21, 29
	$Q(P_1)$	Swap(30, 31)	5, 7, 23, 31
	P_1	Swap(26, 28)	6, 7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27, 30, 31, 34, 35
	P_1	Swap(4, 22)	34, 35
	P_1	Swap(10, 22)	26, 27, 36, 37
	P_1	Swap(16, 22)	18, 19, 28, 29, 38, 39
State	$Q(P_2)$	Swap(28, 31)	37, 39, 45, 47, 53, 55, 61, 63
	$Q(P_2)$	Swap(29, 31)	37, 45, 53, 61
	$Q(P_2)$	Swap(30, 31)	39, 47, 55, 63
	P_2	Swap(24, 28)	42, 43, 50, 51, 58, 59, 66, 67
	P_2	Swap(4, 22)	66, 67
	P_2	Swap(10, 22)	58, 59, 68, 69
	P_2	Swap(16, 22)	50, 51, 60, 61, 70, 71
State	$Q(P_3)$	Swap(28, 31)	69, 71, 77, 79, 85, 87, 93, 95
	$Q(P_3)$	Swap(29, 31)	69, 77, 85, 93
	$Q(P_3)$	Swap(30, 31)	71, 79, 87, 95
	P_3	Swap(22, 28)	74, 75, 82, 83, 90, 91, 98, 99
	P_3	Swap(4, 22)	98, 99
	P_3	Swap(10, 22)	90, 91, 100, 101
	P_3	Swap(16, 22)	82, 83, 92, 93, 102, 103
State	Key Addition	-	[32,96)
	RC Addition	-	Lookup Table
	S-Box	-	[96,128)
Key	Phase 1	Swap(64, 128)	[32,96)
	Phase 2	Swap(32, 128)	[96,128)
	Phase 3	Swap(96, 100)	[32,44)
	Phase 3	Swap(84, 92)	[44,52)
	Phase 3	Swap(76, 78)	[52,66)
	Phase 4	Swap(96, 100)	[48,60)

only because it requires large number of flip-flops to store the key. Because an authenticated encryption scheme produces a tag besides the ciphertext blocks, it is natural to expect a particular value that is initialized at the beginning and updated repetitively after processing each new block of data. We refer to this value as the running state. The running state is eventually

Table 6.8: Synthesis figures for the proposed swap-and-rotate serial GIFT circuits. The power and energy measurements were taken at a clock frequency of 10 MHz.

Library	Area	Latency (cycles)		Throughput	Power	Energy
	GE	Round	Total	Mbit/s	μW	nJ/128-bit
1-Bit GIFT						
NanGate 15 nm	2047	128	5248	192.12	15.36	8.06
NanGate 45 nm	1791	128	5248	32.02	122.29	64.17
UMC 65 nm	1555	128	5248	6.84	16.08	8.44
TSMC 90 nm	1480	128	5248	13.43	45.14	23.69
4-bit GIFT						
NanGate 15 nm	2449	32	1312	886.58	17.11	2.24
NanGate 45 nm	2130	32	1312	153.24	152.96	20.07
UMC 65 nm	1919	32	1312	34.57	18.37	2.41
TSMC 90 nm	1819	32	1312	41.80	50.80	6.67

used to compute the tag, so that all blocks contribute to its value. From the area perspective, an important question is whether storing the running state requires an extra register or not. For the chosen candidates, the running state is actually not a separate value, but rather it is passed between consecutive encryption calls. In other words, we can use the state register inside the block cipher to keep this value temporarily until the next encryption starts. It is precisely the reduction in the storage area that yields the impressive area results for the four candidates. In the special case of Romulus, which actually defines six different variants, we decided to implement two members, the primary member N1 and its sibling N3 that is likely to cost the smallest area in ASIC circuit. Romulus-N1 is larger than Romulus-N3, because the latter favors the smaller SKINNY-256, while its other nonce-based siblings all use SKINNY-384.[27]

6.5.1 SUNDAE-GIFT

The SUNDAE-GIFT AEAD scheme was proposed by Banik et al. [15] and is based on the SUNDAE mode of operation [14], featuring the bit-sliced GIFT block cipher at its core. It is a bare-bones construction that does not require any additional registers aside the ones used within the block cipher. After the encryption of the init vector, each data block is mixed into the AEAD state between the encryption calls. A field multiplication over GF(2^{128}) is applied after the last associated data has been added to the state. The same multiplication is also performed for the last message block. The multiplication is either $\times 2$ when the last AD or message block has been padded or $\times 4$ whenever the last blocks are complete without any padding. More formally, the multiplication $\times 2$ is encoded as a byte-wise shift and the addition of the most significant byte into other bytes of the state such that if $B_0||B_1||\ldots||B_{15}$

[27] In order to facilitate a fair evaluation ground between implementations we re-utilise the input/output interface from Section 3.1 for our bit-serial swap-and-rotate AEAD circuits.

represents the 16 bytes of the intermediate AEAD state (with B_0 being the most significant byte), we have that

$$2 \times (B_0, B_1, \ldots, B_{15}) = (B_1, B_2, \ldots B_{10}, B_{11} \oplus B_0, B_{12}, B_{13} \oplus B_0, B_{14}, B_{15} \oplus B_0, B_0),$$
$$4 \times (B_0, B_1, \ldots, B_{15}) = 2 \times (2 \times (B_0, B_1, \ldots, B_{15})).$$

The tag is produced after processing all the AD and message blocks and the ciphertext blocks are generated by reprocessing the message blocks afterwards. A high-level schematic of the SUNDAE-GIFT construction is shown in Figure 6.9.

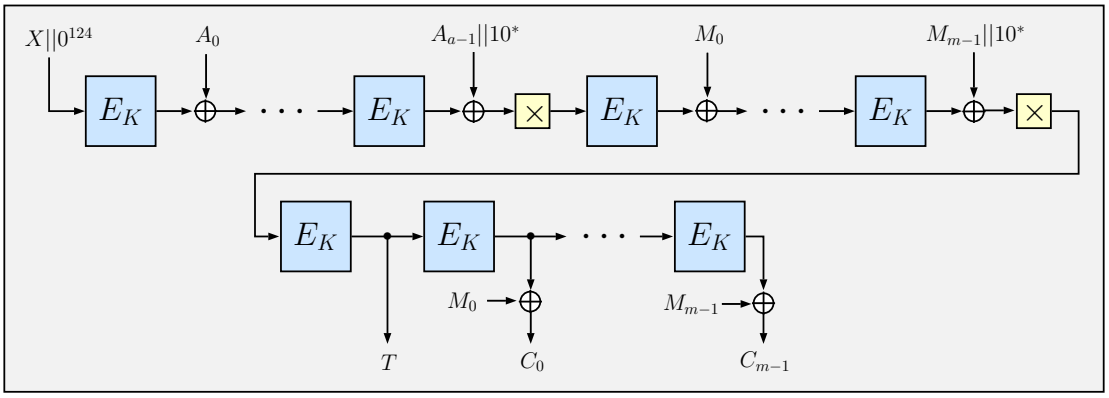


Figure 6.9: The high-level overview of SUNDAE-GIFT, which depicts the processing of m message and a associated data blocks. X denotes a 4-bit parameter, whose value depends on the length of the nonce and whether there are no AD or message blocks.

The simplicity of SUNDAE-GIFT can be exploited in a bit-serial implementation to attain a circuit with very low overhead in terms of area. In fact, except for the slight area increase for the control logic, the sole addition to the swap-and-rotate GIFT circuit presented in Section 6.4 is the field multiplication. The multiplier can be achieved with two swaps (one for $\times 2$, another for $\times 4$) and one XOR gate. More concretely, we allocate 128 rounds for the multiplication $\times 2$ and $\times 4$ during which the block cipher round function and key swaps are disabled. In other words, while the ciphertext bits exit the last round function computation, we place a swap at (119, 127) and activate it during cycles 8 to 127 which rotates the state by 8 positions to the left. Similarly, another swap (111, 127) is active during the cycles 16 to 127 in order to execute the 16-bit rotation. Hence, in the worst case, we require $2 \times 128 = 256$ additional cycles for multiplications. In terms of latency, each new encryption call is loaded with the new plaintext, while the ciphertext bits of the previous computation exit the pipeline. As a consequence the very first encryption operates over $41 \times 128 = 5248$ cycles, while the remaining encryption each take $40 \times 128 = 5120$ cycles. Table 6.9 lists the synthesis measurements for the serial SUNDAE-GIFT constructions.

Table 6.9: Synthesis figures for the proposed swap-and-rotate serial SUNDAE-GIFT circuits. The power and energy measurements were taken at a clock frequency of 10 MHz. Latency and energy correspond to the encryption of one 128-bit AD block and eight 128-bit message blocks.

Library	Area GE	Latency Cycles	Throughput Mbit/s	Power μW	Energy nJ
1-Bit SUNDAE-GIFT					
NanGate 15 nm	2170	92544	91.05	15.90	147.14
NanGate 45 nm	1910	92544	15.17	130.31	1205.94
UMC 65 nm	1671	92544	3.35	15.99	147.98
TSMC 90 nm	1576	92544	5.77	45.98	425.52
4-Bit SUNDAE-GIFT					
NanGate 15 nm	2754	23136	299.93	19.48	45.07
NanGate 45 nm	2339	23136	53.67	168.16	389.05
UMC 65 nm	2093	23136	10.56	18.93	43.79
TSMC 90 nm	2000	23136	13.67	52.12	120.58

6.5.2 SAEAES

The SAEAES AEAD scheme was proposed by Naito et al. [109] and uses the AES block cipher as the underlying encryption core. The white paper proposes a number of parameters according to which the mode can be operated, but the primary candidate among them is SAEAES128-64-128, which implies a key size of 128 bits, message/AD blocks of 64 bits and a tag size of 128 bits. This effectively makes the primary mode of rate 1/2, since 2 block cipher calls are required per 128 bits of message/AD. However, the mode requires no additional state other than those required in the calculation of the block cipher encryption and so a very compact implementation is feasible. A diagram of the SAEAES mode of operations is depicted in Figure 6.10.

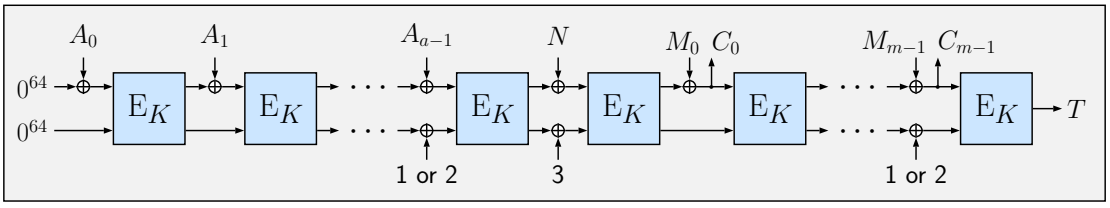


Figure 6.10: The high-level overview of SAEAES, which depicts the processing of m message and a associated data blocks.

From a circuit designer's point of view, it is not difficult to implement the mode, as the only real challenge lies in ensuring that at the beginning of a particular encryption operation the circuit feeds the correct input vectors to the block cipher circuit, which are as follows:

- $X_i = (A_i, 0^{64} \oplus E_K(X_{i-1}))$ or A_0 when $i = 0$ during the associated data processing stage,

where X_i is the i-th input to the block cipher.

- $X_{a-1} = (X_{a-1}, \text{const}_{64} \oplus E_K(X_{a-2}))$ for the last AD block, where const_{64} denotes a 64-bit constant.
- $\text{IV} = N \oplus 0^{126}11 \oplus E_K(X_{a-1})$ before the processing of the plaintext begins, where $0^{126}11$ corresponds to the number 3 encoded as 128-bit string and N denotes the nonce.
- $X_i = M_i \oplus E_K(X_{i-1})$ during the plaintext processing stage, where X_i is the i-th input to the block cipher during plaintext processing. It can also be seen that X_i is also incidentally the i-th ciphertext block, and the tag is simply the outcome of the final encryption call that the mode performs.

A bit-wise AES encryption core produces output 1 bit per clock cycle during the last 128 cycles of the encryption operation. Since we are using no additional storage blocks, the output bits, once produced, need to be XORed with the appropriate input signal and concurrently fed back to the block cipher as the input of the following encryption call. Essentially cycles 1281 to 1408 not only produce the output of the i-th encryption but also serve as the input period for the $(i+1)$-th encryption. Thus one needs to exercise some more fine-grained control over the circuit, to ensure that the block cipher circuit is able to perform the dual role during cycles 1281 to 1408. This effectively means that all encryption calls except the first requires 1280 cycles. Hence, in order to process a AD and m plaintext chunks of 64 bits each, the circuit requires $a+m+1$ encryption calls which leads to $1408+1280\times(a+m)$ cycles. The synthesis figures for the proposes serial SAEAES circuits are tabulated in Table 6.10.

Table 6.10: Synthesis figures for the proposed swap-and-rotate serial SAEAES circuits. The power and energy measurements were taken at a clock frequency of 10 MHz. Latency and energy correspond to the encryption of one 128-bit AD block and eight 128-bit message blocks.

Library	Area GE	Latency Cycles	Throughput Mbit/s	Power μW	Energy nJ
1-Bit SAEAES					
NanGate 15 nm	2362	24448	142.21	18.88	46.01
NanGate 45 nm	2067	24448	21.20	148.82	363.84
UMC 65 nm	1834	24448	4.45	21.67	52.98
TSMC 90 nm	1751	24448	7.11	56.97	138.55
8-Bit SAEAES					
NanGate 15 nm	3086	3056	1119.93	24.93	7.62
NanGate 45 nm	2745	3056	186.27	205.50	62.80
UMC 65 nm	2569	3056	37.60	28.11	8.59
TSMC 90 nm	2452	3056	61.88	70.22	21.46

6.5.3 Romulus

Romulus is an AEAD scheme designed by Iwata et al. [74] making use of the SKINNY family of block ciphers. We provide implementations for two members Romulus-N1 and Romulus-N3. The former is the primary candidate of the family that employs SKINNY-384 whereas the latter is the lightest among them as it uses SKINNY-256.

In order to reduce the number of block cipher calls, and make use of the large tweakey space, that is 384 bits for the primary member, Romulus makes 1/2 block cipher call per associated data block, and 1 block cipher call per message block. Romulus-N1 member admits 128-bit key, 128-bit nonce, variable-length message chopped into 128-bit blocks, and produces 128-bit tag. In terms of input parameter sizes, the difference in Romulus-N3 lies in its 96-bit nonce. An interesting design choice regarding Romulus is that associated data blocks can have alternating size based on which member is chosen. For example, with Romulus-N3, for some integer i, A_{2i-1} blocks are 128-bit, and A_{2i} blocks are 96-bit. In order to ease notation and the description, one can actually treat (A_{2i-1}, A_{2i}) as a single 224-bit block, assuming that the original padding is preserved during this conversion. In Romulus-N1, all associated data blocks are fixed to 128 bits. Figure 6.11 describes the three phases a full AEAD operation passes through, namely processing of (1) associated data, (2) nonce and (3) message blocks.

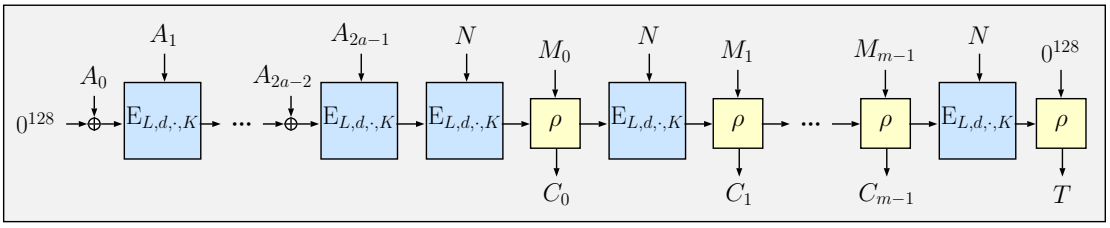


Figure 6.11: The high-level view of Romulus-N1, which depicts the processing of $2a$ associated data and m message blocks. L denotes the 56-bit LFSR that counts the number of processed blocks, and d denotes a single byte domain separator followed by 0^{64}.

During associated data phase, each combined 224-bit (A_{2i-1}, A_{2i}) block is processed with a single block cipher call E_K. For each of these SKINNY-256 calls, the plaintext is A_{2i-1}, and the tweakey is concatenation of 24-bit counter L, 8-bit domain separator d, 96-bit A_{2i} block and the 128-bit key K. The output from the block cipher is treated as the running state, and XORed with each new A_{2i-1} block. Once all (A_{2i-1}, A_{2i}) combined blocks are processed, the running state is encrypted by using the nonce N itself as a part of the tweakey. We refer to this as processing of the nonce. During the message phase, for each of the 128-bit message blocks, the running state and the message block M_i are passed through ρ function defined below. Essentially ρ acts as XOR in the lateral direction, hence the running state is XORed with the message blocks as before. Once all message blocks are processed, the final block cipher output is passed through ρ with 0^{128} to produce the tag. $\rho(S, M) = (S', C)$ is defined as $S' \leftarrow S \oplus M$ and $C \leftarrow G(S) \oplus M$. For each byte, G performs the operation

$$G(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \leftarrow ((x_0 \oplus x_7), x_7, x_6, x_5, x_4, x_3, x_2, x_1).$$

It is then straightforward how we can use 1-bit-serial SKINNY-256 pipeline to realise Romulus-N3. Except for the computation of the ciphertext blocks through ρ, we can simply reuse the state pipeline of SKINNY-256 to store the running state. In order to compute G, we use two external 7-bit buffer pipelines, which keeps the copy of the last 7 bits that exit the state pipeline and the last 7-bit of message block which is being fed to the circuit. This leads to 7 clock cycle of delay in between the time a message block is fed and the time the ciphertext bits become available. This similarly applies to the tag as well, hence the delay of 7 clock cycles must be considered during latency calculation. As for 1-bit-serial implementation of Romulus-N1, the steps taken by the state machine is precisely same. As for differences, however, (1) the invoked block cipher is SKINNY-384, (2) all associated data blocks are 128-bit, hence loading for even and odd-numbered associated data blocks (as well as nonce) starts and ends at the same clock cycles, (3) and the block counter is defined as 56-bit LFSR (instead of 24-bit). Moving towards 8-bit implementation is also quite straightforward, with the only difference being the removal of the 7 clock latency caused by ρ function. Since it operates on the byte level, it is realised as a fully combinatorial circuit.

All in all the 1-bit Romulus-N1 pipeline has latency of, processing 1 AD blocks and 8 message blocks takes $(a+m)\times 56\times 128+128+7$ clock cycles. The additional 128 clock cycles are incurred due to the delay of loading/flushing the pipelines, and the 7 clock cycle is due to the execution delay of ρ. As for 8-bit implementation, the clock cycles are amended as $(a+m)\times 56\times 16+16$. The corresponding synthesis measurements are listed in Table 6.11.

Table 6.11: Synthesis figures for the proposed swap-and-rotate serial Romulus circuits. The power and energy measurements were taken at a clock frequency of 10 MHz. Latency and energy correspond to the encryption of one 128-bit AD block and eight 128-bit message blocks.

Library	Area GE	Latency Cycles	Throughput Mbit/s	Power μW	Energy nJ
1-Bit Romulus-N1					
NanGate 15 nm	4488	64647	79.34	32.08	207.39
NanGate 45 nm	3879	64647	18.43	265.49	1716.31
UMC 65 nm	3415	64647	4.09	31.59	204.22
TSMC 90 nm	3198	64647	7.51	95.27	615.89
8-Bit Romulus-N1					
NanGate 15 nm	5152	8080	608.32	36.77	29.71
NanGate 45 nm	4458	8080	106.57	311.85	251.97
UMC 65 nm	4069	8080	37.15	31.53	25.48
TSMC 90 nm	3742	8080	42.20	100.81	81.45
1-Bit Romulus-N3					
NanGate 15 nm	3310	55431	163.70	25.01	138.63
NanGate 45 nm	2880	55431	21.28	198.99	1103.02
UMC 65 nm	2497	55431	5.62	24.54	136.03
TSMC 90 nm	2361	55431	9.99	73.95	409.91

6.5.4 SKINNY-AEAD

SKINNY-AEAD as proposed by Beierle et al. [29] relies on the ΘCB3 mode of operation [95] and uses the heaviest SKINNY variant, i.e., SKINNY-384, as the core block cipher. ΘCB3 requires the addition of three auxiliary registers that store intermediate values during the computation; a 128-bit register denoted by X that accumulates the encrypted AD block, a second 128-bit register Y that holds the summation of all message blocks and finally a 64-bit LFSR block counter L as shown in Figure 6.12. Both the 1-bit and 8-bit version of SKINNY-AEAD can be instantiated without any further modifications to the serial SKINNY-384 cores. Table 6.12 tabulates the synthesis figures for the proposed serial SKINNY-AEAD circuits.

Table 6.12: Synthesis figures for the proposed swap-and-rotate serial SKINNY-AEAD circuits. The power and energy measurements were taken at a clock frequency of 10 MHz. Latency and energy correspond to the encryption of one 128-bit AD block and eight 128-bit message blocks.

Library	Area GE	Latency Cycles	Throughput Mbit/s	Power μW	Energy nJ
1-Bit SKINNY-AEAD					
NanGate 15 nm	6732	72960	75.29	46.10	336.35
NanGate 45 nm	5980	72960	15.21	408.43	2979.19
UMC 65 nm	5105	72960	4.89	45.72	333.57
TSMC 90 nm	4807	72960	6.84	122.55	894.12
8-Bit SKINNY-AEAD					
NanGate 15 nm	7025	9856	569.94	31.99	31.53
NanGate 45 nm	5992	9856	101.43	410.75	404.83
UMC 65 nm	5258	9856	21.60	38.76	38.20
TSMC 90 nm	4944	9856	38.61	137.05	135.08

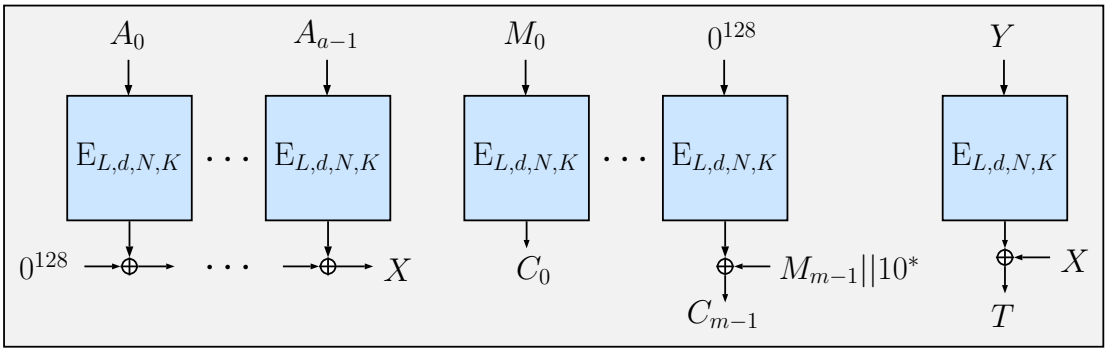


Figure 6.12: The high-level view of the SKINNY-AEAD construction. The block counter L, a domain separator d, the nonce N and the encryption key K together make up the 384-bit tweakey. The encryption of the zero string is only performed when the last message block is incomplete.

6.6 Conclusion

Let us reiterate the content of this chapter. The synthesis results of AES, SKINNY and GIFT are summarized in Table 6.4, Table 6.6 and Table 6.8 respectively. For each of the three block ciphers, we observe that in the bit-serial mode, the area occupied by the circuits is exceedingly close to the total area required by the storage elements. For AES, the area is only slightly larger due its use of an 8-bit S-box and a reasonably heavyweight MixColumns circuit. Nonetheless, for the other ciphers that have relatively lightweight S-box and linear layer, the purely combinatorial circuit elements occupy only around 10% of the total silicon area. Additionally, we are able to reduce the round latency to match precisely the block size of the underlying block cipher. Note that it is not possible to have an implementation that has lower round latency in clock cycles than the block size of the cipher, because for a bit-serial circuit of SPN-based ciphers, all the state bits must be rotated across the pipeline. Therefore, this represents the optimal trade-off spot in the area-latency curve, as far as SPN-based block ciphers are concerned. In the case of implementations with wider datapaths, the area grows only marginally, mainly because the number of MUXes and XOR gates required in the circuit needs to be multiplied by the length of the datapath the circuit aims to achieve.

Table 6.9, Table 6.10, Table 6.11 and Table 6.12 list the synthesis results we obtained for all the individual modes of operation. SUNDAE-GIFT and SAEAES are essentially rate 1/2 modes that need 2 block cipher calls for every 128-bit message block. Note that for these two, the underlying block ciphers admits an 128-bit key, and they require exactly 256 flip-flops to store the key and the state. Thus in a sense, minimalism of the core block cipher comes at the cost of having to execute 2 block cipher calls per 128-bit message block. On the other hand, the rate 1 modes, which require only 1 block cipher call per block of message, such as Romulus and SKINNY-AEAD employ SKINNY-384. They take advantage of the large (384-bit) tweakey space to accommodate nonce, domain separator, and counter for each block cipher invocation. However, for SKINNY-384, this comes at the cost 512 flip-flops for both the state and the tweakey. This results in an interesting latency and area trade-off, and gives further insights on the nature of these designs.

[...] To burn up on re-entry and call the state a traitor [...]

7 Area: A Small GIFT-COFB

[...] Sputnik sickles found in the seats [...]

We remain in the dominion of bit-serial circuits and take on a challenge to had not been solved at the time of publication of the findings in Chapter 6. More specifically, we will solve the riddle of efficiently serialising the NIST LWC finalist GIFT-COFB [21] AEAD scheme to a data path width of one bit. Unlike the bit-serial AEAD implementations proposed in Chapter 6, GIFT-COFB involves finite field arithmetic for which there is no straightforward mapping into a bit-serial setting that is both circuit area and latency efficient.

Contributions. In this chapter, we fill this gap by proposing bit-serial circuits, based on the swap-and-rotate GIFT* circuit that was presented earlier, which stand as the to-date most area-efficient GIFT-COFB implementations known in the literature. More specifically, our contributions are summarized as follows:

1. GIFT-COFB-SER-S: This circuit represents an effective transformation of the swap-and-rotate GIFT* scheme into the GIFT-COFB mode of operation minimizing its area footprint.

2. GIFT-COFB-SER-F: Subsequently, we observed that the interspersing of block cipher invocations with calls to the finite field module as found in the baseline GIFT-COFB design can be reordered by leveraging its inherent mathematical structure in order to further optimize the overall latency of GIFT-COFB-SER-S while only incurring a modest area increase.

3. GIFT-COFB-SER-TI: In a natural progression, we design a bit-serial first-order threshold implementation based on GIFT-COFB-SER-F whose security is experimentally verified through statistical tests on signal traces obtained by measuring the implemented circuit on a SAKURA-G side-channel evaluation FPGA board.

4. We synthesise all of the proposed schemes on ASIC platforms and compare our results to existing bit-serial implementations of NIST LWC candidate submissions, indicating our designs are among the smallest currently in the competition. A brief overview of the synthesis results is tabulated in Table 7.1.

Table 7.1: Synthesis results overview for lightweight block cipher based NIST LWC competitors using the TSMC 90 nm cell library at a clock frequency of 10 MHz. Latency and energy correspond to the encryption of 128 bits of AD and 1024 message bits. Highlighted schemes are NIST LWC finalists. GIFT-COFB-SER-TI is based on GIFT-COFB-SER-F.

	Datapath	Area	Latency	Power	Energy	Reference
	Bits	GE	Cycles	μW	nJ	
SUNDAE-GIFT	1	1576	92544	45.9	424.7	Chapter 6
SAEAES	1	1751	24448	56.9	139.0	Chapter 6
Romulus-N3	1	2361	55431	74.0	409.9	Chapter 6
SKINNY-AEAD	1	4807	72960	122.5	893.8	Chapter 6
GIFT-COFB	128	4710	400	69.3	2.77	Chapter 3
GIFT-COFB-SER-S	1	1907	93312	42.8	399.3	Section
GIFT-COFB-SER-F	1	2075	87168	57.8	504.2	Section
GIFT-COFB-SER-TI	1	4422	87168	128.6	1121	Section

The content of this chapter was presented in 2022 at the thirteenth Internation Conference on Cryptology in Africa [48].

Outline. The chapter unfolds as follows: In Section 7.1 and Section 7.2, we present the circuits for GIFT-COFB-SER-S and GIFT-COFB-SER-F respectively in which the finite field operations are absorbed in the last encryption round of the GIFT block cipher. Section 7.3 details the circuit for the first-order threshold implementation of GIFT-COFB and experimental results for leakage detection in which we do not observe any first-order leakage. Section 7.4 analyses the hardware synthesis measurements. Lastly, this chapter is concluded in Section 7.5.

7.1 GIFT-COFB-SER-S

Recall the description of the bitsliced GIFT block cipher from Section 2.6.2 and its swap-and-rotate implementation detailed in Chapter 6 where the state bits $X = (x_0 \cdots x_{127})$ are partitioned into four lanes such that

$$S_0 = x_0 x_1 \cdots x_{30} x_{31}, \qquad S_1 = x_{32} x_{33} \cdots x_{62} x_{63},$$
$$S_2 = x_{64} x_{65} \cdots x_{94} x_{95}, \qquad S_3 = x_{96} x_{97} \cdots x_{126} x_{127}.$$

The bit permutation Π now reduces to four independent sub-permutations $\Pi_0, \Pi_1, \Pi_2, \Pi_3$ that act on each lane

$$\Pi(x_0 \cdots x_{127}) = \Pi_0(x_0 \cdots x_{31})\Pi_1(x_{32} \cdots x_{63})\Pi_2(x_{32} \cdots x_{63})\Pi_3(x_{96} \cdots x_{127}).$$

The plaintext is loaded into FF_{127} throughout cycles 0-127. In cycles 96-127, the S-box layer of the first round and the swaps that calculate Π_0 are active in cycles 96-159. Subsequently,

the swaps corresponding to Π_1, Π_2, Π_3 are active during the cycles 128-191, 160-223 and 192-255 respectively, concluding the calculation of the first round function. This pattern repeats for the remaining rounds until the 40-th and ultimate round which starts executing in cycle 5088. The first ciphertext bits are made available at FF_0 from cycle 5120 until the last bit has exited the pipeline in cycle 5248. Hence, a full encryption takes exactly $(40+1) \cdot 128 = 5248$ cycles. For a more detailed breakdown of the swap activation cycles the reader is referred to Section 6.4. A schematic timeline diagram is given in Figure 7.1.

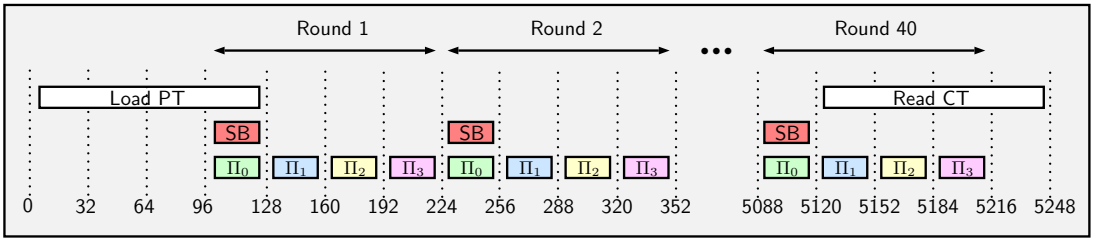


Figure 7.1: Timeline diagram of the swap-and-rotate GIFT implementation; the numbers in the x-axis denote clock cycles. Note that for better readability the permutation swaps intervals are only indicated for the first 32 cycles in this and subsequent timeline figures.

Additionally, recall another peculiarity of the bit-sliced GIFT variant is that the 4-bit S-box is not applied to adjacent bits of the state but to the first bits of each lane, in other words $x_{31}, x_{63}, x_{95}, x_{127}$. Summa summarum, the circuit for the state pipeline is compact and simple as shown in the high-level diagram of Figure 7.2.

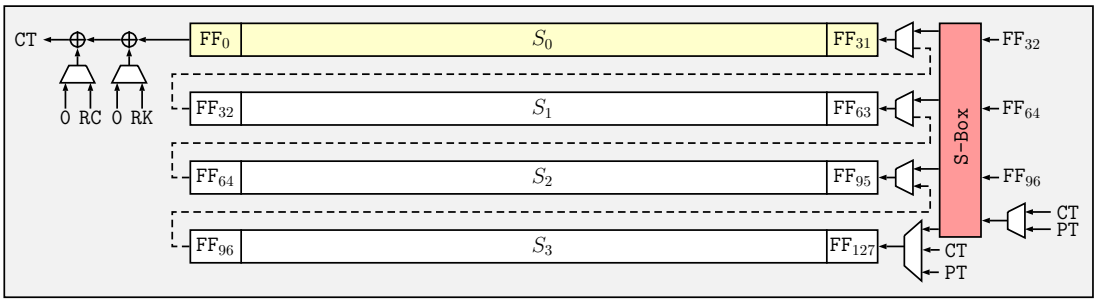
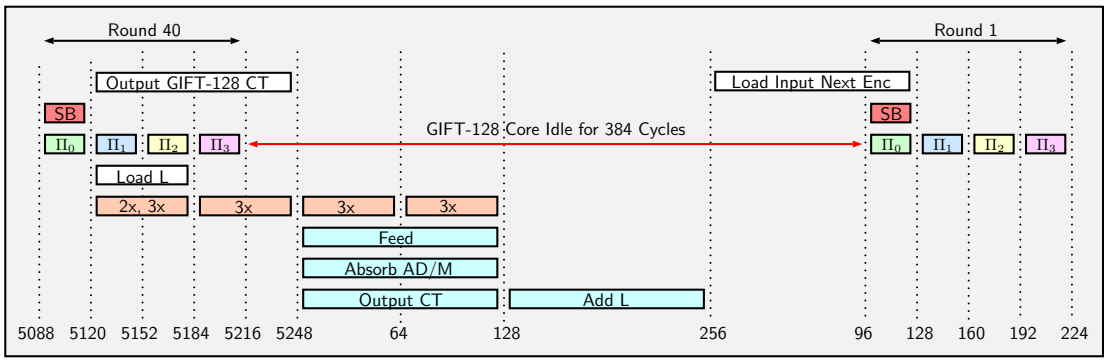


Figure 7.2: The swap-and-rotate GIFT state pipeline circuit. There are in total nine swaps over twelve flip-flops.

In this section, we lay the groundwork for our bit-serial GIFT-COFB circuits and describe how to efficiently implement the field multiplication as well as the feedback function. In the process, we integrate the obtained component circuits with the swap-and-rotate module described in previously which yields the first lightweight bit-serial GIFT-COFB circuit. This is straightforward in the sense that there is a clear separation between the execution of the GIFT encryption, the calculation of the Feedback function and the addition of L to the internal state, alongside the loading of the plaintext as part of the next encryption. Meaning that, after the ciphertext has completely exited the pipeline, these three operations are each performed in 128 separate cycles during which the GIFT pipeline executes the identity function,



Cycles	Operations
0-5087	The nonce is fed into the state pipeline bit by bit in cycles 0 to 127. Thereafter, the first 39 rounds of GIFT are executed.
5088-5247	Round 40 executes during cycles 5088-5216. The resulting ciphertext bits exit the pipeline during cycles 5120-5247. We read the first 64-bits of the ciphertext into the L register during cycles 5120-5183 while executing the first multiplication during the same period.
For each additional data block, the following cycles are executed sequentially:	
0-127	After the ciphertext has fully exited the state pipeline, we start executing the feedback function for 128 cycles. In parallel, we can absorb the input data block and, if needed, produce the ciphertext bits. Subsequent multiplications of L are performed if required.
128-255	The state after the above is now XORed with the content of the L register and the result is written back, bit by bit, into the state register.
0-5247	A new encryption starts after L has been added to the cipher state.

Figure 7.3: Timeline diagram and cycle-by-cycle description of GIFT-COFB-SER-S for two successive encryptions. Note the interval of 3×128 idle cycles between encryptions.

i.e., the state bits rotate through the shift register without the activation of any swap or the S-box. Hence, there is an overhead of 3×128 cycles between encryption invocations. We denote this circuit by $GIFT-COFB-SER-S$ which will be the basis for the latency-optimized variant, presented in the subsequent section, that circumvents those periodic 384 *penalty* cycles with only a marginal increase in circuit area.[28] The exact sequence of operations between encryptions is described in Figure 7.3.

[28] The letters S and F in GIFT-COFB-SER-S and GIFT-COFB-SER-F stand for slow and fast respectively. Similarly to the nomenclature used in Chapter 4.

7.1.1 Implementing the Feedback Function

Recall the feedback function as detailed in Section 2.6.2, in other words

$$\text{Feed}(X_0, X_1) = (X_1, X_0 \lll 1).$$

It is a bit-permutation belonging to the symmetric group over a set of 128 elements and executes two operations sequentially:

1. Swapping of the upper and lower halves of the word $(X_0, X_1) \rightarrow (X_1, X_0)$.
2. Leftward rotation of the lower half X_0 by one position.

Proposition 1. *Using two swaps over four flip-flops, it is possible to fully implement both subroutines of the subroutine Feed in exactly 128 clock cycles.*

Proof. A schematic cycle-by-cycle diagram of the feedback function is depicted in Figure 7.4. The first swap $\text{FF}_{127} \leftrightarrow \text{FF}_{126}$ is active from cycles 2 to 64. As a result, the state at clock cycle 64 is given as $x_{64}x_{65}\cdots x_{126}x_{127}x_1x_2\cdots x_0x_{63}$. Note that this is already the output of the Feedback function if the two least significant bits were swapped, which is then done in cycle 64. The second swap $\text{FF}_0 \leftrightarrow \text{FF}_{64}$ is active from cycle 64 to 127 which effectively computes the identity function over 64 cycles. One can thus see that after 128 cycles that the register contains the intended output of the Feed function. □

Absorbing Data Blocks and Outputting the Ciphertext. In order to avoid having to pass the message block bits twice to the circuit, once to produce the AEAD ciphertext and once for absorption into the state, i.e., $\text{Feed}(X) \oplus M$, this absorption is performed in parallel to the execution of the feedback function. Note that if $X = x_0x_1\cdots x_{127}$ and $M = m_0m_1\cdots m_{127}$, then the i-th bit u_i of $\text{Feed}(X) \oplus M$ is given as:

$$u_i = \begin{cases} m_{127-i} \oplus x_{191-i} & \text{if } 64 \leq i < 128, \\ m_{127-i} \oplus x_{64-i} & \text{if } 0 < i \leq 63, \\ m_{127-i} \oplus x_0 & \text{if } i = 0 \end{cases}$$

By inspection of Figure 7.4, one can see that in order to execute the above seamlessly, the data bits must be added to FF_{64}. This is because, for any i, the state bit x_{191-i} (for $64 \leq i < 128$), x_{64-i} (for $0 < i \leq 63$) and x_0 (for $i = 0$) is always present at FF_{64} at clock cycle i. Thus, to implement the above, we need one additional XOR gate before the 63rd flip-flop in the state register. Additionally, FF_0 always contains the most significant bit of $X \lll i$ at any cycle $i \in [0, 127]$, thus the ciphertext, which is computed as $M \oplus X$, is extracted by adding the input data bit with FF_0. In Figure 7.5, we present the state pipeline circuit of GIFT-COFB-SER-S that integrates the swaps of the feedback function, and the additional XOR gates for the data absorption and ciphertext creation.

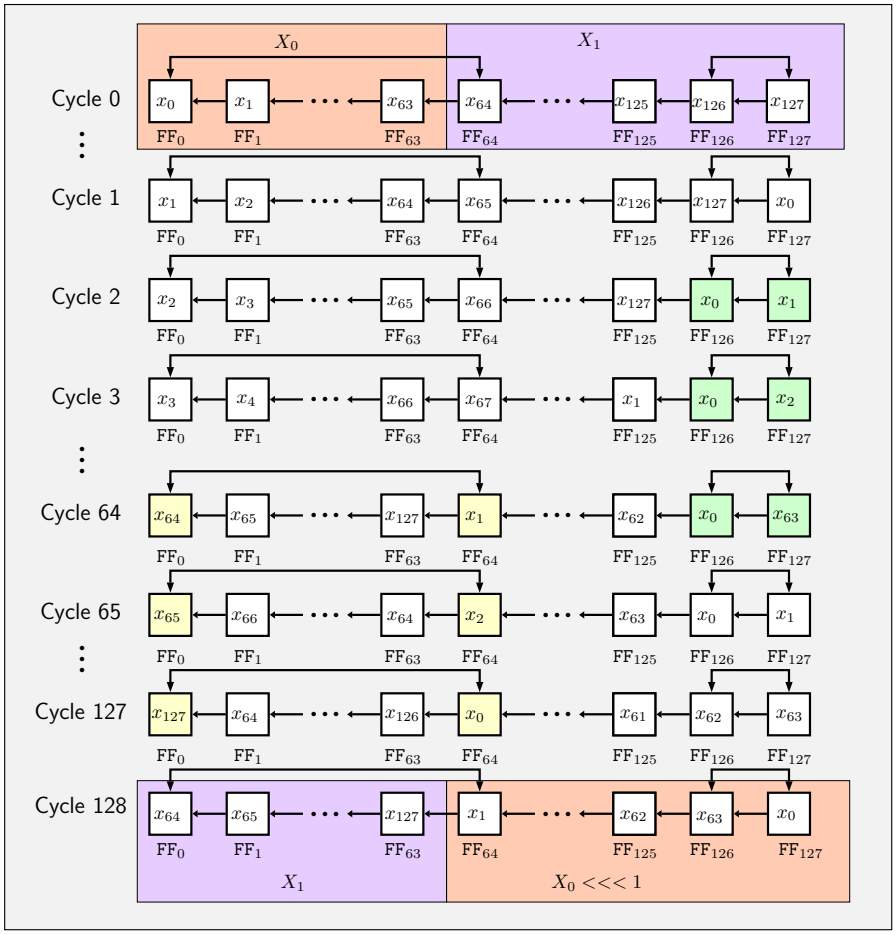


Figure 7.4: Cycle-by-cycle execution diagram of the feedback function. Green marked registers denote active swaps that execute $X_0 \lll 1$ while yellow registers mark active swaps that perform $(X_0, X_1) \rightarrow (X_1, X_0)$. Note that when a swap is active as shown by a coloured box on FF_x and FF_y, then the operation performed in the pipeline is a) swap contents of FF_x and FF_y and then b) rotate.

7.1.2 Multiplication by 2 and 3

GIFT-COFB multiplies the auxiliary state L between encryptions by either the factor 2 or 3^x for $1 \leq x \leq 4$ depending on the associated data and message block sizes and padding. If it were not for the period right after the initial encryption of the nonce N in which L has to be loaded and updated in a short time interval, this would not be too much of an issue as there is ample time to calculate the multiplication while the encryption core is busy. In the following, we demonstrate how to efficiently multiply L by 2 or 3 in 64 cycles, yielding a maximum latency of 256 clock cycles for any factor 3^4.

Let $L = l_0 l_1 \cdots l_{62} l_{63}$ be the individual bits of the register. On a 64-bit shift register, multi-

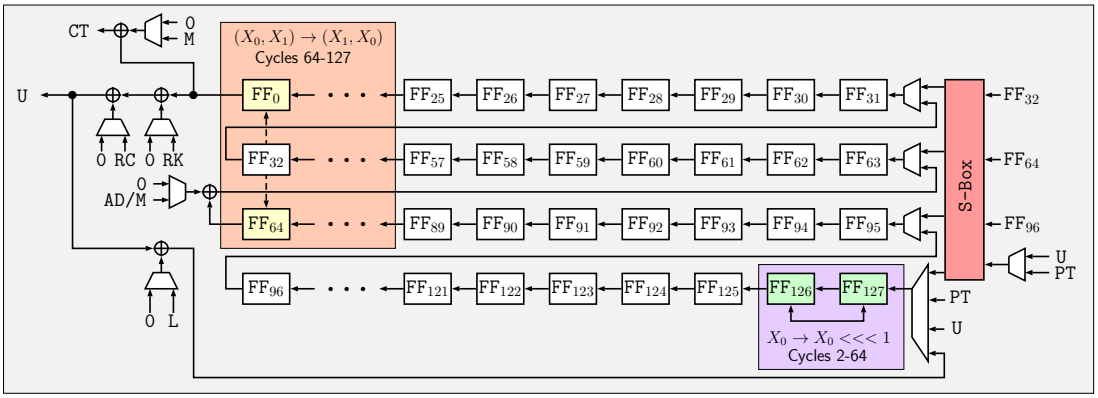


Figure 7.5: Fully integrated GIFT-COFB-SER-S state pipeline.

plication by 2 has the following form:

$$2 \times l_0 l_1 \cdots l_{63} = (L \ll 1) \oplus (l_0 * 0^{59} 11011),$$

which, in plain terms, is simply a leftward shift by one position and the addition of the most significant bit l_0 to four lower bits. On the other hand, the multiplication by three is more involved as $3 \times L = (2 \times L) \oplus L$ and is thus given as

$$3 \times l_0 l_1 \cdots l_{63} = (L \ll 1) \oplus (l_0 * 0^{59} 11011) \oplus L.$$

A single-cycle implementation of this function necessitates 64 additional 2-input XOR gates that would incur roughly 128 GE in most standard libraries, which is a considerable overhead for a bit-serial circuit. Note that technically $3\times$ can be implemented with zero additional gates, if one is prepared to pay with latency. This is because $p_{64}(x)$ is a primitive polynomial, and since the element 2 is the root of $p_{64}(x)$ it must generate the cyclic multiplicative group of the finite field. With some arithmetic, it can be deduced that $3 = 2^d$ where $d = 9686038906114705801$ in this particular representation of the finite field. The discrete logarithm d of 3 is an integer of the order of 2^{63}, hence executing the multiplication by 2 over d would, in theory, compute the multiplication by the factor 3.

Disregarding this theoretical detour, our actual goal consists in implementing, with minimal circuitry, both the multiplication by 2 and 3 in such a way that after 64 clock cycles the first bit of the updated state exits the pipeline and after the 128 cycles the entire multiplication has finished.

Proposition 2. *By equipping the L shift register with a single auxiliary d-flip-flop, three 2-input NAND gates, one 2-input XOR gate and one 2-input XNOR gate, it is possible to multiply L by either 2 or 3, i.e., by the polynomials x or* $(x+1)$.

Proof. We begin by observing the following relation: if $V = v_0 v_1 \cdots v_{63} = 2 \times l_0 l_1 \cdots l_{63}$ and

$W = w_0 w_1 \cdots w_{63} = 3 \times l_0 l_1 \cdots l_{63}$, where v_i, w_i are given as

$$v_i = \begin{cases} l_{i+1} \oplus l_0 & \text{if } i \in \{59, 60, 62\}, \\ l_0 & \text{if } i = 63, \\ l_{i+1} & \text{otherwise} \end{cases} \qquad w_i = \begin{cases} l_{i+1} \oplus l_0 \oplus l_i & \text{if } i \in \{59, 60, 62\}, \\ l_0 \oplus l_i & \text{if } i = 63, \\ l_{i+1} \oplus l_i & \text{otherwise.} \end{cases}$$

It is immediately evident that for all three cases v_i and w_i differ only by the XOR of the term l_i. In cycle 0, bit l_{63} is first stored in an auxiliary register, which we hereafter refer to as Aux. Using this fact, we show how to update the register. Then we calculate each update bit as follows, where α, β and γ are signals defined below:

$$u = (\alpha \cdot \text{Aux}) \oplus (\beta \cdot \text{FF}_0) \oplus (\gamma \cdot \text{FF}_1). \tag{7.1}$$

Identity Function. It is simply a rotation of the L register; $\alpha = \beta = \gamma = 0$.

Multiplication by 2. Signal α is used to add l_{63}, which is stored in register Aux in cycle 0, to the output bit. β is always 0 for multiplication by 2. γ is 1 for all but cycle 63 in order to implement a left shift (and not left rotate). Consequently, we have

$$\alpha = \begin{cases} 1 & \text{cycles } 59, 60, 62, 63, \\ 0 & \text{otherwise;} \end{cases} \quad \beta = 0; \quad \gamma = \begin{cases} 1 & \text{cycle} \neq 63, \\ 0 & \text{otherwise.} \end{cases} \tag{7.2}$$

If the update function u were to simply be $\gamma \cdot \text{FF}_1$, then after 64 cycles, the register would store $l_0 \cdots l_{63} \ll 1$. Now if we added l_{60} to the LFSR update in cycles 59, 60, 62, 63, then after 64 cycles, the LFSR state would be $(l_0 \cdots l_{63} \ll 1) \oplus (l_0 * 0^{59}11011)$ which is the output of the doubling function.

Multiplication by 3. α, and γ as above and always $\beta = 1$. Adding $\beta \cdot \text{FF}_{63}$ to the update function enables the output to be $(l_0 \cdots l_{63} \ll 1) \oplus (l_{60} * 0^{59}11011) \oplus (l_0 \cdots l_{63})$, which is the output of the tripling function.
Using (7.1), we can implement both multiplications by factors 2 and 3 in 64 cycles using one auxiliary d-flip-flop, three 2-input NAND gates and one 2-input XOR gates and one 2-input XNOR gate. A diagram of the resulting circuit is shown in Figure 7.6. □

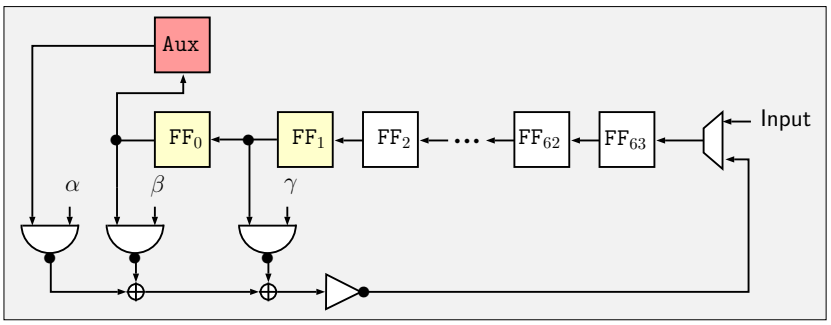


Figure 7.6: Implementation of the bit-serial multiplication by 2 and 3.

7.1.3 GIFT-COFB-SER-S Total Latency

It can be seen that the encryption of the nonce takes 5248 cycles. Thereafter, every additional block takes $256+5248 = 5504$ cycles to process. Thus if the padded associated data and message consist of B blocks in total, then the time taken to produce the ciphertext and tag is $T_S = 5248 + 5504 \cdot B$ clock cycles.

7.2 GIFT-COFB-SER-F

The proposed bit-serial circuit from the previous section already represents the to-date most area-efficient GIFT-COFB implementation. However, as our bit-serial interpretation respects the natural order of operations as given in the specification of the mode of operation, it has a significantly elevated latency. This is mainly due to the encryption core being idle during 3×128 clock cycles between successive invocations which means that if we want to do away with those penalty cycles, the calculation of the Feedback function, the update and addition of L, the addition of incoming associated data and message bits and the loading of the next encryption state all have to occur in parallel. This means that during 128 cycles while the GIFT ciphertext bits $c_i^{(j)}$ for data block j leave the pipeline, the newly entering bits $v_i^{(j+1)}$ for data block $j+1$ at FF_{127} are necessarily of the form

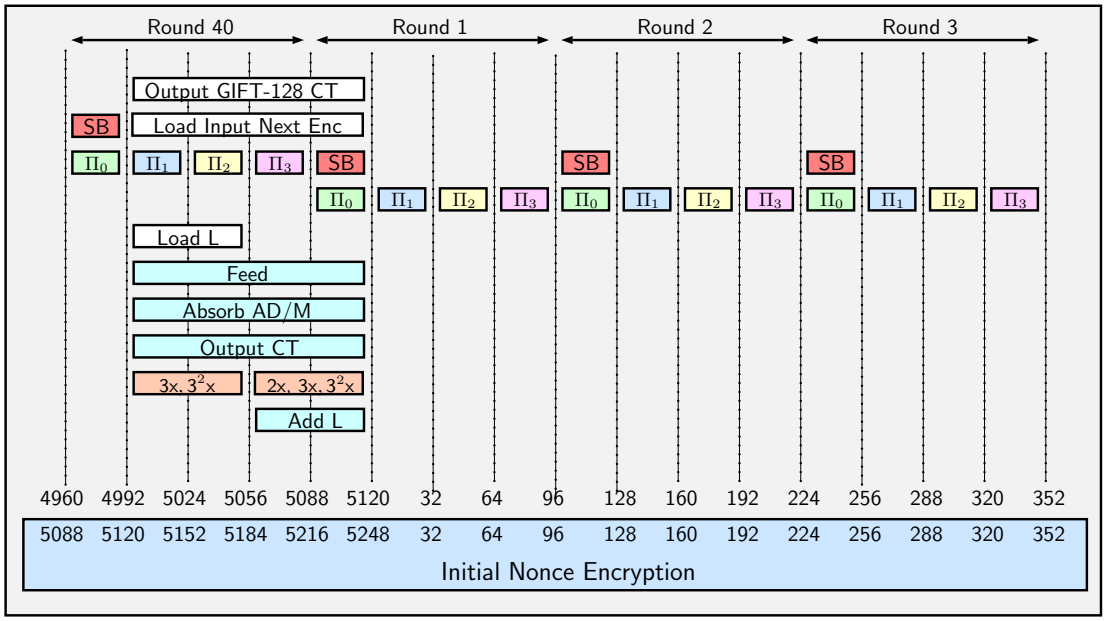
$$v_i^{(j+1)} = c_i^{(j)} \oplus \mathrm{rk}_i \oplus \mathrm{rc}_i \oplus l_i^{(j+1)} \oplus d_i^{(j+1)}, \tag{7.3}$$

where $\mathrm{rk}_i \oplus \mathrm{rc}_i$ denote the i-th bit of the last round key, $l_i^{(j+1)}$ denotes the i-th bit of the l register to be added before the $(j+1)$-th data block and $d_i^{(j+1)}$ is the i-th bit of the $(j+1)$-th data block. In this section, we describe three requisite tweaks to GIFT-COFB-SER-S that let us achieve this goal.

1. Change the swaps of Feed described in Section 7.1.1 as to enable its execution in parallel to the ciphertext bits leaving the state pipeline.
2. Reorder the incoming data bits as well as L such that they can be seamlessly added to the exiting ciphertext bits.
3. Enrich the L circuit from Section 7.1.2 with additional logic in order to compute the multiplication by the factors $2, 3, 3^2, 3^3$ and 3^4 in 128 clock cycles concurrently with the last encryption round. The updated time diagram alongside a cycle-by-cycle description is given in Figure 7.7.

7.2.1 Tweaking the Feedback Function

Note that, as explained in Section 7.1.1, the swap between FF_0 and FF_{63} during the calculation of the Feed function preserves the state over 64 cycles in GIFT-COFB-SER-S. However, the same can be achieved by swapping FF_x and FF_{x+64} for any x. Since we execute Feed concurrently with the last GIFT encryption round, we want the bit exiting the pipeline at FF_0 to be



Cycles	Operations
0-5087	The nonce is fed into the state pipeline bit by bit in cycles 0 to 127. Thereafter, the first 39 rounds of GIFT are executed.
5088-5247	Round 40 executes during cycles 5088-5216. The resulting ciphertext bits exit the pipeline during cycles 5120-5247. We read the first 64-bits of the ciphertext into the L register during cycles 5120-5183 while executing the first multiplication during the same period such that in the second 64 cycles it is added back to the cipher state. In the same 128 cycles, we also execute Feed and add the data bits.
For each additional data block, the following cycles are executed sequentially:	
0-4959	We perform the first 39 rounds of the new encryption call.
4960-4991	The first 32 cycles of the last round of the encryption call.
4992-5119	We execute the following in parallel: we finish executing the encryption call, we load the new cipher state, perform multiplication of L, execute Feed, absorb data bits and the (updated) L, output the ciphertext and start executing round 1 of the next encryption call.

Figure 7.7: Timeline diagram of GIFT-COFB-SER-F. Note that the initial 128 cycles to load the nonce cannot be parallelized with other functions, hence the initial encryption of the nonce takes 5248 cycles.

the output of the $GIFT$ encryption routine in the same order as in GIFT-COFB-SER-S. Swapping out FF_0 and FF_{63}, however, disrupts that order. Thus, we replace the swap $FF_0 \leftrightarrow FF_{63}$ with the swap $FF_{64} \leftrightarrow v_i^{(j)}$, where $v_i^{(j)}$ is the i-th bit of the j-th incoming block as defined above.

A side effect of this choice affects the S-box inputs of just the first round of every new encryption with an incoming data block. In GIFT-COFB-SER-F, in the first S-box invocation of a new encryption, inputs are now of the form FF_{32}, $v_i^{(j)}$, FF_{96}, FF_{64} instead of FF_{32}, FF_{64}, FF_{96}, $v_i^{(j)}$ due to the $FF_{64} \leftrightarrow v_i^{(j)}$ swap. As a result, we need two more multiplexers that swap FF_{64} and $v_i^{(j)}$ before entering into the S-box during cycles 96 to 127. The resulting circuit for GIFT-COFB-SER-F is depicted in Figure 7.8.

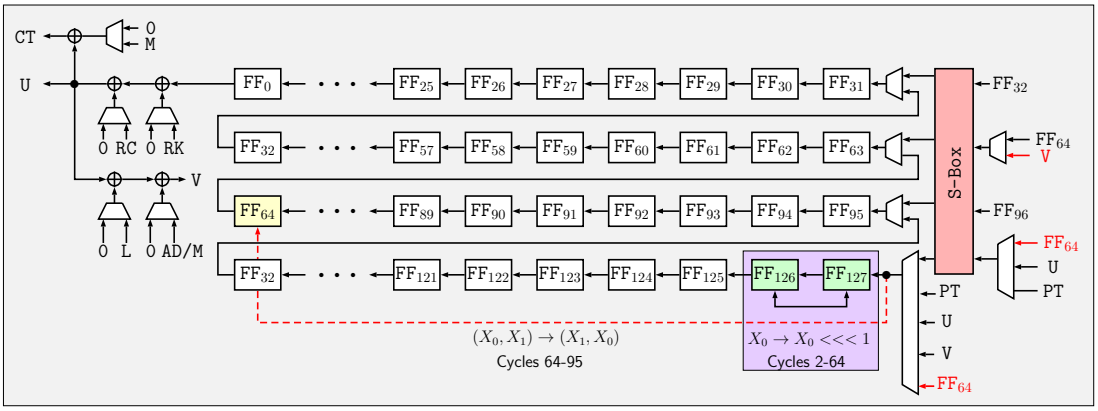


Figure 7.8: GIFT-COFB-SER-F state pipeline. U denotes the input bit during intermediate cipher rounds and V the input during the first round of a new encryption. Wires marked in red enable the concurrent execution of Feed and the S-box.

7.2.2 Reordering Data Bits

The absorption of associated data/message bits and L normally occurs after the computation of the feedback function. However, we have to do it with the last encryption round, which involves some re-ordering of data bits and L. Consider the inverse transformation of Feed:

$$\text{Feed}^{-1}(X_0, X_1) = ((X_1 \ggg 1), X_0).$$

Note that Feed is a linear function, we have since $\text{Feed}^{-1}(L, 0^{64}) = 0^{64}||L$:

$$\text{Feed}(X \oplus \text{Feed}^{-1}(D) \oplus \text{Feed}^{-1}(L, 0^{64})) = \text{Feed}(X) \oplus D \oplus L||0^{64}.$$

We need to re-order the incoming data bits and the output of the L function by the permutation Feed^{-1} before adding it to the state, thereafter performing the Feed function over the modified state $X \oplus \text{Feed}^{-1}(D) \oplus 0^{64}||L$ which thus correctly computes the input to the next encryption call. This comes with a convenient side effect:

Proposition 3. *Placing the addition of L before Feed yields 64 spare cycles that can be used to*

perform the finite field multiplications.

Proof. When we add the string $\text{Feed}^{-1}(L, 0^{64}) = 0^{64}||L$, the first 64 cycles are spent adding the zero string. These 64 cycles can be used to load L into its register and simultaneously multiply it by either $2, 3, 3^2, 3^3$ or 3^4 such that in cycle 64 the first correctly updated bits exit L and the entire register is updated in a total of 128 cycles. □

7.2.3 Enhancing the Multiplier

We proceed to demonstrate that the assertion from the previous proposition, namely that after 64 cycles the first correctly multiplied bit exits the L pipeline, can be integrated into the existing multiplier from Section 7.1.2 with modest overhead.

Proposition 4. *By equipping the L shift register with four auxiliary d-flip-flops, nine 2-input NAND gates, eight 2-input XOR gates and one 2-input XNOR gate, it is possible to multiply L by either 2, 3,* 3^2, 3^3 *or* 3^4 *in 128 cycles.*

Proof. Again let $L = l_0 l_1 \cdots l_{63}$ be the individual state, then the multiplication by 3^2 is written as

$$3^2 \times l_0 l_1 \cdots l_{63} = (L \ll 2) \oplus (l_0 * 0^{58}110110) \oplus L \oplus (l_1 * 0^{59}11011).$$

Recall the multiplication circuit for the factors 2 and 3 from Section 7.1.2. We re-introduce signals α, β, γ as α_0, β_0 and γ_0, and to capture multiplication by 3^2 we further add δ_0 and α_1. Let Aux0 be the register that stores l_0 in cycle 0, and analogously denote by Aux1 the auxiliary register that stores l_1 in the same cycle. Then, the circuit for the multiplication by the factors 2, 3, 3^2 can be written as

$$u = (\alpha_0 \cdot \text{Aux0}) \oplus (\alpha_1 \cdot \text{Aux1}) \oplus (\beta_0 \cdot \text{FF}_0) \oplus (\gamma_0 \cdot \text{FF}_1) \oplus (\delta_0 \cdot \text{FF}_2).$$

In order to compute the multiplication by the higher factors 3^3 and 3^4, we equip the L pipeline with a second 3^2 circuit at the beginning that continuously overwrites register FF_2, which is therefore a scan flip-flop, as the bits enter the pipeline. In cycle 2, the values FF_{61} and FF_{62} are l_0 and l_1 respectively, which are stored in this cycle in auxiliary flip-flops Aux2 and Aux3. The updated bit for these cases can be written as:

$$u' = (\alpha_2 \cdot \text{Aux2}) \oplus (\alpha_3 \cdot \text{Aux3}) \oplus (\beta_1 \cdot \text{FF}_{61}) \oplus (\delta_1 \cdot \text{FF}_{63}).$$

The resulting circuit full multiplier is shown in Figure 7.9.
As before in Section 7.1.2, we give an exact list of activation cycles for each control signal below.

Identity function: All signals are set to 0.

Multiplication by 2. We have $\alpha_0 = \alpha$, $\beta_0 = \beta$ and $\gamma_0 = \gamma$ where α, β, γ are as in (7.2), additionally $\delta_0 = \alpha_1 = 0$. Since only the left half of the diagram is relevant, all other signals are 0.

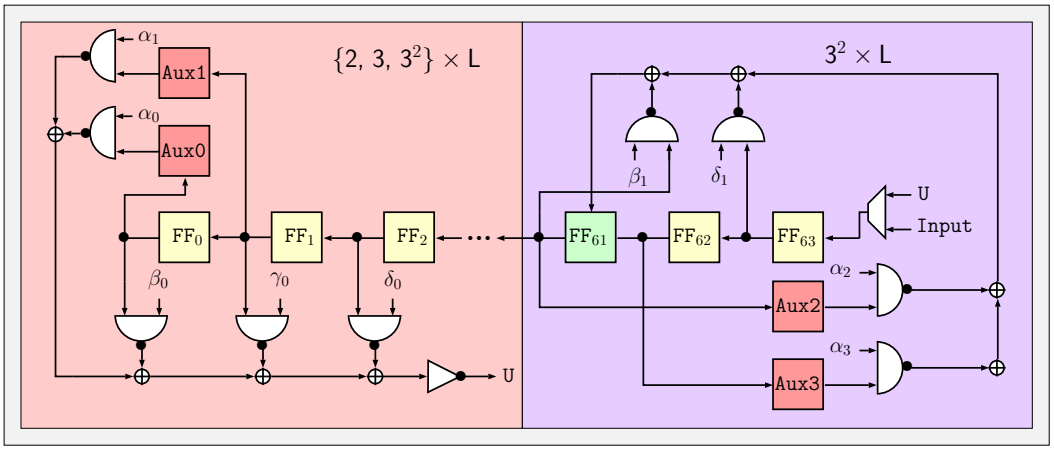


Figure 7.9: L state pipeline that performs the multiplication by the factors 2, 3, 3^2, 3^3 and 3^4.

Multiplication by 3. As in multiplication by 2 except for $\beta_0 = 1$.

Multiplication by 3^2. As above, only the left portion of the diagram is used. δ_0 steers the addition of l_2, and is active except for the last two cycles. α_0 enables the addition of l_0, similarly α_1 enables the addition of l_1. Furthermore, γ_0 is always 0 as it is only used in the multiplication by 3 and β_0 is always 1. In summary, we have

$$\alpha_0 = \begin{cases} 1 & \text{cycles } 58,59,61,62, \\ 0 & \text{otherwise;} \end{cases} \qquad \alpha_1 = \begin{cases} 1 & \text{cycles } 59,60,62,63, \\ 0 & \text{otherwise;} \end{cases}$$

$$\delta_0 = \begin{cases} 1 & \text{cycle} < 62, \\ 0 & \text{otherwise;} \end{cases} \qquad \beta_0 = 1; \quad \gamma_0 = 0.$$

Multiplication by 3^3. We first use the 3^2 multiplier on the right in the diagram that executes on newly entered bits then finish with a multiplication by 3 by the left multiplier. As we always update FF_{61} for the factor 3^2, the activation cycles of the signals α_2, α_3, β_1 and δ_1 are analogous to the signals α_0, α_1, β_0 and δ_0 in the left 3^2 multiplication module except they occur 62 cycles before.

Multiplication by 3^4. The first phase is exactly as in the case of multiplication by 3^3, and the second phase is exactly as in multiplication by 3^2.

□

7.2.4 GIFT-COFB-SER-F Total Latency.

It can be seen from Figure 7.7 that the encryption of the nonce takes 5248 cycles. Thereafter every additional block takes 5120 cycles to process. Thus if the padded AD and message consist of B blocks in total, then the time taken to produce the ciphertext and Tag is $T_F = 5248 + 5120 \cdot B$ clock cycles. We can see that for each block of data processed we save $\frac{T_S - T_F}{B} = 384$ clock cycles.

7.3 GIFT-COFB-SER-TI

The third circuit we propose re-utilises the GIFT-COFB-SER-F component from Section 7.2 and elevates it to a first-order Threshold Implementation. For a primer on the properties of Threshold Implementations, see Section 2.7.

We commence by noting that the GIFT S-box S is cubic and can be decomposed into two quadratic S-boxes S_F and S_G from $\{0,1\}^4 \rightarrow \{0,1\}^4$ such that $S = S_F \circ S_G$. Since S_F and S_G are quadratic, they can be masked using a direct sharing approach using three shares for a first-order threshold implementation such that

$$S_G = S_{G_1} \oplus S_{G_2} \oplus S_{G_3};\ S_F = S_{F_1} \oplus S_{F_2} \oplus S_{F_3},$$

where $S_{G_1}, S_{G_2}, S_{G_3}$ and $S_{F_1}, S_{F_2}, S_{F_3}$ are the component function of S_F and S_G respectively. This approach was used in [85] from which we take the proposed non-complete and uniform first-order TI. We provide the algebraic expression of the component functions in Appendix E. Consequently, our implementation uses three shares for the state and L registers while the key and round constant pipeline remain unshared. The only challenge in constructing the circuit is to place the S_{F_i} and S_{G_i} substitution boxes such that they correctly compute the masked GIFT S-box in consonance with the other operations done in parallel. This can readily be achieved by noting that we can replace the unmasked S and replace it with S_{G_i}, and place S_{F_i} after the first flip-flop of each lane, i.e., $FF_0, FF_{95}, FF_{63}, FF_{31}$, which executes for 32 cycles starting in cycle 97 of each round. A schematic of one of the three shares of the state pipeline is shown in Figure 7.10.

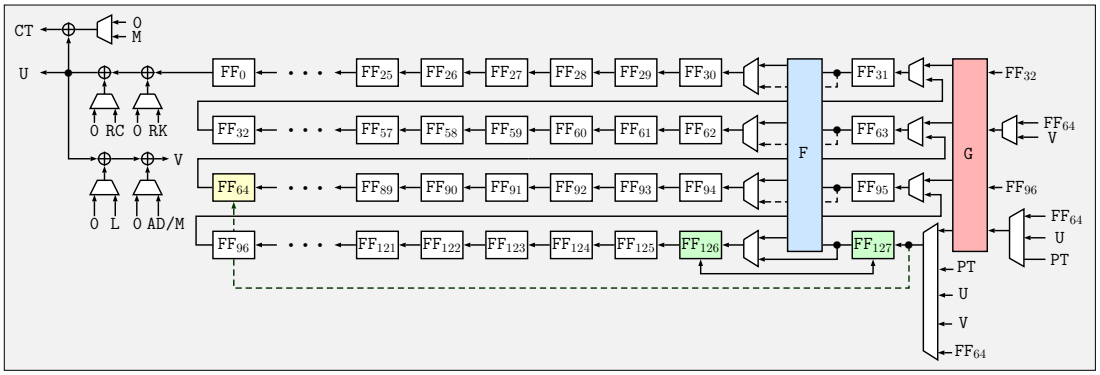


Figure 7.10: One of the three state pipeline shares of the GIFT-COFB-SER-TI circuit.

7.3.1 Leakage Evaluation

We applied the TVLA methodology [117] and performed non-specific t-tests (using Welsh's t-test) to validate the first-order security of our threshold implementation GIFT-COFB-SER-TI. Furthermore, we took a threshold of $|t| > 4.5$ for any value of t computed in order to reject the null hypothesis that GIFT-COFB-SER-TI encryption operations admit indistinguishable mean power consumption in the case that the input is either uniform or fixed.

The SAKURA-G side-channel evaluation board[29] was used that hosts two Spartan 6 FPGA cores, one which performed GIFT-COFB-SER-TI operations clocked at a slow 1.5MHz and the other that interfaces between the cryptographic core and a computer (which generates pre-masked shares for the DUT). To prevent unintended optimisations that could lead to leakage during synthesis, we added `DONT_TOUCH`, `KEEP` and `KEEP_HIERARCHY` constraints to our code. Measurements were taken with a Tektronix MSO44 at 625MS/s taking 3000 data points per trace, which corresponds to 7 cycles of S-box evaluation (the only non-linear component of GIFT-COFB) in the second round of the first GIFT encryption call, i.e., while GIFT is encrypting the nonce. During testing, we reset the cryptographic core between each GIFT-COFB encryption and interleaved encryptions with random and fixed inputs. A sample trace is shown in Figure 7.11a.

As a measure to ensure our setup was calibrated properly, we first performed a t-test in the leaky *masks off* setting, which is as follows. Recall that GIFT-COFB-SER-TI is a three-share TI. Then, in the masks off setting, one input value is set to the original input (fixed or random) and the other two to constant values (the zero vector here). We present the results in Figure 7.11b revealing that significant, potentially exploitable leakage was detected with just 20 thousand traces. Then, with masks on (i.e., with uniform masking used), we found no evidence of leakage with 10,000,000 traces, evident in Figure 7.11c.

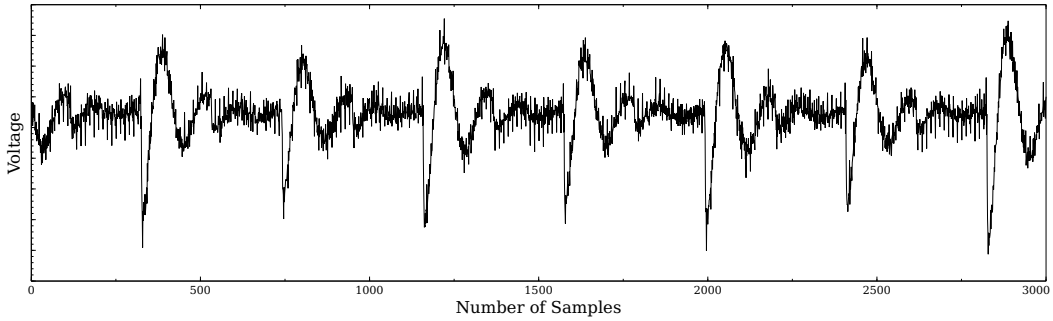
7.4 Hardware Implementation

All hardware figures were obtained through the `compile_ultra` directive was used to generate the netlists for all constructions except GIFT-COFB-SER-TI whose hierarchy is conserved via the `no_autoungroup` flag which ensures that entity boundaries are preserved preventing any security-degrading optimisation that may violate the threshold implementation properties. The obtained measurements are listed in Table 7.2. Naturally, due to the increased complexity of both Feed and the multiplier, GIFT-COFB-SER-F incurs a slightly larger circuit area than GIFT-COFB-SER-S which is offset by the latency savings as part of the parallelisation of all component functions. We note that both GIFT-COFB-SER-S and GIFT-COFB-SER-F significantly undercut Romulus, the only other lightweight block cipher scheme among the NIST LWC finalists, in both area and power/energy.

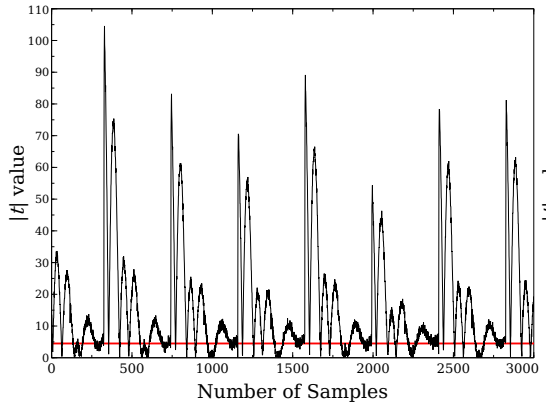
7.5 Conclusion

In this chapter, we investigated bit-serial architectures for the AEAD mode GIFT-COFB, a finalist in the NIST lightweight cryptography competition. In the process, we fill the gap left open by the omission of this scheme in Chapter 6 and propose two architectures: the first follows a natural order of operations in which the finite field operations and other state updates are performed in the time period between 2 successive calls to the encryption module. The second absorbs all these operations in the last 128 cycles of the encryption operation, and saves 384 clock cycles in the processing of every block of associated data or message. We

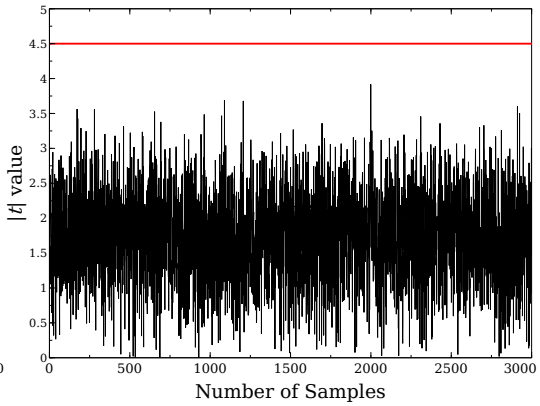
[29] `https://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html`



(a) A sample trace taken over 7 cycles.



(b) 20 thousand traces and masks off.



(c) 10 million traces and masks off.

Figure 7.11: Sample trace (top) and t-test results for the GIFT-COFB-SER-TI circuit (bottom). The red lines correspond to a threshold of $|t| = 4.5$.

then extended the second architecture to construct a first order threshold implementation of GIFT-COFB. We verify the first-order security claims by performing statistical tests on power traces resulting from an implementation of the circuit on the SAKURA-G FPGA platform.

[...] But I guess that's only half the accusation [...]

Table 7.2: Comprehensive synthesis figures for bit-serial AEAD schemes. Latency and energy correspond to the encryption of 128 bits of associated data and 1024 message bits. Highlighted schemes are NIST LWC finalists.

Scheme	Library	Area	Latency	Critical Path	Power (μW)		Energy (nJ)	
		(GE)	(Cycles)	(ns)	10 MHz	100 MHz	10 MHz	100 MHz
SUNDAE-GIFT	NanGate 15	2169	92544	0.12	15.9	108.4	147.2	100.3
	NanGate 45	1913	92544	1.28	52.4	271.7	485.1	251.5
	UMC 65	1670	92544	3.80	15.6	153.3	144.6	141.9
	TSMC 90	1574	92544	2.21	45.9	453.7	424.3	419.9
SAEAES	NanGate 15 nm	2360	24448	0.32	18.8	130.1	46.0	31.8
	NanGate 45 nm	2073	24448	3.05	61.2	329.7	149.5	80.6
	UMC 65 nm	1834	24448	10.08	21.6	212.6	52.9	52.0
	TSMC 90 nm	1752	24448	6.32	57.1	565.9	139.7	138.4
Romulus-N3	NanGate 15 nm	3311	55431	0.12	22.4	144.5	123.9	80.1
	NanGate 45 nm	2877	55431	1.25	43.0	387.8	238.3	215.0
	UMC 65 nm	2497	55431	3.79	24.7	242.3	136.9	134.3
	TSMC 90 nm	2361	55431	2.08	73.9	731.1	409.4	405.3
SKINNY-AEAD	NanGate 15 nm	6729	72960	0.19	43.9	278.3	320.6	203.0
	NanGate 45 nm	5975	72960	1.31	167.4	861.2	1221.4	628.3
	UMC 65 nm	5105	72960	4.67	45.1	452.2	329.2	329.9
	TSMC 90 nm	4806	72960	2.17	128.7	1272.6	939.0	928.5
GIFT-COFB-SER-S	NanGate 15 nm	2741	54784	0.13	14.2	85.9	77.7	47.0
	NanGate 45 nm	2307	54784	1.40	55.3	245.5	302.7	134.5
	UMC 65 nm	2020	54784	4.23	13.9	134.7	75.9	73.8
	TSMC 90 nm	1907	54784	2.18	42.8	422.1	234.4	231.2
GIFT-COFB-SER-F	NanGate 15 nm	2940	51328	0.21	20.2	138.4	103.7	71.0
	NanGate 45 nm	2365	51328	2.22	66.6	343.6	341.9	176.3
	UMC 65 nm	2067	51328	7.32	19.5	190.7	99.9	97.9
	TSMC 90 nm	2075	51328	4.14	57.8	571.7	296.9	293.4
GIFT-COFB-SER-TI	NanGate 15 nm	6114	51328	0.24	46.7	322.9	239.4	165.7
	NanGate 45 nm	5317	51328	2.22	149.0	777.4	764.8	399.0
	UMC 65 nm	4649	51328	8.20	44.1	432.6	226.5	222.0
	TSMC 90 nm	4423	51328	4.39	128.6	1272.1	660.1	652.9

8 Throughput: Rocca-S

[...] Self-destruct sequence [...]

The imminent global standardisation of 5G telecommunication networks marks an important turning point for the involved research community whose gaze should hereafter be directed beyond the 5G era. The 6Genesis project kickstarted this endeavour with a white paper in which 6G channels are projected to provide throughput rates upwards of 100 Gbps in software eclipsing their 5G counterparts by more than an order of magnitude[30] Concerning peak throughput, the paper further illuminates potential avenues that would allow for rates in the Terabit range and states:

"*6G research should look at the problem of transmitting up to 1 Tbps per user.*"

Naturally, performance only covers half of the requirements for a prospective 6G standard with the other one being security as discussed by the 3GPP standardisation organisation which examined the possible impacts of quantum computing in the coming years especially due to Grover's algorithm. The motivation is to design a cipher that provides 256-bit classical security and 128-bit quantum security. Thus apart from the 256-bit key and the 256-bit classical security, which in most cases would provide 128-bit quantum security especially with respect to Grover's algorithm, proposing an algorithm with a suitable tag length is also important. For a tag of size t, a quantum adversary could use Grover's algorithm with the verifier as an oracle to make the verifier accept a random tag with probability $2^{t/2}$. Meaning a tag of size less than 256 bits would not serve the purpose of providing 128-bit quantum security. Consequently, a 256-bit tag is essential to guarantee 128-bit quantum security for forgery attacks.

In 2021, the first algorithm that took on these design challenges was proposed under the name Rocca [116]. Rocca achieves an encryption/decryption speeds of more than 100 Gbps in both raw encryption scheme and AEAD scheme. It supports 256-bit keys and provides 256-bit and 128-bit security against key recovery and distinguishing attacks, respectively making it the first cipher dedicated for 6G environments. However, shortly after its publication Hosoyamada et al. [82] found a key-recovery attack with a complexity of 2^{128} thus invalidating the original security guarantees. Additionally, a quantum adversary can easily modify

[30] The 6G flagship project by the University of Oulu created a preliminary list of requirements: `http://jultika.oulu.fi/913files/isbn9789526223544.pdf`

this attack to a quantum attack with the application of Grover's algorithm, which reduces the complexity of a forgery attack in quantum settings to 2^{64}.

Contributions. In this chapter, our goal lies in the endeavour of designing an ideal cipher that offers both classical and quantum security. Thus apart from the 256-bit key and 256-bit classical security, which in most cases would provide 128-bit quantum security especially with respect to Grover's algorithm, proposing an algorithm with a suitable tag length is also important. To the best of our knowledge, none of the existing algorithms dedicated to 5G and beyond provide throughput rates of more than 1 Tbps, more than 100 Gbps in software, and support 256-bit tags, including AEGIS [126], Tiaoxin-346 [110] and Rocca [116] as well as 5G-oriented constructions such as ZUC-256 [120] and SNOW-V [68]. This fact motivates a search for new algorithms which meets all three of these requirements for 6G applications.

We propose Rocca-S, which is an AES-based encryption scheme with a 256-bit key and a 256-bit tag, can provide 256-bit and 128-bit security against classical and quantum adversary, respectively. Rocca-S is the first cryptographic algorithm that reaches beyond throughput rates of 2 Tbps which we establish in a meticulous simulation environment. Furthermore, in the latest software environments our design achieves encryption and decryption speeds of more than 200 Gbps. The general design of Rocca-S follows the key features of the AEGIS-family, Tiaoxin-346, and Rocca. The most challenging task lies in designing round functions that achieve 256-bit security for forgery attacks (supporting 256-bit tag) without sacrificing hardware and software performance. More specifically, the lower bounds of the number of active S-boxes should be more than twice than in AEGIS, Tiaoxin-346 and all of Jean and Nikolić structures [87], Obtaining this trade-off between security and performance is obviously a non-trivial design problem.

We take advantage of an interesting insight that while increasing the number of AESENC from 4 which was the case for Rocca to 6 in the round function, the overhead in software performance can be made negligible by reducing state size, where AESENC is a single round function of AES realized by the AES-NI instructions. This allows us to add more nonlinear operations into each round and increase the security against forgery attacks while keeping the software performance. From hardware point of view, as AESENC can be executed in parallel, increasing the number of AESENC module in round function does not cause any overhead regarding throughput. The ASIC design space is explored by proposing several constructions ranging from round-based circuits to unrolled variants and ultimately a low-area byte-serial implementation. In particular, the round-based and unrolled circuits achieve throughput rates that exceed 1 Terabit per second (even crossing the 2 Tbps barrier for some instances) and thus eclipse related AEAD circuits known in the literature by at least 50% without sacrificing other metrics such as occupied silicon area or power/energy consumption making Rocca-S a competitive choice satisfying the requirements of a wide spectrum of environments.

The contents of this chapter appear at the 28th European Symposium on Research in Computer Security (ESORICS) held in The Hague in the year 2023.

Outline. In Section 8.1, the specification of Rocca-S is introduced which is subsequently discussed in Section 8.2. Afterwards, we investigate the security guarantees of our construction

against existing attacks in Section 8.3 and highlight the implementation aspects in Section 8.4 for hardware and in Section 8.5 for software. The chapter is terminated in Section 8.6.

8.1 Specification

For the remainder of this chapter, a block denotes a 16-byte value. The constants Z_0 and Z_1 are are taken unaltered from Tiaoxin-346 and are thus of the form

$$Z_0 = \texttt{0x428a2f98d728ae227137449123ef65cd},$$
$$Z_1 = \texttt{0xb5c0fbcfec4d3b2fe9b5dba58189dbbc}.$$

S is the state of Rocca-S, which is composed of 7 blocks, i.e., $S = (S[0], S[1], \ldots, S[6])$, where $S[i]$ $(0 \le i \le 6)$ are blocks and $S[0]$ is the first block. $\mathrm{AES}(X, Y)$ is one AES round applied to the block X, where the round constant is Y, as defined below:

$$\mathrm{AES}(X, Y) = (\mathrm{MixColumns} \circ \mathrm{ShiftRows} \circ \mathrm{SubBytes}(X)) \oplus Y,$$

where MixColumns, ShiftRows and SubBytes are the routines of of the AES round function. $A(X)$ is the AES round function without the constant addition operation as:

$$A(X) = \mathrm{MixColumns} \circ \mathrm{ShiftRows} \circ \mathrm{SubBytes}(X),$$

8.1.1 Round Function

The input of the round function $R(S, X_0, X_1)$ of Rocca-S consists of the state S and two blocks (X_0, X_1) such that

$$\begin{aligned} S[0] &= S[6] \oplus S[1], & S[1] &= \mathrm{AES}(S[0], X_0), \\ S[2] &= \mathrm{AES}(S[1], S[0]), & S[3] &= \mathrm{AES}(S[2], S[6]), \\ S[4] &= \mathrm{AES}(S[3], X_1), & S[5] &= \mathrm{AES}(S[4], S[3]), \\ S[6] &= \mathrm{AES}(S[5], S[4]). \end{aligned}$$

A schematic depiction of the Rocca-S round function is shown in Figure 8.1.

Semantically, Rocca-S is an authenticated-encryption with associated-data scheme composed of four phases: initialisation, processing the associated data, encryption and finalisation. The input consists of a 256-bit key $K_0||K_1 \in \mathbb{F}_2^{128} \times \mathbb{F}_2^{128}$, a 128-bit nonce N, the associated data AD and the message M, where $X||Y$: The concatenation of X and Y. The output is the corresponding ciphertext C and a 256-bit tag T. $|X|$ is the length of X in bits. Define $\overline{X} = X||0^l$ where 0^l is a zero string of length l bits, and l is the minimal non-negative integer such that $|\overline{X}|$ is a multiple of 256. In addition, write X as $X = X_0||X_1||\ldots||X_{\frac{|X|}{256}-1}$ with $|X_i| = 256$. Further, X_i is written as $X_i = X_i^0||X_i^1$ with $|X_i^0| = |X_i^1| = 128$.

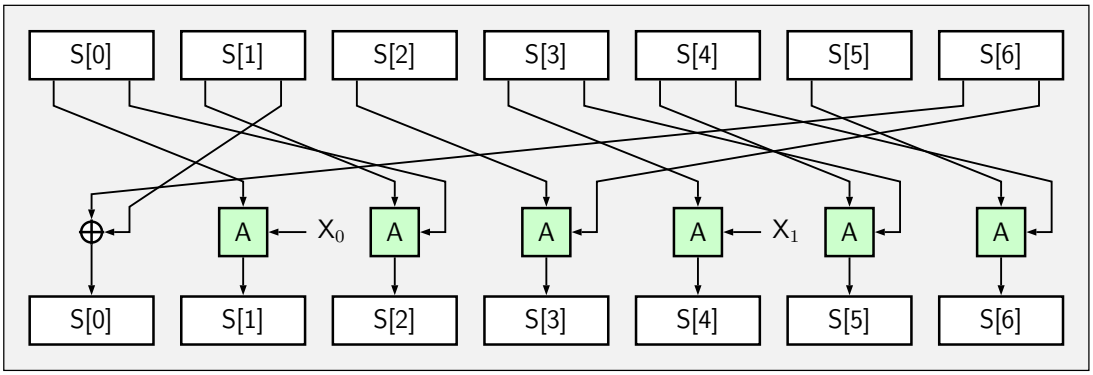


Figure 8.1: Schematic of the Rocca-S round function.

Initialisation. First, (N, K_0, K_1) is loaded into the state S in the following way:

$$\begin{aligned} &S[0] = K_1, \quad S[1] = N, \quad S[2] = Z_0, \\ &S[3] = K_0, \quad S[4] = Z_1, \quad S[5] = N \oplus K_1, \\ &S[6] = 0 \end{aligned}$$

Here, two 128-bit constants Z_0 and Z_1 are encoded as 16-byte little endian words and loaded into $S[2]$ and $S[3]$ respectively. Then, 16 iterations of the round function $R(S, Z_0, Z_1)$ is applied to the state S. After 16 iterations of the round function, two 128-bit keys are XORed with the state S such that

$$\begin{aligned} &S[0] = S[0] \oplus K_0, \quad S[1] = S[1] \oplus K_0, \quad S[2] = S[2] \oplus K_1, \\ &S[3] = S[3] \oplus K_0, \quad S[4] = S[4] \oplus K_0, \quad S[5] = S[5] \oplus K_1, \\ &S[6] = S[6] \oplus K_1. \end{aligned}$$

Processing the Associated Data. If AD is empty, this phase will be skipped. Otherwise, AD is padded to $\overline{AD}$ and the state is updated as $R(S, \overline{AD}_i^0, \overline{AD}_i^1)$ for $i \in \{0, \ldots, d-1\}$, where $d = \frac{|\overline{AD}|}{256}$.

Encryption. The encryption phase is similar to the phase to process the associated data. If M is empty, the encryption phase will be skipped. Otherwise, M is first padded to $\overline{M}$ and then $\overline{M}$ will be absorbed with the round function. During this procedure, the ciphertext C is generated. If the last block of M is incomplete and its length is b bits, i.e. $0 < b < 256$, the last block of C will be truncated to the first b bits. A detailed description is shown below:

$$\begin{aligned} C_i^0 &= \mathrm{AES}(S[3] \oplus S[5], S[0]) \oplus \overline{M}_i^0, \\ C_i^1 &= \mathrm{AES}(S[4] \oplus S[6], S[2]) \oplus \overline{M}_i^1, \\ S &= R(S, \overline{M}_i^0, \overline{M}_i^1), \end{aligned}$$

where $i \in \{0, \ldots, m-1\}$ and $m = \frac{|\overline{M}|}{256}$.

Finalisation. After the above three phases, two 128-bit keys K_0 and K_1 are first XORed with the state S

$$S[1] = S[1] \oplus K_0,$$
$$S[2] = S[2] \oplus K_1.$$

Then, the state S will again pass through 16 iterations of the round function $R(S, |AD|, |M|)$ and then the 256-bit tag is computed

$$T = \bigoplus_{i=0}^{3} S[i] || \bigoplus_{i=4}^{6} S[i].$$

The length of associated data and message is encoded as 16-byte little endian word and stored into $|AD|$ and $|M|$, respectively. A formal description of Rocca-S is described in Algorithm 6 and the corresponding illustration is shown in Figure 8.2.

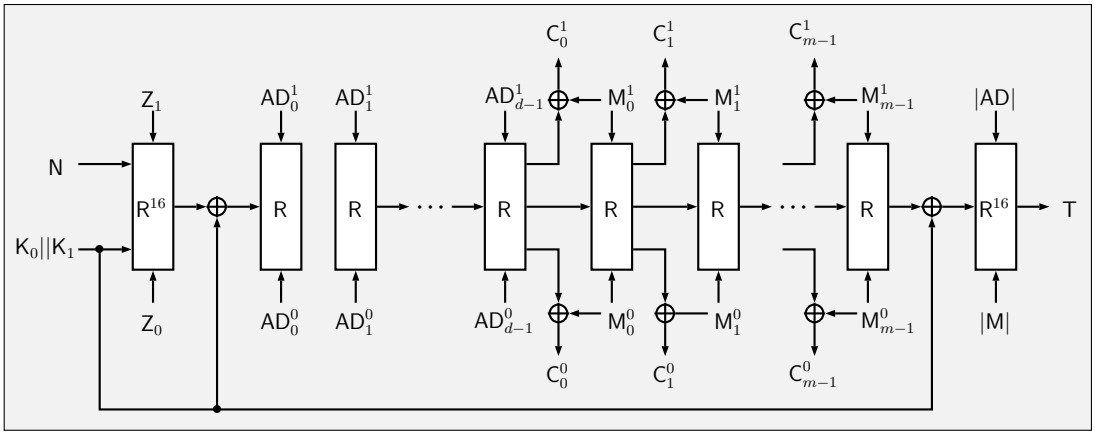


Figure 8.2: Schematic overview of the Rocca-S authenticated encryption with associated data scheme.

A Raw Encryption Scheme. If the phases of processing the associated data and finalisation are removed, a raw encryption scheme is obtained.

A Keystream Generation Scheme. If the phases of processing the associated data and finalisation are removed and there is no message injection into round function such that $R(S, 0, 0)$, a keystream generation scheme is obtained.

8.1.2 Security Claims

Rocca-S is designed to provide 256-bit security against key-recovery and forgery attacks in the nonce-respecting setting. We do not claim its security in the related-key and known-key settings. The message length for a fixed key is limited to at most 2^{128} and we also limit the number of different messages that are produced for a fixed key to be at most 2^{128}. The length of associated data of a fixed key is up to 2^{64}. Secondly, Rocca-S provides 128-bit security

Algorithm 6 Encryption and Decryption Functions of Rocca-S.

function Rocca-S(K_0, K_1, N, AD, M)
  $S \leftarrow \mathsf{Initialise}(N, K_0, K_1)$
  if $|AD| > 0$ **then**
    $S \leftarrow \mathsf{ProcessAD}(S, \overline{AD})$
  if $|M| > 0$ **then**
    $S, C \leftarrow \mathsf{Encrypt}(S, \overline{M})$
    $\mathsf{Truncate}(C)$
  $T \leftarrow \mathsf{Finalise}(S, |AD|, |M|)$
  return (C, T)
function Rocca-S$^{-1}(K_0, K_1, N, AD, C, T)$
  $S \leftarrow \mathsf{Initialise}(N, K_0, K_1)$
  if $|AD| > 0$ **then**
    $S \leftarrow \mathsf{ProcessAD}(S, \overline{AD})$
  if $|C| > 0$ **then**
    $S, M \leftarrow \mathsf{Encrypt}(S, \overline{C})$
    $\mathsf{Truncate}(M)$
  if $T = \mathsf{Finalise}(S, |AD|, |M|)$ **then**
    return M
  else
    return $\perp$
function Initialise(N, K_0, K_1)
  $S[0], S[1], S[2], S[3] \leftarrow K_1, N, Z_0, Z_1$
  $S[4], S[5], S[6] \leftarrow Z_1, N \oplus K_1, 0$
  for $i = 0$ to 15 **do**
    $S \leftarrow R(S, Z_0, Z_1)$
  $S[0], S[1] \leftarrow S[0] \oplus K_0, S[1] \oplus K_0$
  $S[2], S[3] \leftarrow S[2] \oplus K_1, S[3] \oplus K_0$
  $S[4], S[5] \leftarrow S[4] \oplus K_0, S[1] \oplus K_1$
  $S[6] \leftarrow S[6] \oplus K_1$
  return S

function ProcesssAD(S, AD)
  $d \leftarrow |AD|/256$
  for $i = 0$ to $d - 1$ **do**
    $S \leftarrow R(S, AD_i^0, AD_i^1)$
  return S
function Encrypt(S, M)
  $m \leftarrow |M|/256$
  for $i = 0$ to $m - 1$ **do**
    $C_i^0 \leftarrow \mathsf{AES}(S[3] \oplus S[5], S[0])$
    $C_i^1 \leftarrow \mathsf{AES}(S[4] \oplus S[6], S[2])$
    $S \leftarrow R(S, M_i^0, M_i^1)$
  return (S, C)
function Decrypt(S, C)
  $c \leftarrow |C|/256$
  for $i = 0$ to $c - 1$ **do**
    $M_i^0 \leftarrow \mathsf{AES}(S[3] \oplus S[5], S[0])$
    $M_i^1 \leftarrow \mathsf{AES}(S[4] \oplus S[6], S[2])$
    $S \leftarrow R(S, M_i^0, M_i^1)$
  return (S, M)
function Finalise$(S, |AD|, |M|, K_0, K_1)$
  $S[1], S[2] \leftarrow (S[1] \oplus K_0, S[2] \oplus K_1)$
  for $i = 0$ to 15 **do**
    $S \leftarrow R(S, |AD|, |M|)$
  $T_0, T_1 \leftarrow 0, 0$
  for $i = 0$ to 3 **do**
    $T_0 \leftarrow T_0 \oplus S[i]$
  for $i = 4$ to 6 **do**
    $T_1 \leftarrow T_1 \oplus S[i]$
  return $T_0 || T_1$

against quantum adversary with respect to key-recovery and forgery attacks in the Q1 and Q2 settings. Rocca-S does not provide security against related-key and known-key superposition attacks as is the case of all known symmetric ciphers.

We remark that the probability that an adversary will be able to recover a 256-bit key or forge a 256-bit tag in time q (with very little data) is generally $\frac{q}{2^{256}}$ in the classical framework and $\frac{q^2}{2^{256}}$ in the quantum framework. However, these bounds do not consider some additional constant/logarithmic factors, so it would be wise to claim a slightly lower security bound. We would like to point here that in consideration of these reasons the authors of Saturnin [53] had claimed a slightly lower security level – 224/112-bit security in classical/quantum settings. Following it, we believe that if full round Rocca-S is used, the security will be at least 2^{224} and 2^{112} in classical and quantum settings respectively, even if taking all these additional constant/logarithmic factors into account.

8.2 Design Rationale

The general design of Rocca-S follows the key features of Rocca [116], i.e., SIMD-friendly round function and efficient permutation-based structure to achieve the high throughput in software and hardware.

Permutation-Based Structure. Rocca-S is based on permutation-based authenticated encryption schemes using AES round functions such as the AEGIS family [126], Tiaoxin-346 [110] and Rocca [116]. To further increase the resistance against attacks on AEGIS and Tiaoxin-346 [106] while maintaining a high level of performance, we carefully design the nonce and the key loading scheme of the initial state and the output function.

High-Throughput Round Function. To achieve more than 100 Gbps in software, we leverage the power of SIMD instructions, e.g., XOR and AND and the AES-NI (AES New Instructions), equipped on general modern CPUs. In our design, we utilize only AESENC as one of the AES-NI instructions, which executes one round of AES with an input state S and a 128-bit round key K:

$$\mathrm{AESENC}(S,K) = (\mathrm{MixColumns} \circ \mathrm{ShiftRows} \circ \mathrm{SubBytes}(S)) \oplus K.$$

In order to maximise the performance in software and minimize critical path of round functions in hardware, we lay our focus on a class of round functions with following features as with Rocca [116].

- Applying only either AESENC or XOR to each block in one round.
- Applying a state-wise permutation before operations of AESENC or XOR.

In hardware, as there is no delay in a state-wise permutation in hardware, the critical path of this round function is a single AESENC module. This delay is smallest one for AES-based round functions.

8.2.1 Differences to Rocca

Rocca [116] is an AES-based high-throughput authenticated encryption scheme with a 256-bit key and 128-bit tag whose security claims were invalidated soon after its appearance in a key-recovery attack by Hosoyamada et al. [82] that could be mounted with a time and data complexity of 2^{128} by leveraging both an encryption and decryption oracle. The attack is based on the premise of obtaining a nonce-repeated valid plaintext-ciphertext pair by exhaustively querying the decryption oracle. Once such a pair is obtained, the procedure continues in the nonce-misuse setting with an ordinary differential attack to recover the cipher state which is akin to a key-recovery attack as the initialisation function of Rocca is invertible. More precisely, the procedure is initiated with an encryption of some nonce and message of specific length (N, M) yielding a ciphertext and a corresponding tag (C, T). The attacker then iteratively queries the decryption oracle with the tuple $(N, C \oplus \Delta, T')$ for all possible tags T' and an injected ciphertext difference. A nonce-repeated valid plaintext (N, M') is returned with probability 2^{-128}. In the design of Rocca-S, we follow a two-pronged approach in order to counter this pitfall:

1. *Key Feed-Forward.* A simple strategy of avoiding a key-recovery after a potential state exposure consists in adding back the encryption key into the state after the initialisation rounds which prevents any inversion attempts of the initialisation function. A similar strategy is utilised in AEGIS. When compared to the otherwise heavy functions in Rocca-S, this is a low-overhead countermeasure in both software and hardware as shown in Section 8.1.1.

2. *256-bit Tag.* The attack in [82] owes its complexity of 2^{128} to the size of the tag. Hence, a 256-bit tag would require 2^{256} decryption queries until nonce-repeated a plaintext-ciphertext pair is found. However, the authors also remarked that the increase of the tag size is a more nuanced affair due to the existence of forgery attacks in which a collision in the internal state is induced through injected associated data or plaintext differentials. Such differentials occur with probability 2^{-150} in Rocca thus a 256-bit tag would not suffice for 256-bit forgery security. In Rocca-S, we design an entirely new round function whose differential properties guarantee 256-bit state-recovery security (in combination with the key feed-forward operation) and forgery security. A detailed analysis of the Rocca-S security properties are given in Section 8.3. Note that having a 256-bit tag has the added property of offering 128-bit forgery security against quantum computers as explained in Section 8.3.5.

8.2.2 Performance-Security Trade-Off

As in hardware the multiple AESENC modules can be deployed in parallel without sacrificing the critical path of the design, the design problem of finding a sensible round function reduces to explore efficient configurations in software. In other words, we need to find optimal parameters of our round function candidates such as the state size and the number of inserted messages as well as the number of AES-NI invocations.

The performance of AES-NI can be evaluated through latency and throughput, where latency and throughput are the number of clock cycles required to execute one AES-NI instruction and call the same instructions consecutively in the parallel execution, respectively. To keep things simple, we focus on the latest architectures beyond the Intel Ice-Lake series CPU where the latency and throughput figures of AES-NI are 3 and 0.5, respectively. This means that a single AES-NI call is completed in 3 clock cycles and a new invocation can be scheduled 0.5 cycles after the previous one. Jean and Nikolić introduced the Rate metric to estimate the approximate speed of the round function [87], and smaller Rate leads to a more efficient round function.

Definition 7 (Rate [87])**.** *Rate is the required number of AESENC calls to encrypt a 128-bit message, which is defined as*

$$\text{Rate} = \#\text{AESENC} \,/\, \#\text{Message}.$$

They further discussed the number of AESENC calls in each round to fully take advantage

of parallel execution, which is expressed by the following equation:

$$\#\text{AESENC} \geq \text{Latency} \,/\, \text{Throughput}.$$

If #AESENC is less than Latency/throughput, there exist empty cycles in a parallel process. To fully take advantage of the parallel processing, which should be avoided. In the case of our target architectures, we have

$$\#\text{AESENC} \geq \text{Latency}/\text{Throughtput} = 3/0.5 = 6.$$

Note that since our output function utilizes two AESENC modules to be secure against linear attacks [106], #AESENC calls in the round function should be at least 4. Furthermore, for security against forgery attacks, we estimate the lower bound for the number of differentially active S-boxes by a Mixed Integer Linear Programming (MILP) solver [108]. Since the maximum differential probability of the AES S-box is 2^{-6}, the lower bound for the number of active S-boxes should be larger than 44, as it gives $2^{-6\times 43} < 2^{-256}$ as an estimate of the differential probability.

Taking these issues into consideration, we clarify requirements for the round function of Rocca-S as follows:

1. Rate = #AESENC/#Message) is as small as possible.
2. #AESENC is at least 4 in the round function.
3. The state size is as small as possible.
4. 256-bit security against forgery attacks, i.e., the lower bound for the number of active S-boxes is ≥ 43.

We automatised the search for the optimal round function and ultimately settled on the variation specified in Section 8.1 with attributes tabulated in Table 8.1. A detailed explanation of the work that went into finding the round function of Rocca-S is given in Appendix D.1.

Table 8.1: Rocca-S round function attributes. Note that in comparison to Rocca our design has a smaller state and more #AESENC calls but as we will see later with similar performance.

#State	#AESENC	#Message	Rate	#Active S-Boxes	Full Diffusion Rounds
7	6	2	3.0	46	5

8.2.3 Loading Scheme and Output Function

For the loading scheme of the nonce and key, we mainly want to avoid the case occurring in Tiaoxin-346. Specifically, we expect that after some number of rounds, the whole state words cannot be expressed only in terms of $A(N)$ and (K_0, K_1). If this happens, there will be a useless round and it may open a door for more powerful attacks. By setting $S[5] = N \oplus K_1$, this pitfall is avoided easily. For the output functions, to resist the linear attack that has been successfully

applied to AEGIS [106], we use the MILP model described in [67] to search for secure ones. For both efficiency and security, we choose the output functions of the following form:

$$C_i^0 = \mathrm{AES}(S[j_0] \oplus S[j_1], S[j_2]) \oplus \overline{M}_i^0,$$
$$C_i^1 = \mathrm{AES}(S[j_3] \oplus S[j_4], S[j_5]) \oplus \overline{M}_i^1,$$

where $j_{u_1} \neq j_{u_2}$ for $u_1 \neq u_2$ and $0 \leq j_0, j_1, j_2, j_3, j_4, j_5 \leq 6$. Then, with the truncated MILP model [67], for each choice of $(j_0, j_1, j_2, j_3, j_4, j_5)$, we can compute the lower bound of the number of active S-boxes for a exploitable linear trail that can be used for attacks. For our choice, the lower bound of the number of active S-boxes is 45. Hence, the time complexity of the linear attack will be higher than $2^{45\times 6} = 2^{270}$. Note that there is a big gap between the truncated model and the bit-wise model and the actual linear trail that can be used for attacks may be of much lower bias and the time complexity may be much higher than 2^{270}.

8.3 Security Evaluation

Having reiterated the rationale that went into the design of Rocca-S, let us explore the security implications in this section by exploring its resistance against known attacks from the literature.

8.3.1 Differential Attack

To evaluate the security against differential attacks, we compute the lower bound for the number of active S-boxes in the initialisation phase by a MILP-aided method [108]. We evaluate it in both the single-key setting where differences can only be injected into the nonce and the related-key setting where differences can be injected into the key and nonce. Table 8.2 lists the lower bounds for the number of active S-boxes in the single-key setting and related-key setting in the initialisation phase, respectively. Since the maximal differential probability of the S-box of AES is 2^{-6}, it is sufficient to guarantee the security against differential attacks if there are 43 active S-boxes, as it gives $2^{(-6\times 43)} < 2^{-256}$ as an estimate of the differential probability. In Table 8.2, there are 68 active S-boxes over 5 rounds in the single-key setting and 53 active S-boxes over 8 rounds in the related-key setting in the initialisation phase.

Table 8.2: The lower bound for the number of active S-boxes in the initialisation phase where AS_{sk} and AS_{rk} mean an active S-box in the single-key setting and in the related-key setting, respectively.

Rounds	1	2	3	4	5	6	7	8	9	10	11	12
$\#AS_{sk}$	2	7	22	40	68	94	113	122	134	152	159	159
$\#AS_{rk}$	1	6	13	13	30	36	36	53	63	79	99	109

8.3.2 Forgery Attack

It has been shown in [110] that the forgery attack is a main threat to the constructions like Tiaoxin-346 and AEGIS as only a single round update is used to absorb each block of associated data and message. Such a concern has been taken into account in our design phase, as reported in Sect 8.2. Specifically, in the forgery attack, the aim is to find a differential trail where the attackers can arbitrarily choose differences at the associated data and expect that such a choice of difference can lead to a collision in the internal state after several number of rounds. The resistance against this attack vector can be efficiently evaluated with an automatic method [108]. As Rocca-S is based on the AES round function, it suffices to prove that the number of active S-boxes in such a trail is larger than 43 as the length of the tag is 256 bits. With the MILP-based method, it is found that the lower bound is 46. Consequently, Rocca-S can provide 256-bit security against the forgery attack.

8.3.3 State-Recovery Attack

At the keystream phase, with the knowledge of plaintexts and ciphertexts, it is possible to recover the internal state with some guess-and-determine (GnD) strategies. In this part, we discuss the resistance against this attack. To recover the whole internal state, we at least need to consider 4 consecutive rounds at the keystream phase. Specifically, we need to solve the following nonlinear equation system in terms of $S[i]$ $(0 \le i \le 6)$ where α_j $(0 \le j \le 7)$ are known values:

$$
\begin{aligned}
\alpha_0 &= A(S[3] \oplus S[5]) \oplus S[0],\\
\alpha_1 &= A(S[4] \oplus S[6]) \oplus S[2],\\
\alpha_2 &= A(\underline{A(S[2]) \oplus S[6]} \oplus \underline{A(S[4]) \oplus S[3]}) \oplus \underline{S[1] \oplus S[6]},\\
\alpha_3 &= A(\underline{A(S[3])} \oplus \underline{A(S[5]) \oplus S[4]}) \oplus \underline{A(S[1]) \oplus S[0]},\\
\alpha_4 &= A(\underline{A(A(S[1]) \oplus S[0]) \oplus A(S[5]) \oplus S[4]} \oplus \underline{A(A(S[3])) \oplus A(S[2]) \oplus S[6]})\\
&\quad \oplus \underline{A(S[0]) \oplus A(S[5]) \oplus S[4]},\\
\alpha_5 &= A(\underline{A(A(S[2]) \oplus S[6])} \oplus \underline{A(A(S[4]) \oplus S[2]) \oplus A(S[3]))} \oplus \underline{A(A(S[0])) \oplus S[1] \oplus S[6]},\\
\alpha_6 &= A(\underline{A(A(A(S[0])) \oplus S[1] \oplus S[6]) \oplus A(A(S[4]) \oplus S[3]) \oplus A(S[3])}\\
&\quad \oplus \underline{A(A(A(S[2]) \oplus S[6])) \oplus A(A(S[1]) \oplus S[0]) \oplus A(S[5]) \oplus S[4])}\\
&\quad \oplus \underline{A(s[1] \oplus S[6]) \oplus A(A(S[4]) \oplus S[3]) \oplus A(S[3])},\\
\alpha_7 &= A(\underline{A(A(A(S[1]) \oplus S[0]) \oplus A(S[5]) \oplus S[4])}\\
&\quad \oplus \underline{A(A(A(S[3])) \oplus A(S[1]) \oplus S[0]) \oplus A(A(S[2]) \oplus S[6]))}\\
&\quad \oplus \underline{A(A(S[1] \oplus S[6])) \oplus A(S[0]) \oplus A(S[5]) \oplus S[4]}.
\end{aligned}
$$

It can be found that for (α_2, α_3), 2 rounds of AES are involved. For (α_4, α_5), 3 rounds of AES are involved. While for (α_6, α_7), 4 rounds of AES are involved. Indeed, for the state-recovery attack on Rocca discussed in [116], the attacker also needs to consider 4 consecutive rounds and similar 8 equations in 8 variables. However, for all those 8 equations, at most 2 rounds

of AES are involved and Rocca still has a strong resistance against this attack. This implies that recovering the state of Rocca-S becomes much more difficult. As 2 rounds of AES can achieve the full diffusion, it soon implies the GnD attack is not a threat and Rocca-S has a strong resistance against this type of state-recovery attack.

8.3.4 Key-Committing Security

Key-committing security guarantees that a ciphertext C can only be decrypted under the same key used to produce C from some plaintext [3]. It is important for applications such as key rotation in key management services and envelope encryption solutions as they becomes more resistant to partitioning oracle attacks [98]. A key-committing AEAD scheme makes it far more difficult to find multiple keys that are valid for a particular authentication tag. Our MILP-aided method demonstrates that there are at least 147 active S-boxes in the initialisation phase when only key has a difference. Even in the worst case where the adversary fully exploits the degree of freedom of the whole state (896 bits), there are still more than 35 (= 147 - 112) uncontrollable active S-boxes remaining. Besides, the adversary has to control these in the finalisation round. Thus, we believe that it is computationally difficult to find key collision with less than 2^{128} time complexity, which is equivalent to that of generic collision attacks for 256-bit output.

8.3.5 Quantum Security

A quantum adversary can perform an exhaustive key search using Grover's algorithm [73], given a few plaintext-ciphertext pairs. This requires $2^{256/2} = 2^{128}$ iterations, where each iteration requires some basic quantum operations and an evaluation of the quantum implementation of the Rocca-S. Without going into too much detail, we can safely assume that the quantum implementation of Rocca-S would require more quantum gates than that required for AES and in this sense implementing Grover's attack to retrieve the secret key would cost more than the cost for retrieving the secret key of AES-256.

There are two models which are most generally considered for a quantum adversary to analyse the quantum (in)security of a symmetric key algorithm. These models are defined e.g., in [72, 88] as the Q1 model and the Q2 model:

- *Q1.* A symmetric cipher is considered secure in the Q1 model if there exists no quantum algorithm which can distinguish the cipher from a random permutation (or function) when the algorithm is allowed *only classical* online queries.
- *Q2.* A symmetric cipher is considered secure in the Q2 model if there exists no quantum algorithm which can distinguish the cipher from a random permutation (or function) when the algorithm is allowed quantum online queries.

The Q1 model is considered to be more realistic and are more relevant to the present communication technology, as it only requires classical online queries. The Grover's exhaustive search algorithm [73] runs in this model as it requires only few classical online plaintext-ciphertext pairs. The quantum collision search algorithms also run in this model, since hash

functions are key-less and the construction is public, so a quantum adversary can implement a hash function on an offline quantum computer. The Q2 model is a powerful model yet it is simple and easy to define. The quantum adversary in this model is allowed quantum superposition queries to the secret key oracle. This model also ensures security in all other reasonable intermediate scenarios, in which Q1 model cannot be employed, such as classical machines with quantum modules. Our quantum security claims for Rocca-S are all in the Q2 model.

Although the round function of Rocca-S makes use of the AES round function the quantum attacks proposed on reduced round AES do not apply to Rocca-S. Thus, to the best of our understanding, the quantum DS-MITM and quantum Square attacks on reduced-round AES described in [39] will not be applicable. Following [88], if there existed a classical distinguisher, differential or linear, with probability 2^{-p}, a quantum adversary could use this to mount a distinguishing attack in the $Q2$ model with time and data complexity $2^{p/2}$. It has been shown in the previous sections that the differential and linear distinguishers for Rocca-S have a probability $p > 256$, thus a quantum distinguishing attack would have a time and data complexity of at least 2^{128}. We could not find an attack using Simon's algorithm. Consequently, we claim that with respect to all known quantum attacks on symmetric ciphers, Rocca-S as an encryption scheme offers at least 128-bit security.

Furthermore, if t is the tag length in bits, $t = 256$ for Rocca-S, a classical adversary succeeds in making the verifier accept a random tag in time 2^t. A quantum adversary can however use Grover's algorithm with the verifier as the oracle and make the verifier accept a random tag in time $2^{t/2}$. Thus for Rocca-S as an AEAD Scheme, a quantum adversary can forge a tag in time 2^{128}. A quantum adversary could also try to break the scheme as a PRF by outputing a collision but this would be very costly. The optimal quantum collision algorithm proposed in [41] has a time complexity of $\mathcal{O}(2^{n/3})$, would thus not be effective against Rocca-S since the state size is 896 and the effective complexity would be $> 2^{298}$.

8.4 Hardware Implementation

The design of Rocca-S lends itself well to hardware implementations as, apart from the state registers and the AES modules of the round and encryption functions, little additional circuitry is required. In this section, we commence by investigating three separate round-based implementations of the Rocca-S specification, each of them aiming for a different hardware metric trade-off, and compare them to related AES-based AEAD constructions that also feature a key size of 256 bits. As is usually the case with hardware implementations of cryptographic primitives the focus lies in the exploration of the circuit area, throughput, latency and power/energy consumption; for all of which we demonstrate the competitiveness of Rocca-S within all those disciplines. Our approach follows a similar structure to what was established in [44] in which the authors performed an extensive analysis of different aspects when it comes to implementing SNOW-V stream cipher as a hardware circuit. In particular, the authors investigated several micro-architectural directions to implement the AES round function components.

- *S-Box.* The substitution table can be synthesized in a straightforward fashion by providing the look-up table specification LUT to the circuit compiler and letting the tool choose the actual implementation in terms of logic gates. This choice usually leads to an inefficient circuit in both area, latency and power. The Decode-Switch-Encode DSE architecture mitigates the power overhead of the S-box look-up table by encoding and decoding the inputs and outputs to the look-up table in order to reduce the switching activity of each wire. It was shown in [18] that the DSE design choice leads to the most power and thus energy-efficient implementation of the AES S-box. The combinatorial optimisation space of the S-box was explored in a work by Maximov and Ekdahl [103] in which the currently smallest description of the S-box in terms of logic gates was proposed alongside a low-depth variant and a trade-off alternative between the former two. In the remainder, we will denote these three implementations by S, F and T.

- *MixColumns.* We can similarly distinguish several ways to implement the linear layer. The currently smallest circuit comprised of 92 two-input XOR gates is due to Maximov [101]. In a separate work, Li et al. [99] demonstrated a low-depth implementation consisting of 103 XOR gates. In practice, the choice of the MixColumns has inconsequential effects on the overall metrics of the surrounding design, a fact already investigated in [44], hence for the sake of conciseness we will limit ourselves to the low-depth circuit of [99].

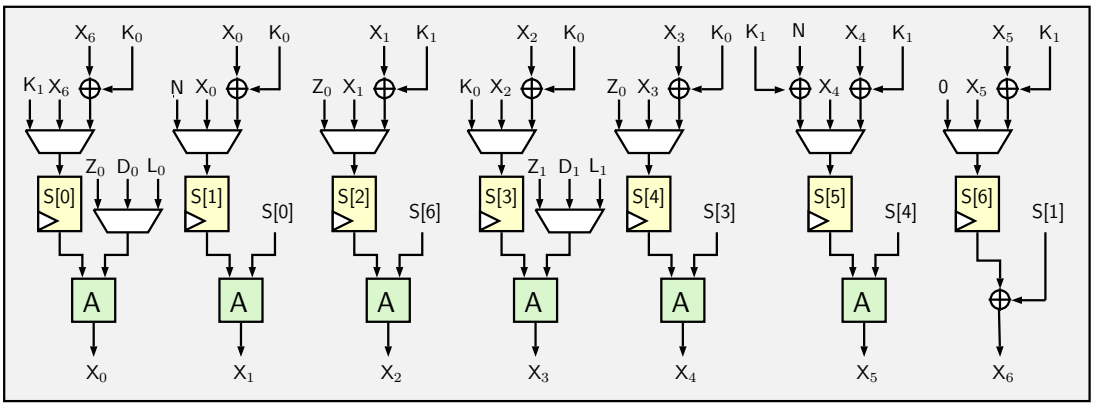
- *T-Table.* A popular way of combining the S-box, ShiftRows and MixColumns into a single procedure is the T-table approach which encodes these functions into four look-up tables which then allows to compute the an entire round function in only sixteen look-ups and a some auxiliary XOR operations. This is particularly efficient in software implementations but can also be emulated in hardware similarly to the approach of synthesizing the S-box look-up table mentioned beforehand. Henceforth, the T-table configuration will be denoted by the abbreviation TT.

8.4.1 Round-Based Circuits

A round-based implementation of Rocca-S computes one invocation of the round update function R in one clock cycle, hence sixteen cycles are required to execute both the initialisation and finalisation routines and in the same vein, the circuit absorbs 256-bit data blocks and outputs 256-bit ciphertext blocks per clock cycle. The approach we follow for the round-based implementation is relatively elementary and can be deduced from the original schematic in Figure 8.1. Six AES modules, whose plaintext inputs are directly fed from the state registers, are placed in parallel. Their computed outputs are wired back to the corresponding register inputs thus taking care of the permutation without additional circuitry. A diagram of the round-based Rocca-S architecture is depicted in Figure 8.3.

Unrolled Round Function. The round update function of Rocca-S can easily be replicated and chained together in order to compute multiple invocation in a single clock cycle. Although the area increase quickly reaches prohibitive regions the length of the critical usually rises

Figure 8.3: Round-based Rocca-S circuit. All wires without exception have a width of 128 bits. For the sake of simplicity, the control unit, the ciphertext generation component and the tag generation module have been omitted from the figure. Note that feature-rich cell-libraries often provide a dedicated flip-flop type, called a scan-flip-flop, that integrates a 2-way multiplexed input signal into the gate thus saving some gate area compared to externally multiplexing the input signal before it enters the flip-flop.



at slower pace thus yielding designs that admit the highest throughput. Note that this was already observed in [43] and by the authors of [44] where the 2-round unrolled variant of their round-based SNOW-V stream cipher circuit demonstrated the highest throughput.

Partial Round Function. As the six AES modules of the round update function significantly inflate the required circuit area, it is appropriate to investigate alternative, more area-efficient designs. One potential angle consists in computing the round function in two steps, i.e., over two clock cycles, by only using three instead of six AES components, effectively reusing the same module for two computations. In the first clock cycle the cipher state is updated as follows:

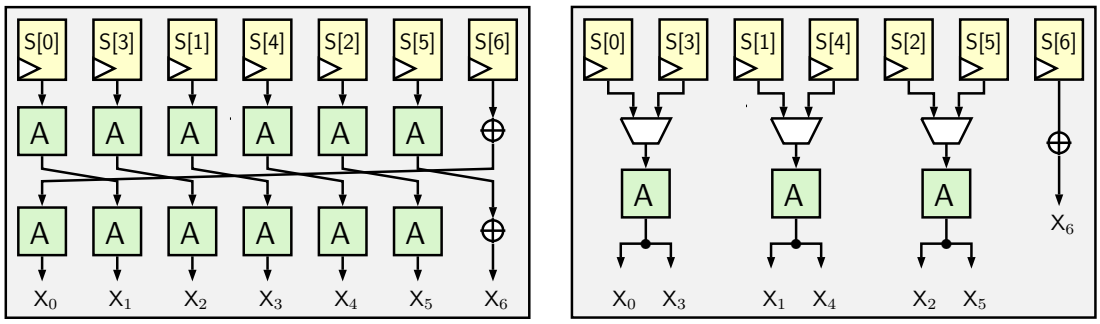
$$
\begin{aligned}
S'[0] &= S[6] \oplus S[1], & S'[1] &= \mathrm{AES}(S[0], X_0),\\
S'[2] &= \mathrm{AES}(S[1], S[0]), & S'[3] &= \mathrm{AES}(S[2], S[6]),\\
S'[4] &= S[3], & S'[5] &= S[4],\\
S'[6] &= S[5].
\end{aligned}
$$

Subsequently, the remaining three executions are performed in a similar fashion. Note that the permutation of the state has already been performed in the first step.

$$
\begin{aligned}
S''[0] &= S'[0], & S''[1] &= S'[1],\\
S''[2] &= S'[2], & S''[3] &= S'[3],\\
S''[4] &= \mathrm{AES}(S'[4], X_1), & S''[5] &= \mathrm{AES}(S'[5], S'[4]),\\
S''[6] &= \mathrm{AES}(S'[6], S'[5]).
\end{aligned}
$$

A schematic diagram of both the unrolled and partial round functions is shown in Fig-

ure 8.4. For the sake of brevity, the synthesis results for the partial round functions are tabulated in Appendix D.3.



(a) Twice Unrolled Round Function

(b) Partial Round Function

Figure 8.4: Illustration of the unrolled (a) and partial round function (b) of Rocca-S. Note only the plaintext data path of the round function is shown without control logic and encryption function.

8.4.2 Synthesis Results

Our round-based Rocca-S hardware implementations are juxtaposed against other AEAD schemes with a key size of 256 bits, namely AEGIS-256, AES-256-GCM and SNOW-V-GCM [68, 104, 126]. Note that actual published ASIC implementations of said algorithms are hard to come by, hence we chose to devise them for this comparison section. AEGIS-256 is reminiscent of Rocca-S in design and thus can be adapted accordingly, on the other hand, AES-256-GCM and SNOW-V-GCM require a Galois field multiplication module over 128 bits for which we opted for a straightforward Karatsuba architecture which is then attached to a AES-256 module extracted and extended from the Rocca-S round function and a SNOW-V stream cipher core whose implementation is available in [44].

Circuit Area. The lion's share of gate area in Rocca-S is due to the eight AES round function cores that compose its round function and ciphertext generation function. This induces a significant overhead in comparison to the other schemes. AEGIS requires only six cores whereas AES-256-GCM and SNOW-V-GCM are equipped with only and two core respectively. Note that the Galois field multiplication module, a notoriously difficult function to map to hardware, found in the latter two has an area footprint of roughly 30000 GE across cell libraries and thus constitutes a sizeable percentage of their overall silicon area. Across all implementation choices the silicon area of our round-based Rocca-S circuit remains competitive. A detailed circuit area comparison chart is given in Table 8.3.

Throughput. The premise of Rocca-S is a high-speed construction that improves on other known schemes in terms of throughput i.e., how many bits per second can be processed. In hardware, this figure is inextricably tied to the length of the critical path which specifies the maximum clock frequency at which a design can be run. In both Rocca-S and AEGIS-256 the critical path is due to the AES modules, thus it is highly variable regarding the choice of

Table 8.3: Circuit area comparison of the investigated AEAD scheme for two cell libraries and several round function implementations. Measurements for the NanGate 45 nm and UMC 65 nm cell libraries are given in Appendix D.2.

(a) Rocca-S

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 15 nm						
μm^2	22901	22999	11254	12241	11222	28648
GE	116481	116979	57241	62261	57078	145711
TSMC 90 nm						
μm^2	265446	297684	153001	169439	151556	365124
GE	94050	105472	54210	60034	53698	129366
2-Round Unrolled						
NanGate 15 nm						
μm^2	43048	43402	19884	21847	19671	54660
GE	218953	220754	101135	111120	100052	278015
TSMC 90 nm						
μm^2	496511	561725	272354	305220	269455	696585
GE	175918	199024	96497	108142	95470	246806

(b) AEGIS

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 15 nm						
μm^2	17404	17521	8703	9439	8622	21743
GE	88521	89116	44266	48009	43854	110591
TSMC 90 nm						
μm^2	199018	223580	115069	127397	113985	274184
GE	70514	79216	40770	45138	40386	97146
2-Round Unrolled						
NanGate 15 nm						
μm^2	32899	33158	15520	16992	15360	41602
GE	167333	168650	78939	86426	78125	211599
TSMC 90 nm						
μm^2	377208	426252	209225	233842	207057	527453
GE	133648	151025	74130	82852	73362	186881

(c) AES-256-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 15 nm						
μm^2	10023	10102	8265	8418	8248	12599
GE	50980	51381	42038	42816	41951	64082
TSMC 90 nm						
μm^2	131801	136861	114254	116822	114028	163112
GE	46698	48491	40481	41391	40401	57792
2-Round Unrolled						
NanGate 15 nm						
μm^2	19279	19328	15653	15960	15620	24328
GE	98058	98307	79615	81177	79447	123739
TSMC 90 nm						
μm^2	254830	265005	219790	224927	219339	317100
GE	90288	93893	77873	79694	77714	112351

(d) SNOW-V-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 15 nm						
μm^2	14889	14925	11986	12230	11959	16333
GE	75729	75912	60964	62205	60827	83074
TSMC 90 nm						
μm^2	184937	193517	157345	161454	156984	210355
GE	65525	68565	55749	57205	55621	74531
2-Round Unrolled						
NanGate 15 nm						
μm^2	26766	26861	20981	21472	20928	29675
GE	136139	136622	106715	109212	106445	150935
TSMC 90 nm						
μm^2	340672	357719	285375	293594	284653	391396
GE	120703	126743	101111	104023	100855	138675

round function implementation, whereas in AES-256-GCM and SNOW-V-GCM it is imposed by the field multiplication thus constant across implementation choices. This means that for both AES-256-GCM and SNOW-V-GCM unrolling the round function exerts only marginal effects on the overall throughput. Excluding the initialisation and finalisation phases, Rocca-S processes 256 bits of data with each clock cycle. Similarly AEGIS-256 and SNOW-V-GCM are able to process one 128-bit data block in one clock cycle wheres AES-256-GCM requires a full AES-256 encryption for each 128-bit plaintext block hence asymptotically for large plaintexts AES-256-GCM only processes 8 bits per clock cycle as one encryption necessitates 15 clock cycles for the round functions and an additional cycle to load the new plaintext of a subsequently encryption. Consequently, the ability to accept larger data blocks paired with a competitive critical path allows Rocca-S to reach a throughput well beyond 1 Terabit per second for the NanGate 15 nm cell library regardless of the choice of round function implementation, outperforming the other schemes by at least 50%. Furthermore, a throughput

rate beyond 2 Terabits per second is reached for some 2-round unrolled circuits, marking Rocca-S as first cryptographic algorithm that crosses this barrier. A throughput comparison chart is tabulated in Table 8.4.

Table 8.4: Throughput comparison of the investigated AEAD scheme for two cell libraries and several round function implementations. Note that the T-table approach of implementing the round function offers the overall best choice for both Rocca-S and AEGIS. This phenomenon was already observed in [44]. Measurements for the NanGate 45 nm and UMC 65 nm cell libraries are given in Appendix D.2

(a) Rocca-S

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 15 nm						
Critical Path (ns)	0.179	0.177	0.232	0.207	0.207	0.168
Max TP (Tbps)	1.430	1.446	1.103	1.237	1.237	1.524
TSMC 90 nm						
Critical Path (ns)	2.87	2.81	3.76	3.40	3.62	2.81
Max TP (Tbps)	0.089	0.091	0.068	0.075	0.071	0.091
2-Round Unrolled						
NanGate 15 nm						
Critical Path (ns)	0.322	0.322	0.413	0.384	0.389	0.254
Max TP (Tbps)	1.591	1.59	1.240	1.333	1.316	2.016
TSMC 90 nm						
Critical Path (ns)	5.21	4.72	7.05	6.26	6.79	3.95
Max TP (Tbps)	0.098	0.108	0.073	0.082	0.075	0.130

(b) AEGIS

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 15 nm						
Critical Path (ns)	0.167	0.165	0.21	0.196	0.198	0.132
Max TP (Tbps)	0.766	0.776	0.610	0.653	0.646	0.970
TSMC 90 nm						
Critical Path (ns)	2.66	2.48	3.50	3.12	3.36	2.48
Max TP (Tbps)	0.048	0.052	0.037	0.041	0.038	0.052
2-Round Unrolled						
NanGate 15 nm						
Critical Path (ns)	0.321	0.318	0.409	0.381	0.384	0.249
Max TP (Tbps)	0.798	0.805	0.626	0.672	0.667	1.028
TSMC 90 nm						
Critical Path (ns)	5.15	4.60	6.93	6.15	6.67	3.95
Max TP (Tbps)	0.050	0.056	0.037	0.042	0.038	0.065

(c) AES-256-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 15 nm						
Critical Path (ns)	0.349	0.349	0.349	0.349	0.349	0.349
Max TP (Tbps)	0.023	0.023	0.023	0.023	0.023	0.023
TSMC 90 nm						
Critical Path (ns)	4.89	4.89	4.89	4.89	4.89	4.89
Max TP (Tbps)	0.0016	0.0016	0.0016	0.0016	0.0016	0.0016
2-Round Unrolled						
NanGate 15 nm						
Critical Path (ns)	0.673	0.673	0.674	0.674	0.674	0.674
Max TP (Tbps)	0.024	0.024	0.024	0.024	0.024	0.024
TSMC 90 nm						
Critical Path (ns)	9.96	9.96	9.96	9.96	9.96	9.96
Max TP (Tbps)	0.0016	0.0016	0.0016	0.0016	0.0016	0.0016

(d) SNOW-V-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 15 nm						
Critical Path (ns)	0.351	0.351	0.351	0.351	0.351	0.351
Max TP (Tbps)	0.365	0.365	0.365	0.365	0.365	0.365
TSMC 90 nm						
Critical Path (ns)	6.22	6.22	6.22	6.22	6.22	6.22
Max TP (Tbps)	0.021	0.021	0.021	0.021	0.021	0.021
2-Round Unrolled						
NanGate 15 nm						
Critical Path (ns)	0.579	0.577	0.579	0.577	0.579	0.579
Max TP (Tbps)	0.442	0.444	0.442	0.444	0.442	0.442
TSMC 90 nm						
Critical Path (ns)	10.15	10.15	10.15	10.15	10.15	10.15
Max TP (Tbps)	0.025	0.025	0.025	0.025	0.025	0.025

Power/Energy Consumption. Capturing the consumptive behaviour of a cryptographic circuit is a more intricate endeavor. The energy consumption describes the sum total of electrical work performed by the circuit in a given time interval with the power consumption being its rate. To allow for a more complete picture of the energy consumption, our experiments were conducted on two workloads. A short workload describes the processing of 1024 bits of associated data and 2048 bits of plaintext whereas a long workload consists of 1024 bits of associated data and 1.28 Megabits of plaintext. Again, the round-based Rocca-S circuit stands

as competitive choice regarding its power and energy consumption. A list of all obtained power/energy measurements is given in Table 8.5.

Table 8.5: Power/energy consumption comparison of the investigated AEAD scheme for two cell libraries and several round function implementations. All figures were obtained by clocking the designs at constant frequency of 10 MHz. Measurements for the NanGate 45 nm and UMC 65 nm cell libraries are given in Appendix D.2

(a) Rocca-S

	LUT	DSE	S	F	T	TT
Round-Based						
Lat. Short (Cycles)	44	44	44	44	44	44
Lat. Long (Cycles)	5036	5036	5036	5036	5036	5036
NanGate 15 nm						
Power (mW)	1.404	0.767	1.259	1.314	1.135	0.882
Energy Short (nJ)	6.180	3.374	5.548	5.781	4.990	3.881
Energy Long (nJ)	707.1	386.3	634.01	661.7	571.6	444.2
TSMC 90 nm						
Power (mW)	1.948	0.874	1.906	1.833	1.751	0.761
Energy Short (nJ)	8.571	3.846	8.386	8.065	7.704	3.348
Energy Long (nJ)	981.0	440.1	959.9	923.1	881.8	383.2
2-Round Unrolled						
Lat. Short (Cycles)	22	22	22	22	22	22
Lat. Long (Cycles)	2518	2518	2518	2518	2518	2518
NanGate 15 nm						
Power (mW)	6.307	2.183	5.701	5.894	5.310	2.221
Energy Short (nJ)	13.88	4.80	12.54	12.97	11.68	4.89
Energy Long (nJ)	1588	549.7	1435	1484	1337	559.2
TSMC 90 nm						
Power (mW)	9.518	2.997	8.865	7.956	7.956	2.298
Energy Short (nJ)	20.94	6.593	19.50	17.50	17.50	5.056
Energy Long (nJ)	2396	754.6	2232	2003	2003	578.6

(b) AEGIS

	LUT	DSE	S	F	T	TT
Round-Based						
Lat. Short (Cycles)	48	48	48	48	48	48
Lat. Long (Cycles)	10032	10032	10032	10032	10032	10032
NanGate 15 nm						
Power (mW)	1.106	0.613	1.014	1.081	0.954	0.691
Energy Short (nJ)	5.309	2.945	4.868	5.184	4.579	3.317
Energy Long (nJ)	1109	615.6	1017	1083	956.9	693.3
TSMC 90 nm						
Power (mW)	0.600	0.388	0.451	0.477	0.442	0.406
Energy Short (nJ)	2.880	1.862	2.165	2.290	2.122	1.949
Energy Long (nJ)	601.9	389.2	452.4	478.5	443.4	407.3
2-Round Unrolled						
Lat. Short (Cycles)	24	24	24	24	24	24
Lat. Long (Cycles)	5016	5016	5016	5016	5016	5016
NanGate 15 nm						
Power (mW)	4.147	1.674	3.779	3.887	3.539	1.670
Energy Short (nJ)	9.953	4.018	9.069	9.328	8.494	4.008
Energy Long (nJ)	2080	839.7	1895	1949	1775	837.8
TSMC 90 nm						
Power (mW)	6.047	2.094	5.576	5.413	5.046	1.634
Energy Short (nJ)	14.51	5.026	13.38	12.99	12.11	3.922
Energy Long (nJ)	3033	1050	2796	2715	2531	819.6

(c) AES-256-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
Lat. Short (Cycles)	266	266	266	266	266	266
Lat. Long (Cycles)	160010	160010	160010	160010	160010	160010
NanGate 15 nm						
Power (mW)	0.521	0.417	0.502	0.515	0.490	0.577
Energy Short (nJ)	13.85	11.09	13.36	13.69	13.04	15.33
Energy Long (nJ)	8328	6674	8035	8235	7843	9224
TSMC 90 nm						
Power (mW)	0.765	0.589	0.761	0.741	0.738	0.754
Energy Short (nJ)	20.35	15.67	20.24	19.71	19.63	20.06
Energy Long (nJ)	12240	9424	12176	11856	11808	12064
2-Round Unrolled						
Lat. Short (Cycles)	133	133	133	133	133	133
Lat. Long (Cycles)	80005	80005	80005	80005	80005	80005
NanGate 15 nm						
Power (mW)	1.356	1.145	1.313	1.339	1.287	1.461
Energy Short (nJ)	18.03	15.23	17.46	17.80	17.11	19.43
Energy Long (nJ)	10845	9159	10505	10710	10293	11689
TSMC 90 nm						
Power (mW)	1.948	1.592	1.936	1.895	1.891	1.922
Energy Short (nJ)	25.91	21.17	25.75	25.20	25.15	25.56
Energy Long (nJ)	15585	12736	15489	15160	15128	15377

(d) SNOW-V-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
Lat. Short (Cycles)	42	42	42	42	42	42
Lat. Long (Cycles)	10026	10026	10026	10026	10026	10026
NanGate 15 nm						
Power (mW)	0.726	0.602	0.689	0.704	0.676	0.634
Energy Short (nJ)	3.047	2.528	2.895	2.958	2.840	2.662
Energy Long (nJ)	727.4	603.5	691.2	706.0	678.1	635.3
TSMC 90 nm						
Power (mW)	0.968	0.766	0.949	0.937	0.923	0.750
Energy Short (nJ)	4.066	3.217	3.986	3.935	3.877	3.150
Energy Long (nJ)	970.5	767.9	951.5	939.4	925.4	751.9
2-Round Unrolled						
Lat. Short (Cycles)	21	21	21	21	21	21
Lat. Long (Cycles)	5013	5013	5013	5013	5013	5013
NanGate 15 nm						
Power (mW)	1.947	1.520	1.862	1.902	1.820	1.551
Energy Short (nJ)	4.089	3.192	3.910	3.994	3.822	3.257
Energy Long (nJ)	976.1	761.7	933.6	953.6	912.6	777.6
TSMC 90 nm						
Power (mW)	2.632	1.948	2.584	2.550	2.490	1.854
Energy Short (nJ)	5.527	4.091	5.426	5.355	5.229	3.893
Energy Long (nJ)	1319	976.5	1295	1278	1248	929.4

8.4.3 Byte-Serial Circuit

Round-based circuits of Rocca-S excel in terms of throughput at the expense of occupied silicon area. A generic technique in reducing this overhead lies in restricting the data path to smaller widths which effectively serializes the design by trading gate area for latency. In [44], the authors proposed a byte-serial architecture for the SNOW-V stream cipher based on a modified byte Atomic-AES round function [17] which is the currently smallest known byte-serial AES implementation. The modification allows the computation of one round function in sixteen clock cycles meaning that at the beginning of the sixteenth cycle the first correctly computed byte exits the state pipeline.

Naturally, this Atomic-AES design also finds use in our byte-serial Rocca-S construction in which there are six Atomic-AES pipelines for the six AES invocations of the round function. Note that due to the structure of the Rocca-S round function it is not possible to activate each pipeline in parallel, thus computing each round in sixteen cycles, due to the usage of some states as keys. In fact, our circuit will execute each round in $3 \cdot 16 = 48$ cycles where the pipelines are activated in pairs during three batches. Another complication comes in the form of the encryption function that requires an additional two AES invocations. In order to save circuit area, we aim to reuse two existing AES pipelines of the round function for this task which in turn induces the need for two decryption pipelines that inverse the result before the computation of the round function. More precisely, in sixteen cycles we compute the following two equations

$$C_i^0 = \mathrm{AES}(S[3] \oplus S[5], S[0]) \oplus \overline{M_i}^0$$
$$C_i^1 = \mathrm{AES}(S[4] \oplus S[6], S[2]) \oplus \overline{M_i}^1,$$

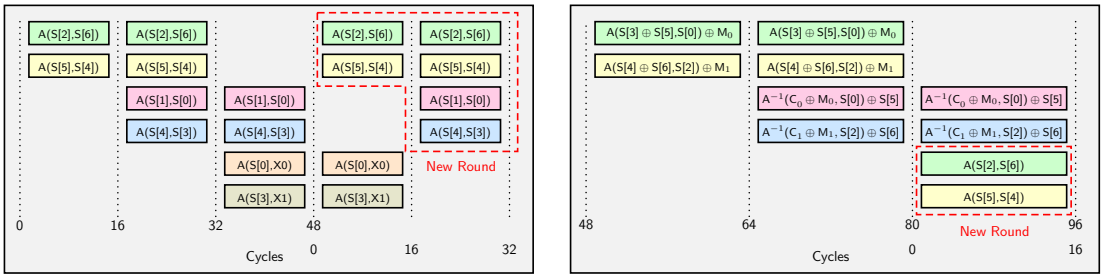
which is then inversed in the subsequent sixteen cycles to reestablish the original state, more specifically

$$S[3] = \mathrm{AES}^{-1}(C_i^0 \oplus \overline{M_i}^0, S[0]) \oplus S[5],$$
$$S[4] = \mathrm{AES}^{-1}(C_i^1 \oplus \overline{M_i}^1, S[2]) \oplus S[6|.$$

Consequently, to process one 256-bit message blocks requires $2 \cdot 16 + 3 \cdot 16 = 80$ clock cycles. A diagrammatic depiction of the byte-serial timeline is shown in Figure 8.5.

Putting all the pieces together we obtain a compact byte-serial Rocca-S circuit that utilizes six Atomic-AES pipelines in order compute both the round and encryption function but remains relatively closed in nature to the initial round-based constructions detailed in Section 8.4.1. A schematic depiction of the byte-serial architecture is given in Figure 8.6.

After synthesis, the byte-serial Rocca-S circuit achieves a gate area reduction of between 75% and 90% compared to the round-based variant described in 8.4.1 and Table 8.3. A breakdown of the various evaluation metrics of the byte-serial architecture is detailed in Table 8.6.



(a) Round function *without* encryption

(b) Round function *with* encryption

Figure 8.5: Timeline of the Rocca-S byte-serial round function for both with (a) and without (b) encryption phase. A round function in the initialisation, associated data and finalisation stages is computed in 48 clock cycles whereas it takes 80 cycles to process one message blocks, hence the total latency is $(32+\alpha)\cdot 48+\beta\cdot 80$ clock cycles where α and β denote the number of associated data and message blocks respectively.

Table 8.6: Synthesis measurements for the byte-serial Rocca-S circuit for two cell libraries and a clock frequency of 10 MHz. The S-box was implemented using low-area circuit S.

	Area		Critical Path	Latency (cycles)		Throughput	Power	Energy (nJ)	
	μm^2	GE	ns	Short	Long	Mbps	μW	Short	Long
NanGate 15 nm	2851	14501	0.46	2368	401728	6.941	0.157	37.18	6307.13
NanGate 45 nm	10038	12580	4.88	2368	401728	0.656	0.543	128.58	21813.83
UMC 65 nm	17660	12264	12.57	2368	401728	0.255	0.134	31.61	5363.07
TSMC 90 nm	33147	11744	8.11	2368	401728	0.395	0.305	72.11	12232.62

8.5 Software Implementation

Ultimately, we also evaluated the performance of Rocca-S and show that modifications only incur small overhead to the performance, despite the increase of number of AES round functions in one round of state update. For the comparison to existing algorithms, we included Rocca-S as well as AEGIS, SNOW-V, and Tiaoxin-346 to OpenSSL 3.1.0-dev and measured their performances with the *speed* command.[31] The evaluation is performed on a PC with Intel(R) Core(TM) i9 12900K @ 2.40 GHz with 32GB RAM. As shown in Table 8.7, Rocca-S can achieve 230 Gbps in the encryption only mode, which is the fastest in our comparison even compared to 128-bit algorithms. In the AEAD mode, Rocca-S also shows the highest performance and achieves 205 Gbps. The reference implementation for our scheme can be found in Appendix D.4.

8.6 Conclusion

In this chapter, we propose the AES-based authenticated encryption scheme Rocca-S with a 256-bit key and a 256-bit tag. Unlike existing schemes for 5G and 6G, Rocca-S can guarantee

[31] The implementation of SNOW-V was published in [68], and implementations of Tiaoxin-346 and AEGIS are available at `https://github.com/floodyberry/supercop`

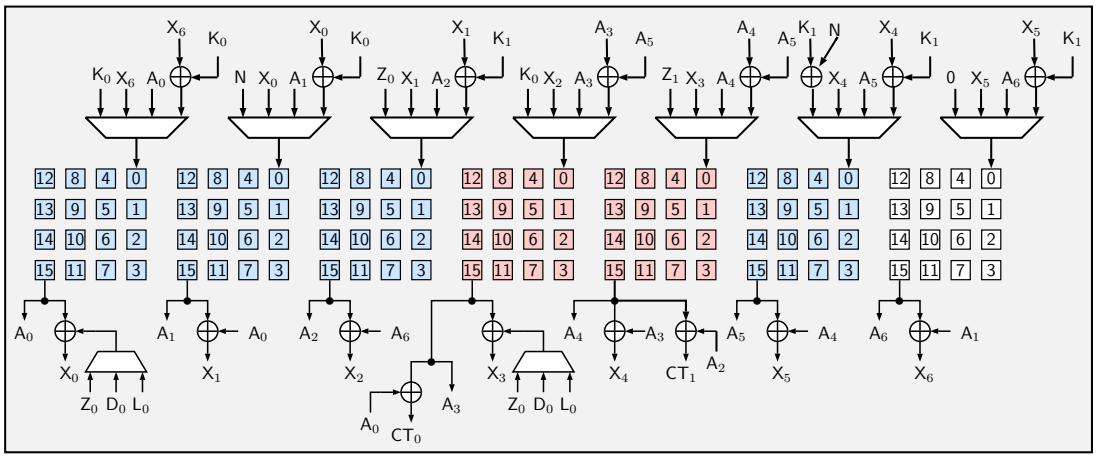


Figure 8.6: Byte-serial Rocca-S circuit. Blue pipelines denote the modified Atomic-AES circuits from [44] and pipelines marked in red are equipped with both encryption and decryption capabilities. Finally, the seventh white state is an empty rotating shift register. For the sake of simplicity, control logic has been omitted from the figure.

256/128-bit security for not only key recovery attacks but also forgery attacks in the classic and quantum settings, respectively. In spite of higher security levels, Rocca-S achieves a speed of more than 200 Gbps in software. In hardware implementation, Rocca-S is the first cryptographic algorithm to achieve speeds consistently between 1 and 2 Terabits per second without sacrificing other metrics such area or power/energy consumption.

[...] In hindsight, I was lost and didn't have a map to recreate it [...]

Table 8.7: Software throughput (GBps) evaluation of Rocca-S and related schemes in both the encryption-only and AEAD settings.

Scheme	Key Length	Tag Length	Input Size (Bytes)				
			16384	8192	1024	256	64
Encryption-only							
AEGIS-128	128	128	47.10	47.47	46.77	42.93	32.90
AEGIS-128L	128	128	166.26	163.50	146.39	104.35	30.00
Tiaoxin-346-v2	128	128	195.87	191.92	163.50	92.18	34.70
AEGIS-256	256	128	48.24	48.72	48.44	45.33	36.86
AES-256-CBC	256	128	13.74	13.58	13.41	13.56	13.43
AES-256-CTR	256	128	79.78	80.58	75.49	61.33	31.39
ChaCha20	256	128	32.29	31.91	31.56	15.38	7.80
SNOW-V	256	128	68.27	67.54	65.15	56.99	39.67
Rocca	256	128	226.15	223.92	197.19	132.22	52.80
Rocca-S	256	256	230.08	225.84	201.77	136.24	54.83
AEGIS-128	128	128	46.76	45.42	32.42	16.32	5.38
AEGIS-128L	128	128	151.53	137.49	60.37	20.68	5.40
Tiaoxin-346-v2	128	128	176.94	159.51	68.13	22.90	5.92
AEGIS-256	256	128	47.96	46.50	33.29	16.72	5.52
AES-256-GCM	256	128	60.29	57.67	36.23	15.86	5.06
ChaCha20-Poly1305	256	128	22.40	21.71	15.25	6.15	2.150
SNOW-V-GCM	256	128	37.87	36.60	25.15	12.15	3.97
Rocca	256	128	199.88	177.41	68.98	22.33	6.01
Rocca-S	256	256	205.65	183.22	74.33	24.78	6.65

9 Side-Channels: Partitioning SKINNY

[...] This station is non-operational [...]

The final contribution of this thesis revolves around the investigation and optimisation of first-order Threshold Implementations concerning the SKINNY block cipher. Recall from Section 2.6.3 that SKINNY designates a lightweight family of tweakable block ciphers designed by Beierle et al. [30]. The cipher performs extremely well on both software and hardware platforms, and is the core encryption primitive used in the authenticated encryption scheme Romulus [74] which is a finalist in the NIST lightweight cryptography competition. Moreover, a criterion for the competition is the efficiency of protected circuit implementations. In the 64-bit block size versions of SKINNY, the underlying S-box defined over four bits. Designing Threshold Implementations for 4-bit S-boxes is a well-studied problem [36], and so in this work we focus on the 128-bit block size versions of SKINNY which use an 8-bit S-box, hereafter denoted by S.

S is exceedingly compact and uses only sixteen cross-connected two-input logic gates. Using the fact that S can be decomposed in the form $I \circ H \circ G \circ F$ (hereafter denoted by S_{2222})[32] where each sub-function is quadratic, the designers of SKINNY proposed a first-order TI of SKINNY using a byte-serial circuit. However, when this decomposition is used to construct a TI of a round-based circuit, a single S-box layer takes four cycles to execute. This increases the latency and hence energy consumption per encryption operation in the circuit, as was shown in Chapter 3.

Contributions. In the remainder of this chapter, we take a closer look at first-order Threshold Implementations of the 8-bit substitution box of round-based SKINNY instantiations. As previously mentioned, the only in-depth analysis and indeed proposal of such a masked circuit is that of S_{2222} which appeared in the design paper [30] for the byte-serial variant of SKINNY. This 3-share scheme is likely the optimal choice for a first-order secure realisation in the byte-serial setting when it comes to area, latency and power/energy consumption. However, for round-based circuits, this assertion does not hold true any more. In fact, we propose two novel decompositions that eclipse the existing variant in both latency, power and energy consumption without significantly increasing the circuit area. More specifically, our contributions are summarized as follows:

[32] Note that throughout this chapter we use the notation $S_{i_1 \ldots i_k}$ to denote decompositions of the same S-box S into k component S-boxes of algebraic degrees $i_1 \ldots i_k$.

1. We devise an approach that exploits the simple 4×4 cross-connected structure of S and automatizes the search for decompositions and thus Threshold Implementations.

2. The proposed technique is then used as a gateway to efficiently decompose S into three quadratic functions $S_{222} = H \circ G \circ F$ that is computed over three cycles. The resulting 3-share masked circuit exhibits a similar area footprint to S_{2222} but cuts the number of required cycles for an encryption by one quarter and consumes around 30% less energy across different clock frequencies and cell libraries.

3. In a second step, by extending the previous technique, we propose a decomposition of S into two cubic functions $S_{33} = G \circ F$ that is thus computed in two cycles. The corresponding 4-share TI halves the number of encryption cycles and consumes 30% less energy while moderately increasing the circuit area relative to S_{2222}. We emphasise that neither of the above circuits require additional randomness beyond the initial plaintext masking.

4. We provide an extensive suite of synthesis measurements on both ASIC and FPGA targets for all investigated schemes showcasing the advantages of both S_{222} and S_{33}.

5. The proposed schemes are proven sound via the SILVER verification framework [92] that performs its analysis on ASIC netlists, which in our case are generated by the NanGate 45 nm standard cell library. In addition, we perform practical leakage assessments using the TVLA methodology [117] by taking power traces on FPGA targets.

The contents of this chapter were presented in 2021 at the 22nd International Conference on Cryptology in India (Indocrypt-2021).

Outline. The chapter unfolds as follows: Section 2 reiterates some preliminaries regarding masking and Threshold Implementations. Subsequently in Section 3, we detail the derivation of S_{222} and S_{33} . Synthesis results are given in Section 4 and leakage assessment is performed in Section 5. Finally, we conclude in Section 6.

9.1 Partitioning the S-Box

Algorithm 3 portrayed the 8-bit S-box of SKINNY as a procedure, more formally it is given by the iterative mapping

$$\Pi' \circ T \circ [\Pi \circ T]^3 (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7),$$

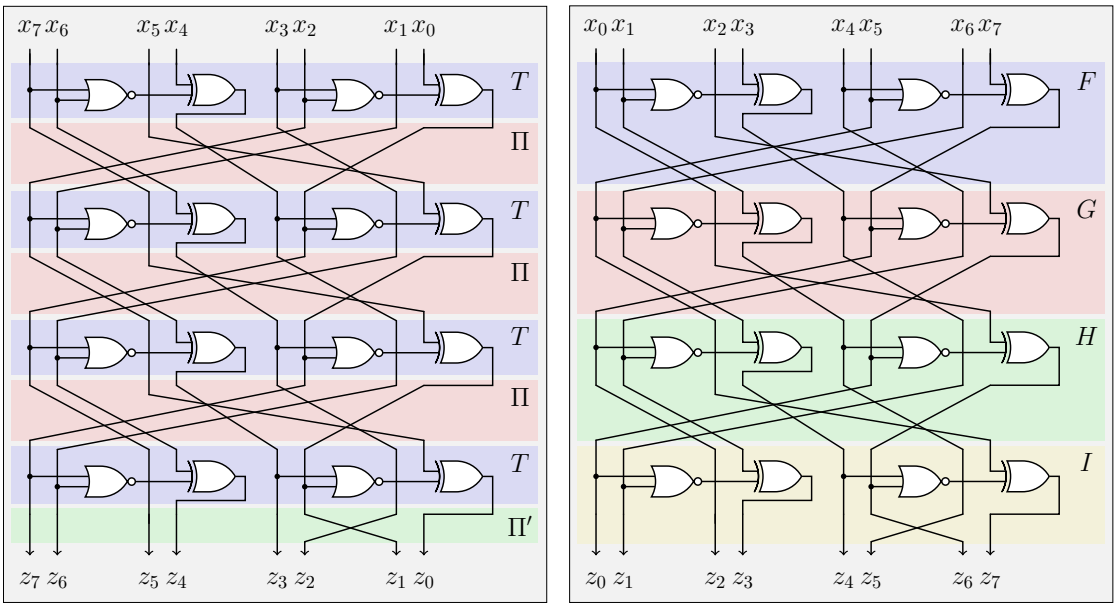
composed of a transformation T and two bitwise permutations Π, Π' such that

$$\begin{aligned} T(x_0, \ldots, x_6, x_7) &= (x_0, x_1, x_2, x_3 \oplus (x_0 \,\overline{\vee}\, x_1), x_4, x_5, x_6, x_7 \oplus (x_4 \,\overline{\vee}\, x_5)) \\ \Pi(x_0, \ldots, x_6, x_7) &= (x_5, x_6, x_0, x_1, x_3, x_7, x_4, x_2) \\ \Pi'(x_1, \ldots, x_6, x_7) &= (x_0, x_1, x_2, x_3, x_4, x_6, x_5, x_7). \end{aligned}$$

Here, $\overline{\vee}$ denotes the logical NOR gate, i.e., $x \,\overline{\vee}\, y = xy \oplus x \oplus y \oplus 1$. A graphical depiction of the 8-bit S-box circuit is given in Figure 9.1a and the quadratic decomposition S_{2222} proposed in the white paper [30] is depicted in Figure 9.1b. Note that the highest algebraic degree of six is reached in output term z_0. The full expression of each term is given below. Note that in order to ease notation the multiplication operator is omitted.

$$
\begin{aligned}
z_0 &= x_0x_1x_4x_5 \oplus x_0x_1x_4 \oplus x_0x_1x_5 \oplus x_0x_1x_7 \oplus x_0x_4x_5 \oplus x_0x_4 \oplus x_0x_5 \oplus \\
&\quad x_0x_7 \oplus x_1x_4x_5 \oplus x_1x_4 \oplus x_1x_5 \oplus x_1x_7 \oplus x_2 \oplus x_3x_4x_5 \oplus x_3x_4 \oplus \\
&\quad x_3x_5 \oplus x_3x_7 \\
z_1 &= x_0x_1 \oplus x_0 \oplus x_1 \oplus x_3 \oplus 1,\ z_2 = x_4x_5 \oplus x_4 \oplus x_5 \oplus x_7 \oplus 1, \\
z_3 &= x_0x_1x_2 \oplus x_0x_1x_4x_5 \oplus x_0x_1x_4 \oplus x_0x_1x_5 \oplus x_0x_1x_7 \oplus x_0x_1 \oplus x_0x_2 \oplus \\
&\quad x_0x_4x_5 \oplus x_0x_4 \oplus x_0x_5 \oplus x_0x_7 \oplus x_0 \oplus x_1x_2 \oplus x_1x_4x_5 \oplus x_1x_4 \oplus x_1x_5 \oplus \\
&\quad x_1x_7 \oplus x_1 \oplus x_2x_3 \oplus x_3x_4x_5 \oplus x_3x_4 \oplus x_3x_5 \oplus x_3x_7 \oplus x_3 \oplus x_4 \\
z_4 &= x_4x_5 \oplus x_4x_7 \oplus x_5 \oplus x_6 \oplus x_7,\ z_5 = x_1 \oplus x_5x_6 \oplus x_5 \oplus x_6 \oplus 1, \\
z_6 &= x_0x_1x_4x_5x_6 \oplus x_0x_1x_4x_6 \oplus x_0x_1x_5x_6x_7 \oplus x_0x_1x_5x_7 \oplus x_0x_1x_5 \oplus \\
&\quad x_0x_1x_6x_7 \oplus x_0x_4x_5x_6 \oplus x_0x_4x_6 \oplus x_0x_5x_6x_7 \oplus x_0x_5x_7 \oplus x_0x_5 \oplus \\
&\quad x_0x_6x_7 \oplus x_0 \oplus x_1x_2 \oplus x_1x_3x_4x_5 \oplus x_1x_3x_4 \oplus x_1x_3x_5 \oplus x_1x_3x_7 \oplus \\
&\quad x_1x_4x_5x_6 \oplus x_1x_4x_5 \oplus x_1x_4x_6 \oplus x_1x_4 \oplus x_1x_5x_6x_7 \oplus x_1x_5x_7 \oplus \\
&\quad x_1x_6x_7 \oplus x_1x_7 \oplus x_1 \oplus x_2x_5x_6 \oplus x_2x_5 \oplus x_2x_6 \oplus x_3x_4x_5x_6 \oplus x_3x_4x_6 \oplus \\
&\quad x_3x_5x_6x_7 \oplus x_3x_5x_7 \oplus x_3x_5 \oplus x_3x_6x_7 \oplus x_5x_6 \oplus x_5 \oplus x_6 \\
z_7 &= x_0x_1x_4x_5x_6x_7 \oplus x_0x_1x_4x_5x_7 \oplus x_0x_1x_4x_5 \oplus x_0x_1x_4x_6x_7 \oplus x_0x_1x_5x_6 \oplus \\
&\quad x_0x_1x_6x_7 \oplus x_0x_4x_5x_6x_7 \oplus x_0x_4x_5x_7 \oplus x_0x_4x_6x_7 \oplus x_0x_4x_7 \oplus x_0x_5x_6 \oplus \\
&\quad x_0x_5 \oplus x_0x_6x_7 \oplus x_0x_6 \oplus x_0x_7 \oplus x_0 \oplus x_1x_2x_4x_5 \oplus x_1x_2x_4x_7 \oplus x_1x_2x_5 \oplus \\
&\quad x_1x_2x_6 \oplus x_1x_2x_7 \oplus x_1x_2 \oplus x_1x_3x_4x_5x_6 \oplus x_1x_3x_4x_6 \oplus x_1x_3x_4x_7 \oplus \\
&\quad x_1x_3x_4 \oplus x_1x_3x_5x_6 \oplus x_1x_3x_6x_7 \oplus x_1x_4x_5x_6x_7 \oplus x_1x_4x_5x_6 \oplus x_1x_4x_5x_7 \oplus \\
&\quad x_1x_4x_6x_7 \oplus x_1x_4x_6 \oplus x_1x_4 \oplus x_1x_5 \oplus x_1x_6 \oplus x_1x_7 \oplus x_1 \oplus x_2x_4x_5x_6x_7 \oplus \\
&\quad x_2x_4x_5x_7 \oplus x_2x_4x_5 \oplus x_2x_4x_6x_7 \oplus x_2x_5x_6x_7 \oplus x_2x_5x_6 \oplus x_2x_5x_7 \oplus \\
&\quad x_2x_6x_7 \oplus x_3x_4x_5x_6x_7 \oplus x_3x_4x_5x_7 \oplus x_3x_4x_5 \oplus x_3x_4x_6x_7 \oplus x_3x_5x_6 \oplus \\
&\quad x_3x_6x_7 \oplus x_4x_5x_6x_7 \oplus x_4x_5x_7 \oplus x_4x_6x_7 \oplus x_4x_7 \oplus x_5x_6x_7 \oplus x_5x_6 \oplus \\
&\quad x_5x_7 \oplus x_6x_7 \oplus x_6 \oplus x_7 \oplus 1
\end{aligned}
$$

In [112], the authors demonstrated how to decompose the S-box S_P of the PRESENT block cipher into two quadratic S-boxes F,and G such that $S_P = G \circ F$. This enabled the authors to construct a 3-share TI of PRESENT by constructing Threshold Implementations of F and G separately with a register bank in between which suppresses and thus prevents the glitches produced by the F layer from propagating to the G layer. This means that every evaluation of the shared S-box requires two cycles to complete. However, this is compensated by the fact



(a) (b)

Figure 9.1: (a) Definition of the 8-bit SKINNY substitution box given the transformation T and two permutations Π, Π'. (b) TI decomposition proposed in [30] using four quadratic functions F, G, H and I.

that the construction requires only three shares and thus the total silicon area required for the circuit is minimal. The approach used by the authors to obtain the decomposition can be summarised as follows:

1. Evaluate all quartets of 4-bit vectorial Boolean functions f_0, f_1, f_2, f_3 such that all the f_i's are quadratic. There are 2^{11} quadratic functions in 4 bits and so a total of 2^{44} such quartets are possible.

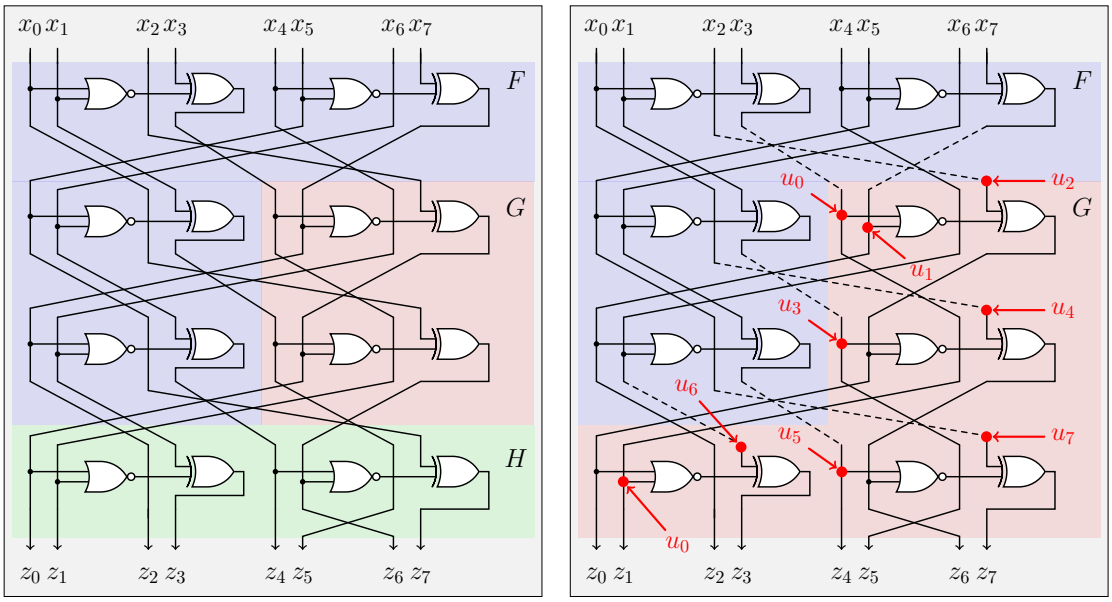
2. Of the above list only filter for the quartets such that the function $F : \{0,1\}^4 \rightarrow \{0,1\}^4$ with $F(x_0, x_1, x_2, x_3) = (f_0, f_1, f_2, f_3)$ is a bijective S-box.

3. For all such F check if $G = S_P \circ F^{-1}$ is also a quadratic S-box. If so, output the pair of S-boxes (G, F).

It was later shown in [36] that S_P belongs to the affine equivalence class $\mathscr{C}_{266}$ of 4-bit S-boxes. All S-boxes in this class allows decomposition into two quadratic S-boxes. The above approach can not be extended to 8-bit S-boxes even considering the authors' suggested optimisations. To begin with there are 2^{37} quadratic functions over 8 bits, and therefore the number of octets of the form $f_0, f_1, \ldots, f_7$ will be $2^{37\times 8} = 2^{296}$.

9.1.1 Angle of Attack

As done with PRESENT our goal lies in finding decompositions of the 8-bit SKINNY S-box S that allow for efficient Threshold Implementations in terms of circuit area, latency and en-



(a) (b)

Figure 9.2: (a) $S_{232} = H \circ G \circ F$ decomposition with $\deg(F) = \deg(H) = 2$ and $\deg(G) = 3$. (b) $S_{24} = G \circ F$ decomposition with $\deg(F) = 2$ and $\deg(G) = 4$. We later introduce the terminology S_{Blue} and S_{Red} to denote F, G respectively in (b).

ergy consumption. In turn, this implies finding an appropriate balance between the number of shares, coordinate functions, and their degrees and gate complexity. To obtain a similar decomposition of S let us first state the following definitions:

Definition 8 (i-representable)**.** *A Boolean function B has AND-complexity n, if its circuit can be constructed with a total of n 2-input AND gates or fewer. Its AND-depth is i (or equivalently it is i-representable) if there exists a circuit in which the AND gates can be arranged in i distinct levels in the following sense: All quadratic functions are 1-representable of some order, and a function B_i is i-representable if it can be expressed as $B_i = Q(t_0, t_1, \ldots, t_{m-1})$ where Q is quadratic and the functions $t_0, t_1, \ldots, t_{m-1}$ are each k-representable of some order for $k \leq (i-1)$. B is i-representable of order n if there exists a circuit which constructs it with AND-depth i and AND-complexity n.*

Thus a function which is i-representable of order n can be necessarily implemented by n or a smaller number of 2-input AND gates (connected such that the total AND-depth is at most i) along with other linear gates. Thus all four coordinate functions of S_P are 2-representable of some fixed order, which allows a 3-share TI over two clock cycles.

Regarding S, the eight output functions $z_0, z_1, \ldots, z_7$ are of different algebraic degrees. z_5, z_4, z_2, z_1 are themselves quadratic and their algebraic expressions contain only a single quadratic term and hence are 1-representable of order one. z_3, z_0 have algebraic degree four: the fact that z_0 is 2-representable of order three can be easily deduced from Figure 9.3a: the paths from the input bits to the z_0 node go through exactly three NOR gates arranged so that

the depth is two. We have $z_3 = z_0 \overline{\vee} z_1 + x_4$. Hence z_3 is at most 3-representable (in fact we will later prove that it is 2-representable too). z_7 and z_6 have algebraic degree six and five respectively: they can not be 2-representable since the set of all 2-representable functions contains members of degree four or less.

9.1.2 Exhaustive Partition Search

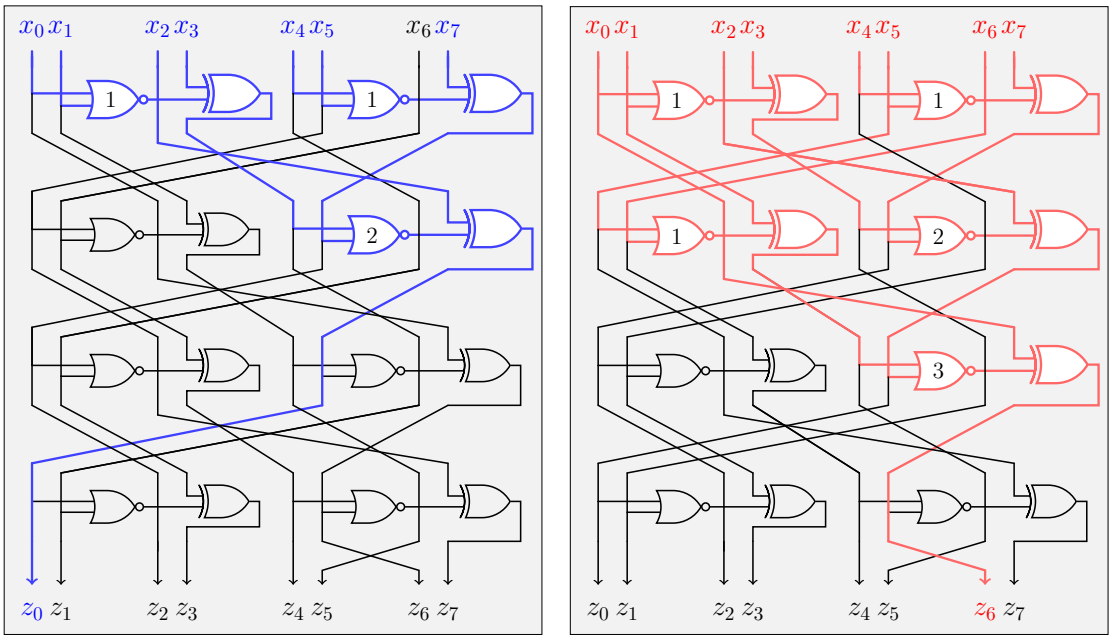
As mentioned, the byte-serial scheme presented in the SKINNY design paper [30], and later adapted to round-based setting presented in Chapter 3, considers a three-share decomposition into four functions of degree two which we denote by S_{2222}. As a consequence, the S-box operation is performed in a pipelined fashion over four clock cycles which incurs a large latency thus energy penalty, i.e., a single encryption of a plaintext takes four times the number of rounds when implemented as a round-based circuit. Since z_7 and z_6 are not 2-representable, the decomposition of S into quadratic S-boxes $F_i \circ F_{i-1} \circ \cdots \circ F_1$ is not possible for $i \leq 2$. Consequently, we aim to decompose every coordinate Boolean function of S into 3-representable functions of low order. Given that S can be realized in only 16 logical two-input gates, a natural approach to obtain efficient decompositions is by partitioning the circuit into connected sub-circuits. For example, the S_{2222} decomposition corresponds to making three horizontal cuts after each row of gates. The number of possible partitions of 16 gates into n sets is n^{16}, however among those, only a small fraction of those partitions respect functional correctness. Hence, if $n = 3$, it is feasible to enumerate all correct partitions. Although this procedure does not admit a 3-representable decomposition of each coordinate function, we found many decompositions of the form $S = H \circ G \circ F$ where $\deg(F) = \deg(H) = 2$ and $\deg(G) = 3$. One such example denoted by S_{232} is shown in Figure 9.2a.

9.1.3 A Deeper Dive

As noted above, all coordinate functions of S except z_7 and z_6 are 3-representable. If we can argue that z_7 and z_6 are also 3-representable, then it becomes straightforward to decompose S into three quadratic S-boxes. z_6 is clearly 3-representable of order five as can be deduced from Figure 9.3b. The set of all paths from the input bits to z_6 traverses exactly five NOR gates arranged in three levels and so the result follows (they are marked in red in Figure 9.3b).

z_7 is of algebraic degree 6 and from Figure 9.1 it is at least 4-representable of order 7. This is because all but one of the 8 NOR gates are used to produce the z_0 bit and they are clearly arranged in 4 levels. However the question is: *Is z_7 also 3-representable of a suitable low order?* If yes, a 3-share first-order TI which evaluates the S-box in only three cycles is possible.

In this part we will show that z_7 is indeed 3-representable of order 8. Note that since the algebraic expression for z_7 is very complex, we avoid directly working with it to prove 3-representability: it would be very difficult to keep the AND-complexity down to a suitable value. Instead, consider the function $\pi(x, y, z) = (x \overline{\vee} y) + z$, whose algebraic expression is given by $xy + x + y + z + 1$. Note that π is completely linear in the last input z. In Figure 9.4, π is represented by a green circular node, and the figure represents the circuit graph for z_7. The figure itself is redrawn by isolating the circuit path for z_7 as in Figure 9.1, and will help



(a) (b)

Figure 9.3: (a) The path up to z_0 is marked in blue. There are 3 NOR gates, whose levels are marked inside. There is a single NOR gate at level 2, which takes inputs from the 2 other level 1 NOR gates in the first row. (b) The path up to z_6 is marked in red. There are 5 NOR gates, whose levels are marked inside. There is a single NOR gate at level 3, which takes inputs from the level 2 NOR gate and another level 1 NOR gate in the second row.

us prove the 3-representability of z_7. Note that Figure 9.4 also makes it clear that z_7 is 4-representable of order 7.

Lemma 2. *It is possible to transform the circuit graph for z_7 according to the transformation (a) → (b) shown in Figure 9.5.*

Proof. This transformation is easy to prove: consider the nodes labeled in darker green in Figure 9.5a. The output bit $e = \pi(b, x_4, x_6)$ is given by the following algebraic expression:

$$\begin{aligned}
e &= \pi(b, x_4, x_6) = \pi(\pi(x_5, x_4, x_7), x_4, x_6) \\
&= \pi(x_5x_4 + x_5 + x_4 + x_7 + 1, x_4, x_6) \\
&= x_3(x_5x_4 + x_5 + x_4 + x_7 + 1) + x_4 + (x_5x_4 + x_5 + x_4 + x_7 + 1) + x_6 + 1 \\
&= x_7x_4 + x_5x_4 + x_5 + x_7 + x_6 \\
&= x_4(x_7 + x_5) + (x_7 + x_5) + x_4 + (x_6 + x_4 + 1) + 1 \\
&= \pi(x_7 + x_5, x_4, x_6 + x_4 + 1)
\end{aligned}$$

□

Lemma 3. *It is possible to transform the circuit graph for z_7 according to the transformation (a) → (b) shown in Figure 9.6. Thus, z_7 is 3-representable of order eight.*

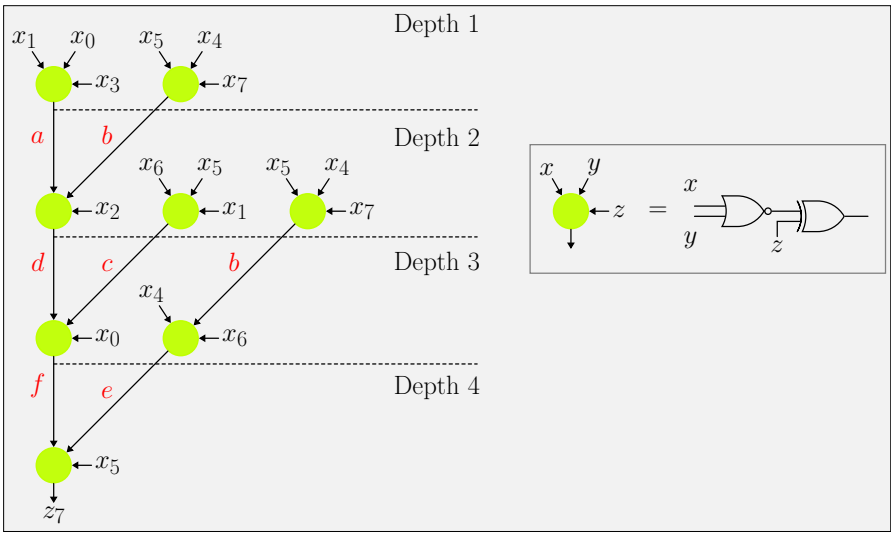


Figure 9.4: Circuit graph for z_7. Its AND-complexity is 7 (note the gate $\pi(x_5, x_4, x_7)$ is shown twice for a clearer representation).

Proof. The proof for this transformation is slightly more involved. Consider again the gates labeled in dark green in Figure 9.6a. They lie entirely in levels 3 and 4 of the circuit graph, and takes as input the signals d, c, e, x_0, x_5 and produces z_7 as output such that

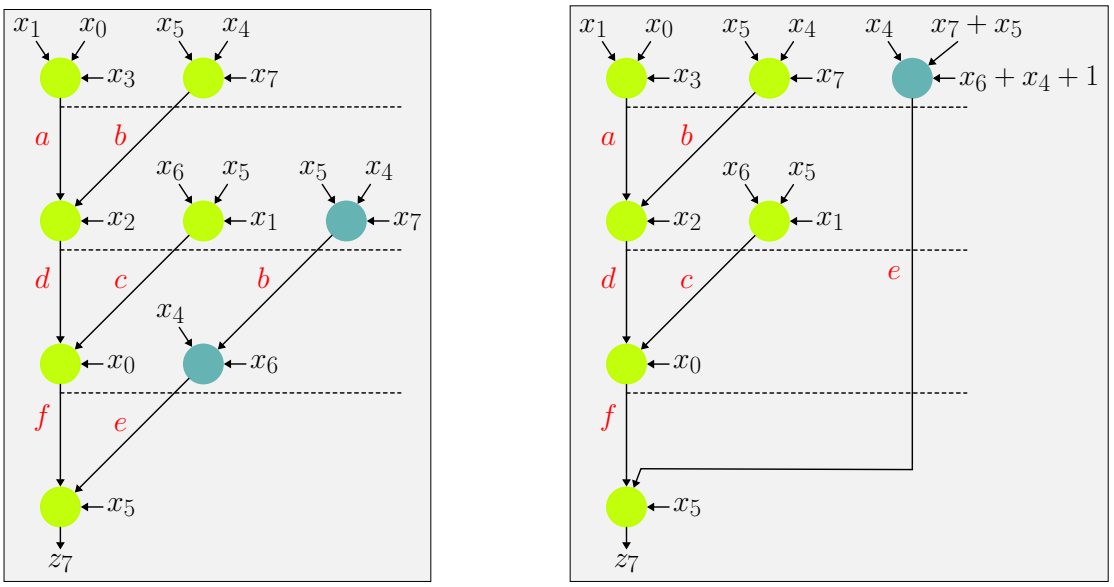
$$\begin{aligned}
z_7 &= \pi(f, e, x_5) = \pi(\pi(d, c, x_0), e, x_5) \\
&= \pi(dc + d + c + x_0 + 1, e, x_5) \\
&= e(dc + d + c + x_0 + 1) + e + (dc + d + c + x_0 + 1) + x_5 + 1 \\
&= edc + ed + ec + ex_0 + dc + d + c + x_0 + x_5 \\
&= d(ec + e + c + 1) + ec + ex_0 + c + x_0 + x_5 \\
&= d(\pi(e, c, 0)) + (ec + e + c + 1) + d + (d + e + 1 + ex_0 + x_0 + x_5) \\
&= d(\pi(e, c, 0)) + \pi(e, c, 0) + d + (ex_0 + e + x_0 + x_5 + 1 + d) \\
&= \pi\Big(\pi(e, c, 0), d, d + \pi(e, x_0, x_5)\Big)
\end{aligned}$$

This completes the proof of the transformation. Figure 9.6 also proves that z_7 can be constructed with a AND-depth of 3 and so it is 3-representable. □

This allows us to decompose the S-box into $H \circ G \circ F = S_{222}$, where $F : \{0,1\}^8 \to \{0,1\}^8$, $G : \{0,1\}^8 \to \{0,1\}^9$ and $H : \{0,1\}^9 \to \{0,1\}^8$ are each quadratic S-boxes. The algebraic expressions are as follows:

$$\begin{aligned}
F(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) &= (u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7) \\
u_7 = x_3 + x_1x_0 + x_1 + x_0 + 1,\ & u_6 = x_7 + x_5x_4 + x_5 + x_4 + 1 \\
u_5 = x_7x_4 + x_7 + x_6 + x_5x_4 + x_5,\ & u_4 = x_6x_5 + x_6 + x_5 + x_1 + 1
\end{aligned}$$

$$u_3 = x_5,\ u_2 = x_4,\ u_1 = x_2,\ u_0 = x_0$$
$$G(u_0,u_1,u_2,u_3,u_4,u_5,u_6,u_7) = (v_0,v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8)$$
$$v_8 = u_1 + u_7u_6 + u_7 + u_6 + 1,\ v_7 = u_2 + u_1u_7 + u_1 + u_7u_6 + u_6$$
$$v_6 = u_5u_4,\ v_5 = u_7,\ v_4 = u_6,\ v_3 = u_5,\ v_2 = u_4,$$
$$v_1 = u_3,\ v_0 = u_0$$
$$H(v_0,v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8) = (z_0,z_1,z_2,z_3,z_4,z_5,z_6,z_7)$$
$$z_7 = v_6v_8 + v_6 + v_3v_8 + v_0v_3 + v_2v_8 + v_2 + v_8 + v_0 + v_1,$$
$$z_6 = v_0 + v_8v_2 + v_8 + v_2 + 1,$$
$$z_5 = v_2,\ z_4 = v_3,\ z_3 = v_7,\ z_2 = v_4,\ z_1 = v_5,\ z_0 = v_8$$

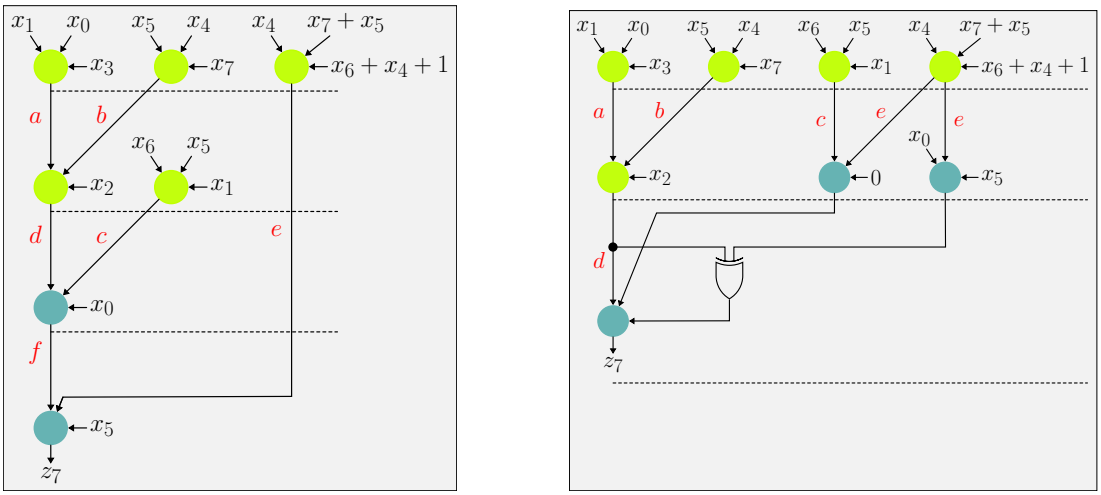


(a) (b)

Figure 9.5: Transformation (a)→(b) of the circuit graph of z_7 for Lemma 2.

9.1.4 Decomposition into Two Cubic S-boxes

It is straightforward to decompose S into two S-boxes of degree 4 each. For example from $S_{2222} = I \circ H \circ G \circ F$, both $G \circ F$ and $I \circ H$ are degree 4 S-boxes. A first order TI of degree 4 S-box requires 5 shares. So by using the above decomposition we can implement a circuit that evaluates the shared S-box in only 2 clock cycles but requires 5 shares. Suppose we were able to decompose S into two cubic S-boxes: if this were so then a first order TI would need only 4 shares. Such a circuit would require smaller circuit area and hence consume less power on account of the reduced number of shares and also consume less energy to encrypt a plaintext on account of the reduced power consumption. So in principle it is an interesting exercise to see if this decomposition is at all possible.



(a) (b)

Figure 9.6: Transformation (a)→(b) of the circuit graph of z_7 for Lemma 3, proving that z_7 is 3-representable of order 8 (right).

In order to decompose S into two cubic S-boxes, we can again mount an exhaustive search on all partitions of two sets as done in Section 9.1.3. This procedure does not yield such a decomposition but many of the form $S = G \circ F$ where $\deg(F) = 2$ and $\deg(G) = 4$ or vice-versa as shown in Figure 9.2b. However, we can follow a similar strategy as in detailed in the previous section. We begin with the following definition:

Definition 9. *A Boolean function B is said to have cubic depth 2, if it can be expressed as $B = C(c_1, c_2, \ldots, c_n)$ where $C, c_1, c_2, \ldots, c_n$ are each either cubic Boolean functions or functions of algebraic degree strictly less than 3. The cubic order of such a function is said to be i, if the total number cubic terms in the algebraic expressions of $C, c_1, c_2, \ldots, c_n$ combined is i.*

A lower cubic depth allows us to construct a TI of the given function lower number of cycles using only 4 shares. Since every cubic term wxy in the algebraic expression has to be opened up as $(w_1 + w_2 + w_3 + w_4)(x_1 + x_2 + x_3 + x_4)(y_1 + y_2 + y_3 + y_4)$ to construct a 4 share TI, a low cubic order will obviously help make the circuit more lightweight and efficient. It is straightforward to see that $z_6, z_5, \ldots, z_0$ all have cubic depth 2: z_5, z_4, z_2, z_1 are quadratic. z_0 has algebraic degree 4 and we have already seen that it is 2-representable, and so it automatically follows that its cubic depth is 2 and cubic order is 0. The fact that z_6, z_3 also have cubic depth equal to two can be seen in Figure 9.2b of the SKINNY S-box circuit. The part shaded in blue is an 8×8 quadratic S-box, call it S_{Blue} and the part in red is another 8×8 S-box of degree 4 (call it S_{Red}). We obviously have $S = S_{\text{Red}} \circ S_{\text{Blue}}$. The corresponding algebraic expressions are given below:

$$S_{\text{Blue}}(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7)$$
$$u_0 = x_5,\ u_1 = x_4,\ u_2 = x_4x_5 + x_4x_7 + x_5 + x_6 + x_7,\ u_3 = x_0,$$
$$u_4 = x_1 + x_5x_6 + x_5 + x_6 + 1,\ u_5 = x_2,$$

$$u_6 = x_4x_5 + x_4 + x_5 + x_7 + 1,\ u_7 = x_0x_1 + x_0 + x_1 + x_3 + 1$$
$$S_{\text{Red}}(u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7) = (z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7)$$
$$z_0 = u_5 + u_6u_7 + u_6 + u_7 + 1,\ z_1 = u_7,\ z_2 = u_6$$
$$z_3 = u_1 + u_5u_7 + u_5 + u_6u_7 + u_6,\ z_4 = u_2,\ z_5 = u_4,$$
$$z_6 = u_3 + u_4u_5 + u_4u_6u_7 + u_4u_6 + u_4u_7 + u_5 + u_6u_7 + u_6 + u_7,$$
$$\begin{aligned} z_7 = {} & u_0 + u_2u_3 + u_2u_4u_5 + u_2u_4u_6u_7 + u_2u_4u_6 + u_2u_4u_7 + u_2u_5 \\ & + u_2u_6u_7 + u_2u_6 + u_2u_7 + u_2 + u_3 + u_4u_5 + u_4u_6u_7 \\ & + u_4u_6 + u_4u_7 + u_5 + u_6u_7 + u_6 + u_7 + 1 \end{aligned}$$

From the expression we can see that z_6 as the output of S_{Red} is a cubic function with only a single cubic term. And since the u_i's are at most quadratic this follows that the cubic depth of z_1 is 2 and its cubic order is 1. Also the expression for z_3 is quadratic in S_{Red}, which proves that not only is its cubic depth 2 and cubic order 0, but it is also 2-representable. It is elementary to verify that its AND-complexity is 3. The only problematic part is proving that z_7 also has cubic depth 2 of some suitably low order, since it is not clear from this decomposition. Note that there is only one degree 4 term $u_2u_4u_6u_7$ in the expression of z_7. Also $u_2u_6 = x_4x_5x_6 + x_4x_6 + x_6 + x_6x_5 + x_7x_6$ is a cubic expression in the x_i's. Therefore, we construct the following S-box $S'_{\text{Blue}} : \{0,1\}^8 \rightarrow \{0,1\}^9$ where

$$S'_{\text{Blue}}(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8)$$

such that $u_0 = x_4x_5x_6 + x_4x_6 + x_6 + x_6x_5 + x_7x_6$ and the other u_i's are as defined for S_{Blue}. Correspondingly we define $S'_{\text{Red}} : \{0,1\}^9 \rightarrow \{0,1\}^8$ where

$$S'_{\text{Red}}(u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8) = (z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7)$$

such that

$$\begin{aligned} z_7 = {} & u_1 + u_3u_4 + u_3u_5u_6 + u_0u_5u_8 + u_3u_5u_7 + u_3u_5u_8 + u_3u_6 \\ & + u_3u_7u_8 + u_3u_7 + u_3u_8 + u_3 + u_4 + u_5u_6 + u_5u_7u_8 + u_5u_7 \\ & + u_5u_8 + u_6 + u_7u_8 + u_7 + u_8 + 1 \end{aligned}$$

and the other z_i's are as defined for S_{Red}. Since both S'_{Blue} and S'_{Red} are cubic S-boxes this proves that the cubic depth of z_7 is also 2. It is easy to count that there are 5 cubic terms in the modified expression of z_7 and one cubic term in the expression for u_0, which implies that the cubic order of z_7 is 6. Since we also have that $S = S'_{\text{Red}} \circ S'_{\text{Blue}}$, this also gives us the cubic decomposition required to construct a first order TI using 4 input/output shares that can evaluate the shared S-box in just 2 cycles.

9.2 Hardware Implementation

After decomposing the S-box into quadratic and cubic component functions, we use the direct sharing approach to obtain the algebraic expressions for each of the individual shares of the masked S-box. In all cases, except for S_{2222}, correction terms were required to ensure uniform sharing. All investigated schemes were synthesised using the *no_autoungroup* compilation directive that respects entity boundaries and thus prevents the optimizer from potentially interfering with the threshold properties of the circuit.

In Table 9.1, we detail the measurements for the investigated S-box circuits and note that both in latency and power, S_{222} as well as S_{33} eclipse the other variants. This trend is amplified when the entire SKINNY circuit is implemented as shown in Table 9.2. We denote by SKINNY$_{i_1\ldots i_k}$ the full SKINNY circuit using the S-box $S_{i_1\ldots i_k}$.

Table 9.1: ASIC synthesis measurements for the investigated substitution boxes.

Scheme	Library	Latency (Cycles)	Area (GE)	Timing (ns)	Power (μW)	
					10 MHz	100 MHz
S_{2222}	NanGate 15 nm	4	769.0	0.04	7.52	51.41
	NanGate 45 nm	4	584.3	0.24	43.81	157.1
	UMC 65 nm	4	597.9	1.15	5.74	56.14
	TSMC 90 nm	4	501.3	0.69	14.06	139.1
S_{232}	NanGate 15 nm	3	1027.4	0.09	9.96	71.87
	NanGate 45 nm	3	915.3	1.11	86.15	166.2
	UMC 65 nm	3	941.3	3.82	7.99	77.86
	TSMC 90 nm	3	865.1	1.74	17.23	169.8
S_{222}	NanGate 15 nm	3	676.5	0.05	5.65	38.44
	NanGate 45 nm	3	600.6	0.31	46.77	154.4
	UMC 65 nm	3	616.5	1.73	5.40	52.63
	TSMC 90 nm	3	541.8	0.87	12.23	120.8
S_{33}	NanGate 15 nm	2	1185	0.11	9.55	61.73
	NanGate 45 nm	2	1906	1.21	159.7	553.7
	UMC 65 nm	2	1924	4.79	14.35	139.1
	TSMC 90 nm	2	1049	1.71	14.82	145.5

9.3 Leakage Evaluation

SILVER [92] is a formal verification tool for masking countermeasures. For a given security property [63], the tool exhaustive evaluates the input netlist using reduced-ordered binary decision diagrams. We compile the netlist for the S_{222} and S_{33} S-boxes using the NanGate 45 nm standard cell library and verified that both netlists satisfied first-order probing security

Table 9.2: ASIC synthesis figures for all investigated schemes for three cell libraries.

Scheme	Library	Latency (Cycles)	Area (GE)	Critical Path (ns)	Power (μW)		Energy (nJ/128 bits)	
					10 MHz	100 MHz	10 MHz	100 MHz
SKINNY-128$_{2222}$	NanGate 15 nm	872	4461	0.31	21.91	186.2	1.911	1.623
Byte-Serial	NanGate 45 nm	872	5039	0.51	100.6	343.5	8.772	2.995
	UMC 65 nm	872	4989	1.59	25.82	244.5	2.251	2.132
	TSMC 90 nm	872	4989	1.59	25.82	244.5	2.251	2.132
SKINNY-256$_{2222}$	NanGate 15 nm	1040	5280	0.33	25.90	219.6	2.694	2.284
Byte-Serial	NanGate 45 nm	1040	5993	0.52	120.7	420.8	12.55	4.376
	UMC 65 nm	1040	5876	1.64	30.33	287.3	3.154	2.988
	TSMC 90 nm	1040	5876	1.64	30.33	287.3	3.154	2.988
SKINNY-384$_{2222}$	NanGate 15 nm	1208	6122	0.35	26.97	222.5	3.258	2.688
Byte-Serial	NanGate 45 nm	1208	6949	0.57	140.3	496.4	16.94	5.993
	UMC 65 nm	1208	6782	1.69	34.98	333.1	4.226	4.024
	TSMC 90 nm	1208	6782	1.69	34.98	333.1	4.226	4.024
SKINNY-128$_{2222}$	NanGate 15 nm	160	16952	0.07	103.0	707.1	1.28	1.13
	NanGate 45 nm	160	14637	0.47	917.3	2199	14.68	3.518
	UMC 65 nm	160	15116	2.03	93.57	898.7	1.497	1.438
	TSMC 90 nm	160	12594	1.23	239.3	2353.3	3.83	3.77
SKINNY-256$_{2222}$	NanGate 15 nm	192	18742	0.07	114.7	665.3	2.20	1.28
	NanGate 45 nm	192	16315	0.47	1041	2490	19.98	4.781
	UMC 65 nm	192	16735	2.12	103.1	990.3	1.979	1.901
	TSMC 90 nm	192	14074	1.23	263.5	2591.0	5.06	4.97
SKINNY-384$_{2222}$	NanGate 15 nm	224	20558	0.07	126.5	733.2	2.83	1.64
	NanGate 45 nm	224	17991	0.47	1166	2774	26.12	6.213
	UMC 65 nm	224	18357	2.12	113.4	1088	2.538	2.437
	TSMC 90 nm	224	15528	1.23	287.4	2825.2	6.44	6.33
SKINNY-128$_{222}$	NanGate 15 nm	120	17155	0.92	110.5	647.2	1.33	0.78
	NanGate 45 nm	120	14899	0.66	474.9	1890	5.699	2.268
	UMC 65 nm	120	15413	3.40	93.05	892.7	1.156	1.071
	TSMC 90 nm	120	13233	1.64	217.6	2136.8	2.61	2.56
SKINNY-256$_{222}$	NanGate 15 nm	144	18946	0.09	123.1	724.1	1.77	1.04
	NanGate 45 nm	144	16576	0.66	501.5	2010	7.222	2.894
	UMC 65 nm	144	17031	3.51	104.0	997.1	1.497	1.436
	TSMC 90 nm	144	14711	1.64	244.0	2396.0	3.51	3.45
SKINNY-384$_{222}$	NanGate 15 nm	168	20757	0.09	136.3	803.4	2.28	1.35
	NanGate 45 nm	168	18253	0.66	632.1	2298	10.62	3.861
	UMC 65 nm	168	18654	3.51	115.6	1109	1.942	1.863
	TSMC 90 nm	168	16165	1.65	270.7	2658.3	4.55	4.47
SKINNY-128$_{33}$	NanGate 15 nm	80	27092	0.14	206.5	1310.7	1.65	1.05
	NanGate 45 nm	80	23954	0.88	980.1	3200	7.84	2.56
	UMC 65 nm	80	24923	4.13	139.1	1391	1.11	1.11
	TSMC 90 nm	80	22724	2.26	286.4	2800.7	2.29	2.24
SKINNY-256$_{33}$	NanGate 15 nm	96	29169	0.14	221.8	1406.9	2.13	1.35
	NanGate 45 nm	96	25888	0.87	1109	3678	10.64	3.53
	UMC 65 nm	96	26767	4.23	159.3	1542	1.53	1.48
	TSMC 90 nm	96	24433	2.26	318.5	3116.8	3.06	2.99
SKINNY-384$_{33}$	NanGate 15 nm	112	27964	0.66	137.5	1190	1.54	1.33
	NanGate 45 nm	112	27820	0.87	1382	4001	15.48	4.48
	UMC 65 nm	112	28621	4.24	147.7	1636	1.65	1.83
	TSMC 90 nm	112	26171	2.27	351.7	3442	3.94	3.85

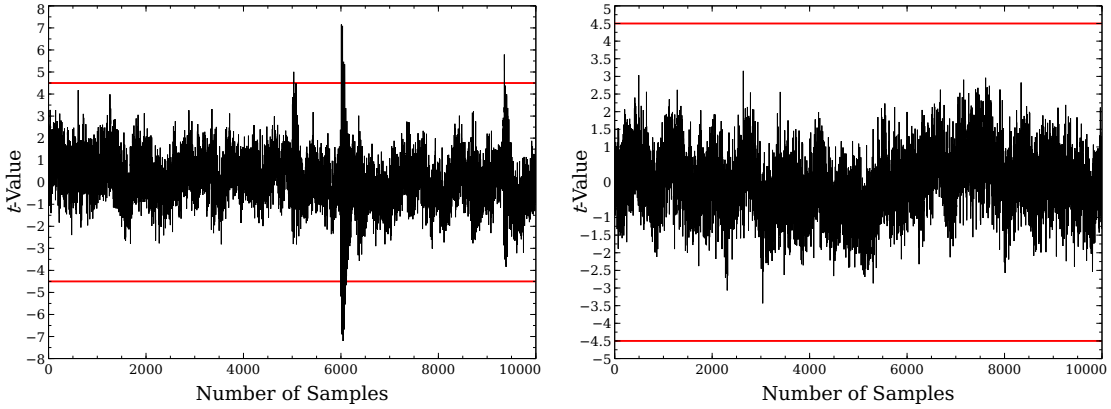
in the standard and robust probing models as well as uniformity.

Secondly, similarly to the leakage assessment in Chapter 8, we verified our masked SKINNY circuits using the TVLA methodology [117] using Welch's t-test and the min-p strategy for null hypothesis rejection. In particular, we perform non-specific fixed versus random t-tests, where we aim to determine the validity of the null hypothesis that *encryptions with a fixed and uniformly sampled plaintext admit the same mean power consumption* (i.e., are indistinguishable under first-order statistical analysis). Following the state of the art [7, 117], we set a threshold $|t| > 4.5$ for any t-value to reject the null hypothesis. To perform t-tests, power traces of SKINNY$_{222}$ and SKINNY$_{33}$ were measured using the Sakura-X and Sasebo-GII power side-channel leakage evaluation boards. These boards contain a core FPGA target on which a cryptographic circuit can be programmed, allowing the evaluation of custom hardware implementations of cryptographic primitives. To reduce noise, the boards contain an additional FPGA for communication with the host PC, which is used to send keys and plaintexts and read ciphertexts. Moreover, these boards contain direct connectors for oscilloscope probes, facilitating the acquisition of the power supply voltage traces for the side-channel evaluation. The encryption FPGA has direct connections to header pins on the board, allowing easy synchronisation using a dedicated trigger signal.

The Sakura-X board hosts a more recent FPGA as part of the Xilinx 7-Series (Kintex-7, XC7K160T), while the Sasebo-GII board contains an older FPGA from the 5-Series (Virtex-5, XC5VLX30) architecture. To prevent unwanted optimisations during the FPGA toolchain synthesis and implementation, *DONT_TOUCH*, *KEEP_HIERARCHY*, and *KEEP* constraints are added. The clock frequency of our designs is constrained to a low 3 MHz on both boards. All power measurements are performed using a Tektronix MDO3104 oscilloscope with a sampling rate of 1 GS/s and AC coupling; we take 10000 sample points per trace with 1 microsecond horizontal graduations. To perform non-specific t-tests, all encryptions were performed with a fixed key. The cryptographic primitive was reset before every encryption to ensure identical initial conditions for both the fixed and random traces. Consequently, this allowed us to record traces for t-tests in a deterministic interleaving fashion, where a random plaintext preceded a fixed plaintext and vice-versa, reducing bias in any one dataset from potential variation in noise and environmental conditions over time. To avoid leakage arising from generating random masks on the DUT itself, we sent pre-masked plaintext shares to the FPGA.

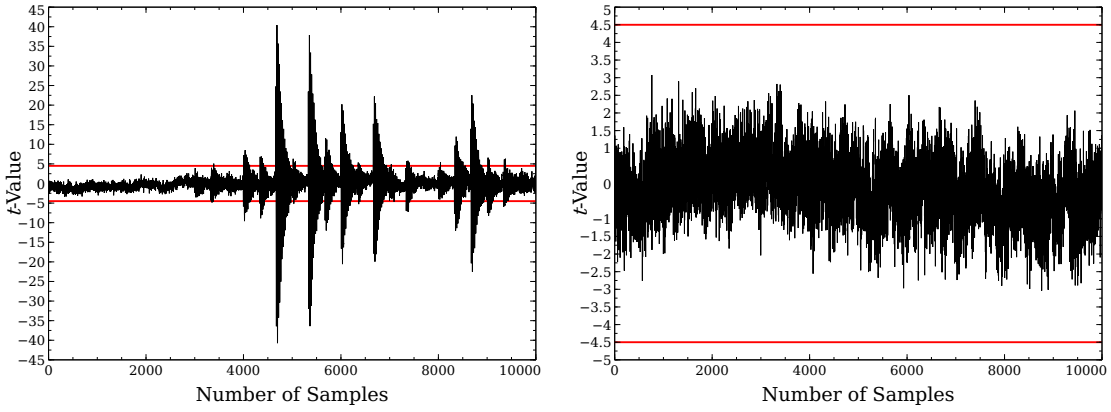
In order to verify the soundness of our experimental setup, we first ran t-tests in the *masks off* setting by setting all but one share of the plaintext to the zero vector. We perform the masks off t-tests on 10000 traces for each design. Figures 9.10a and 9.10b plot a sample trace for the two designs. Note that we take traces corresponding to 10 rounds of an encryption operation in each experiment. Recall that executing a round of SKINNY with S_{33} uses two cycles, rather than three like with S_{222}. The encryption operation for the SKINNY$_{33}$ experiments only begins after a few thousand data points, whereas we record from the beginning of an encryption for the SKINNY$_{222}$ experiments.

The results in Figures 9.7a and 9.8a indicate that there is potentially exploitable leakage with just 10000 traces, even with measurements with low SNR taken on the Sakura-X board.



(a) (b)

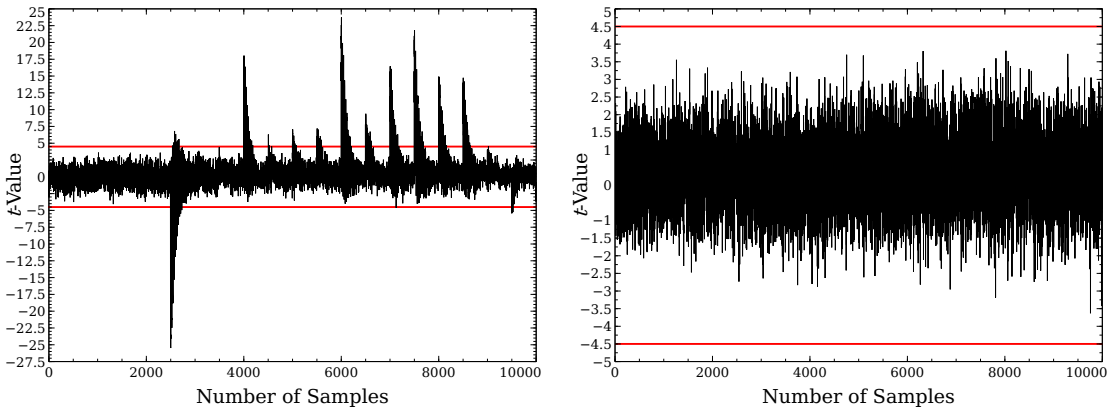
Figure 9.7: t-test results for SKINNY$_{222}$ on the Sakura-X with (a) 10000 traces and masks off and (b) one million traces and masks on.



(a) (b)

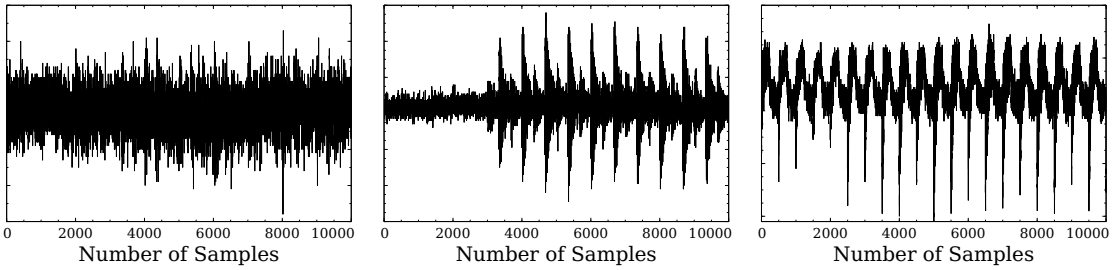
Figure 9.8: t-test results for SKINNY$_{33}$ on the Sakura-X with (a) 10000 traces and masks off and (b) one million traces and masks on.

We then record 1 million traces with randomly generated masks to assess the first-order security of our designs (Figures 9.7b and 9.8b). Our results indicate that the threshold of 4.5 is not crossed in any of the trace samples, and that no leakage is detected with this number of traces. Since Threshold Implementations are well-studied, we expect these results to hold with a larger number of traces also. To demonstrate that our Threshold Implementation of SKINNY$_{222}$ is secure even on a smaller FPGA with a higher SNR (lower noise), we also performed t-tests with both randomly generated and zero masks using the Sasebo-II side-channel evaluation board. Figure 9.10c shows a sample trace taken during the experiments, where the power consumption from the encryption operation in each clock cycle is clearly visible. Figure 9.9 shows the t-values obtained for the power traces. As before, with 10000 traces in the masks off setting, we note substantial leakage. With one million traces and masks on, we find no evidence of leakage.



(a) (b)

Figure 9.9: t-test results for SKINNY$_{222}$ on the Sasebo-II with (a) 10000 traces and masks off and (b) one million traces and masks on.



(a) (b) (c)

Figure 9.10: Sample power traces of encryption operations for (a) SKINNY$_{222}$ on the Sakura-X, (b) SKINNY$_{33}$ on the Sakura-X and (c) SKINNY$_{222}$ on the Sasebo-II.

9.4 Conclusion

In this chapter, we re-envision first-order TI for the SKINNY family of tweakable block ciphers in the round-based setting. More specifically, we propose different decompositions of the 8-bit S-box which enable significantly more efficient implementations of a protected SKINNY circuit in terms of latency and energy consumption, which we demonstrate through an extensive suite of synthesis benchmarks. We conclude by assessing the security of our designs via leveraging existing leakage detection and formal verification techniques.

[...] The home you said you came with, a moment of weakness labeled revelation [...]

10 Conclusion

> Es wird mir ganz angst um die Welt, wenn ich an die Ewigkeit denke. Beschäftigung, Woyzeck, Beschäftigung! ewig das ist ewig, das ist ewig, das siehst du ein; nun ist es aber wieder nicht ewig und das ist ein Augenblick, ja, ein Augenblick – Woyzeck, es schaudert mich, wenn ich denk, dass sich die Welt in einem Tag herumdreht, was eine Zeitverschwendung, wo soll das hinaus? Woyzeck, ich kann kein Mühlrad mehr sehn, oder ich werd' melancholisch.
>
> — *Woyzeck, Georg Büchner, 1878*

In this thesis, we chronicle a perennial scientific journey through an ever-evolving but insular domain. It is the progeny of an opaque dendritic mesh of roots whose conglomerative assembly bears the conferred fruits of knowledge that encapsulates both the past and an eternal presence hence manifests itself as a snapshot of a microcosm whose ethereal structure is painted in the colours of progress that precludes the subjective futility inherent to frontier-expanding endeavours. This manuscript is a metaphorical *Solanaceae*, a delirious frenzy that delivers the cryptographic congregation from their self-imposed lopsidedness. In this cathartic haze, let us, for one last time, summon the cornerstones of this voyage and in the process illuminate prospective diversions.

We have commenced with a survey-type chapter in which we examined the energy consumption landscape of nine NIST LWC second-round candidate designs bootstrapped from lightweight block-ciphers. The methodology enabled us to extend the energy model from [18] to modes of operation thus paved the way for a better understanding of the effects of implementation choices. Our work stands today as the most meticulous energy study in the field of cryptography. Nonetheless, there is ample scope for extensions. For example, the non-trivial affair of padding incomplete blocks is vital in real-world applications but remains unexplored in Chapter 3. Consequently, the effects that dedicated padding modules exert on energy consumption are unclear. Similarly, due to the choice of investigating only schemes built around block ciphers the applicability of the energy to other paradigms, such as permutation-based designs or more complex constructions (see Chapter 8), is opaque. Finally, we ask, Can the acquired information in the course of this study be put to use in the form of a dedicated energy-optimal AEAD scheme for hardware environments?

In the subsequent chapter, we have directed our gaze onto the realm of stream ciphers,

in particular the class of Trivium-like constructions. By exploring the implementation space of round-unrolled variants, we discovered a natural algebraic circuit structure that, for the first time, made it possible to formulate a heuristic energy model for a class of stream ciphers. This model is also versatile enough to be used in the design of novel energy-optimal ciphers, which we demonstrated with the proposal of Trivium-LE(F) and Trivium-LE(S). Both of which stand as the most energy-efficient encryption solutions known in the cryptography literature. We are optimistic that this work will echo in the design of future hardware-oriented lightweight stream ciphers. However for a general adoption of the proposed energy model, some unilluminated avenues still need to be explored, such as the question regarding generalisation of the model to arbitrary Boolean functions that differ from the exceedingly simple strands in the case of Trivium. A general model would enable the composition of improved new schemes that are not based on previous designs. In the meantime, it is worth investigating whether the strand-based energy model is applicable to other stream ciphers, apart from the families that have been covered in the chapter.

In the first section of this thesis, we have concluded with the proposal of Atom, a small-state stream cipher in the Grain family. Atom guarantees a 128-bit security with a state size of only 128 bits. The design ranks as both the most area and energy-efficient choice among other stream cipher constructions that produce one keystream bit per clock cycle and as a feature of a key size of 128 bits. The automatised approach for finding the Atom-state update routine, which would resist known attacks, resulted in a complex component functions that exhibit comparably large circuit areas. In this sense, it is worth investigating whether this complexity can be broken down in order to facilitate an even better performing cipher. Similarly, as Atom adheres to the methodology that gave rise to the Grain family of stream ciphers, a potential research project could involve the investigation of designs that offer the same security guarantees, without compromising the performance aspects but that are based on different structural paradigms.

Book number two began with a treatise on serialised block ciphers and modes of operations. In particular, we have revisit the swap-and-rotate technique, as proposed in [11], and we have extended its applicability. More specifically, we design bit-serial circuits for AES, SKINNY, and GIFT; they compute one round function in exactly 128 cycles, which corresponds to their state sizes in bits. This results in bit-serial constructions that stand as both the most area-efficient and latency-efficient compared to existing designs. In the second part of this chapter, we have repurposed the proposed circuits as building blocks for four NIST LWC second-round candidates and have demonstrate that these modes can be implemented in a serial fashion, with only a small increase in circuit complexity. In its current state, to find efficient mappings that implement a given function, the swap-and-rotate methodology heavily relies on the ingenuity of the researcher. This ad-hoc approach neither scales nor guarantees that a devised sequence of swaps is optimal. Hence, a potential research direction lies in lifting the technique to a higher-level representation that could ease the automatisation of the process; in the sense that, for a given function and cycle budget, this technique could find the most optimal sequence of swaps that implement it. On a lower level, the control logic required for a swap-and-rotate circuit is rather involved, which means that a lower number

of swaps might not directly result in a saved circuit area if the corresponding control logic comes with overhead.

One AEAD construction that was omitted in Chapter 6 is a bit-serial implementation of GIFT-COFB that competes as a NIST LWC finalist. This is due to the complex nature of its auxiliary modules as part of the combined-feedback mode of operation, specifically the finite-field multiplication executed between encryptions. Chapter 7 was dedicated to the construction of efficient bit-serial circuits by using the swap-and-rotate construction of GIFT, as detailed in Section 6.4. We propose two designs that feature different area-latency trade-offs and, ultimately, a first-order threshold implementation. All three proposals currently stand as the most area-efficient circuits in their respective category and sensibly extend the canon of the GIFT-COFB design space. Still, the work on hardware metrics of cryptographic implementations can never be considered truly completed, which also holds true for GIFT-COFB for which other trade-offs such as area/throughput for bit-serial circuits are worthy of a future investigation. The combined efforts that revolved around GIFT and GIFT-COFB (along with SKINNY) make it the most thoroughly examined candidate of the final NIST-LWC round, in terms of its hardware properties.

Rocca-S is the name of the ultra-high throughput AEAD scheme that we have put forward in Chapter 8; it exhibits its prowess both in the hardware and software realms. In its design philosophy of existing schemes, Rocca-S is reminiscent of AEGIS [126] and Tiaoxin [110], as its round function is also put together by a parallel chain of multiple AES round function modules. The cipher features both a key and tag size of 256 bits, which makes it key-recovery and, for the first time in the known literature, forgery secure in the post-quantum setting. Rocca-S achieves a maximum throughput of over two Terabits per second on hardware and over 200 Gigabits per second, as a software algorithm that eclipses related constructions by more than 50%. In this sense, this mode operation is adequately primed as component for the upcoming standardisation process of 6G telecommunication networks that will presumably support unprecedented throughput rates and guarantees against quantum adversaries. In terms of research, Rocca-S should serve as a beacon for future cryptographic work that concerns 6G. This means that efforts should be guided towards the optimisation of the round function in order to lower the number of AES modules and to keep security and performance competitive. This naturally includes the search for a round function composition or even entirely new designs that admit even higher throughput rates in both hardware and software.

Lastly, for the main body of work in this thesis, we have concluded with a chapter on first-order threshold implementations of the SKINNY block-cipher. The starting point of the endeavour was the fact that the decomposition of its S-box into efficient component functions, more than specified in the SKINNY white paper [30], remained an open problem. We have tackled this challenge by taking a mathematical approach to the task and, in the process, have uncovered both a decomposition into three functions of algebraic degree two and a related variant that corresponds to a decomposition into two functions of degree three. Together, this pair of circuits exhibits a area-latency trade-off that elevates the side-channel-resistant implementations of SKINNY to an astute level but does not answer questions regarding high-order threshold implementations or different probing models.

This is the end, the end of four years of research combined into a single document. In science, the quest for the unknown is an *exitus acta probat*, where discovery, comprehension and achievement are misnomers for hardship, failure, and desperation in a circus that shrouds humanity in favour of often purposeless drudgery. Unfortunately, this thesis is no counterexample. With these words I say, to those who were, those who are, and those who will be; farewell.

Andrea Caforio
Lausanne, 2023

Bibliography

[1] Alexandre Adomnicai and Thomas Peyrin. Fixslicing AES-like Ciphers. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.1 (2021), pp. 402–425.

[2] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of Grain-128 with optional authentication. In: *Int. J. Wirel. Mob. Comput.* 5.1 (2011), pp. 48–59.

[3] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. How to Abuse and Fix Authenticated Encryption Without Key Commitment. In: *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*. Ed. by Kevin R. B. Butler and Kurt Thomas. USENIX Association, 2022, pp. 3291–3308.

[4] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. Forkcipher: A New Primitive for Authenticated Encryption of Very Short Messages. In: *Advances in Cryptology – ASIACRYPT 2019, Part II*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11922. Lecture Notes in Computer Science. Kobe, Japan: Springer, Heidelberg, Germany, Dec. 2019, pp. 153–182.

[5] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, and Arnab Roy Damian Vizár. ForkAE v.1. In: *NIST Lightweight Cryptography Project* (2019).

[6] Frederik Armknecht and Vasily Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. In: *Fast Software Encryption – FSE 2015*. Ed. by Gregor Leander. Vol. 9054. Lecture Notes in Computer Science. Istanbul, Turkey: Springer, Heidelberg, Germany, Mar. 2015, pp. 451–470.

[7] Victor Arribas, Begül Bilgin, George Petrides, Svetla Nikova, and Vincent Rijmen. Rhythmic Keccak: SCA Security and Low Latency in HW. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.1 (2018), pp. 269–290.

[8] Roberto Avanzi. The QARMA Block Cipher Family. In: *IACR Transactions on Symmetric Cryptology* 2017.1 (2017), pp. 4–44.

[9] Fatih Balli, Andrea Caforio, and Subhadeep Banik. The Area-Latency Symbiosis: Towards Improved Serial Encryption Circuits. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.1 (2021), pp. 239–278.

[10] Subhadeep Banik. Some Results on Sprout. In: *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*. Ed. by Alex Biryukov and Vipul Goyal. Vol. 9462. Lecture Notes in Computer Science. Springer, 2015, pp. 124–139.

[11] Subhadeep Banik, Fatih Balli, Francesco Regazzoni, and Serge Vaudenay. Swap and Rotate: Lightweight Linear Layers for SPN-based Blockciphers. In: *IACR Transactions on Symmetric Cryptology* 2020.1 (2020), pp. 185–232.

[12] Subhadeep Banik, Khashayar Barooti, and Takanori Isobe. Cryptanalysis of Plantlet. In: *IACR Transactions on Symmetric Cryptology* 2019.3 (2019), pp. 103–120.

[13] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In: *Advances in Cryptology – ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. Lecture Notes in Computer Science. Auckland, New Zealand: Springer, Heidelberg, Germany, Nov. 2015, pp. 411–436.

[14] Subhadeep Banik, Andrey Bogdanov, Atul Luykx, and Elmar Tischhauser. SUNDAE: Small Universal Deterministic Authenticated Encryption for the Internet of Things. In: *IACR Transactions on Symmetric Cryptology* 2018.3 (2018), pp. 1–35.

[15] Subhadeep Banik, Andrey Bogdanov, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, Elmar Tischhauser, and Yosuke Todo. SUNDAE-GIFT v1.0. In: *NIST Lightweight Cryptography Project* (2019).

[16] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-AES v2.0. Cryptology ePrint Archive, Report 2016/1005. 2016.

[17] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-AES: A Compact Implementation of the AES Encryption/Decryption Core. In: *Progress in Cryptology - INDOCRYPT 2016: 17th International Conference in Cryptology in India*. Ed. by Orr Dunkelman and Somitra Kumar Sanadhya. Vol. 10095. Lecture Notes in Computer Science. Kolkata, India: Springer, Heidelberg, Germany, Dec. 2016, pp. 173–190.

[18] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Exploring Energy Efficiency of Lightweight Block Ciphers. In: *SAC 2015: 22nd Annual International Workshop on Selected Areas in Cryptography*. Ed. by Orr Dunkelman and Liam Keliher. Vol. 9566. Lecture Notes in Computer Science. Sackville, NB, Canada: Springer, Heidelberg, Germany, Aug. 2016, pp. 178–194.

[19] Subhadeep Banik, Andrea Caforio, Kazuhide Fukushima, Takanori Isobe, Shisaku Kiyomoto, Fukang Liu, Yuto Nakano, Kosei Sakamoto, Nobuyuki Takeuchi, and Ravi Anand. Rocca-S: Ultra High-Throughput and Quantum-Secure Authenticated Encryption. 2023.

[20] Subhadeep Banik, Andrea Caforio, Takanori Isobe, Fukang Liu, Willi Meier, Kosei Sakamoto, and Santanu Sarkar. Atom: A Stream Cipher with Double Key Filter. In: *IACR Transactions on Symmetric Cryptology* 2021.1 (2021), pp. 5–36.

[21] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT-COFB v1.1. In: *NIST Lightweight Cryptography Project* (2022).

[22] Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. More Results on Shortest Linear Programs. In: *IWSEC 19: 14th International Workshop on Security, Advances in Information and Computer Security*. Ed. by Nuttapong Attrapadung and Takeshi Yagi. Vol. 11689. Lecture Notes in Computer Science. Tokyo, Japan: Springer, Heidelberg, Germany, Aug. 2019, pp. 109–128.

[23] Subhadeep Banik, Takanori Isobe, Tingting Cui, and Jian Guo. Some cryptanalytic results on Lizard. In: *IACR Transactions on Symmetric Cryptology* 2017.4 (2017), pp. 82–98.

[24] Subhadeep Banik, Takanori Isobe, Willi Meier, Yosuke Todo, and Bin Zhang. TRIAD v1: A Lightweight AEAD and Hash Function Based on Stream Cipher. In: *NIST Lightweight Cryptography Project* (2019).

[25] Subhadeep Banik, Vasily Mikhalev, Frederik Armknecht, Takanori Isobe, Willi Meier, Andrey Bogdanov, Yuhei Watanabe, and Francesco Regazzoni. Towards Low Energy Stream Ciphers. In: *IACR Transactions on Symmetric Cryptology* 2018.2 (2018), pp. 1–19.

[26] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption. In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, Sept. 2017, pp. 321–345.

[27] Lejla Batina, Amitabh Das, Baris Ege, Elif Bilge Kavun, Nele Mentens, Christof Paar, Ingrid Verbauwhede, and Tolga Yalçin. Dietary Recommendations for Lightweight Block Ciphers: Power, Energy and Area Analysis of Recently Developed Architectures. In: *Radio Frequency Identification - Security and Privacy Issues 9th International Workshop, RFIDsec 2013, Graz, Austria, July 9-11, 2013, Revised Selected Papers*. Ed. by Michael Hutter and Jörn-Marc Schmidt. Vol. 8262. Lecture Notes in Computer Science. Springer, 2013, pp. 103–112.

[28] Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In: *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2015, pp. 1–6.

[29] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. SKINNY-AEAD and SKINNY-HASH. In: *NIST Lightweight Cryptography Project* (2019).

[30] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In: *Advances in Cryptology – CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. Lecture

Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2016, pp. 123–153.

[31] Ishai Ben-Aroya and Eli Biham. Differtial Cryptanalysis of Lucifer. In: *Advances in Cryptology – CRYPTO'93*. Ed. by Douglas R. Stinson. Vol. 773. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1994, pp. 187–199.

[32] Côme Berbain, Henri Gilbert, and Antoine Joux. Algebraic and Correlation Attacks against Linearly Filtered Non Linear Feedback Shift Registers. In: *SAC 2008: 15th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Roberto Maria Avanzi, Liam Keliher, and Francesco Sica. Vol. 5381. Lecture Notes in Computer Science. Sackville, New Brunswick, Canada: Springer, Heidelberg, Germany, Aug. 2009, pp. 184–198.

[33] Eli Biham. A Fast New DES Implementation in Software. In: *Fast Software Encryption – FSE'97*. Ed. by Eli Biham. Vol. 1267. Lecture Notes in Computer Science. Haifa, Israel: Springer, Heidelberg, Germany, Jan. 1997, pp. 260–272.

[34] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In: *Advances in Cryptology – CRYPTO'90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1991, pp. 2–21.

[35] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-Order Threshold Implementations. In: *Advances in Cryptology – ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. Lecture Notes in Computer Science. Kaoshiung, Taiwan, R.O.C.: Springer, Heidelberg, Germany, Dec. 2014, pp. 326–343.

[36] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. Threshold Implementations of All 3×3 and 4×4 S-Boxes. In: *Cryptographic Hardware and Embedded Systems – CHES 2012*. Ed. by Emmanuel Prouff and Patrick Schaumont. Vol. 7428. Lecture Notes in Computer Science. Leuven, Belgium: Springer, Heidelberg, Germany, Sept. 2012, pp. 76–91.

[37] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In: *Advances in Cryptology – ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. Lecture Notes in Computer Science. Kyoto, Japan: Springer, Heidelberg, Germany, Dec. 2000, pp. 1–13.

[38] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In: *Cryptographic Hardware and Embedded Systems – CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. Lecture Notes in Computer Science. Vienna, Austria: Springer, Heidelberg, Germany, Sept. 2007, pp. 450–466.

[39] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. Quantum Security Analysis of AES. In: *IACR Transactions on Symmetric Cryptology* 2019.2 (2019), pp. 55–93.

[40] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Dec. 2012, pp. 208–225.

[41] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum Cryptanalysis of Hash and Claw-Free Functions. In: *LATIN 1998: Theoretical Informatics, 3rd Latin American Symposium*. Ed. by Claudio L. Lucchesi and Arnaldo V. Moura. Vol. 1380. Lecture Notes in Computer Science. Campinas, Brazil: Springer, Heidelberg, Germany, Apr. 1998, pp. 163–169.

[42] Andrea Caforio, Fatih Balli, and Subhadeep Banik. Complete Practical Side-Channel-Assisted Reverse Engineering of AES-Like Ciphers. In: *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*. Ed. by Vincent Grosso and Thomas Pöppelmann. Vol. 13173. Lecture Notes in Computer Science. Springer, 2021, pp. 97–117.

[43] Andrea Caforio, Fatih Balli, and Subhadeep Banik. Energy Analysis of Lightweight AEAD Circuits. In: *Cryptology and Network Security - 19th International Conference, CANS 2020, Vienna, Austria, December 14-16, 2020, Proceedings*. Ed. by Stephan Krenn, Haya Shulman, and Serge Vaudenay. Vol. 12579. Lecture Notes in Computer Science. Springer, 2020, pp. 23–42.

[44] Andrea Caforio, Fatih Balli, and Subhadeep Banik. Melting SNOW-V: improved lightweight architectures. In: *J. Cryptogr. Eng.* 12.1 (2022), pp. 53–73.

[45] Andrea Caforio, Fatih Balli, Subhadeep Banik, and Francesco Regazzoni. A Deeper Look at the Energy Consumption of Lightweight Block Ciphers. In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*. IEEE, 2021, pp. 170–175.

[46] Andrea Caforio and Subhadeep Banik. A Study of Persistent Fault Analysis. In: *Security, Privacy, and Applied Cryptography Engineering - 9th International Conference, SPACE 2019, Gandhinagar, India, December 3-7, 2019, Proceedings*. Ed. by Shivam Bhasin, Avi Mendelson, and Mridul Nandi. Vol. 11947. Lecture Notes in Computer Science. Springer, 2019, pp. 13–33.

[47] Andrea Caforio, Subhadeep Banik, Yosuke Todo, Willi Meier, Takanori Isobe, Fukang Liu, and Bin Zhang. Perfect Trees: Designing Energy-Optimal Symmetric Encryption Primitives. In: *IACR Trans. Symmetric Cryptol.* 2021.4 (2021), pp. 36–73.

[48] Andrea Caforio, Daniel Collins, Subhadeep Banik, and Francesco Regazzoni. A Small GIFT-COFB: Lightweight Bit-Serial Architectures. In: *Progress in Cryptology - AFRICACRYPT 2022 - 13th International Conference on Cryptology in Africa, Fes, Morocco, July 18-20, 2022, Proceedings*. Ed. by Abderrahmane Nitaj and Lhoussain El Fadil. Vol. 13143. Lecture Notes in Computer Science. Springer, 2022, pp. 246–267.

[49] Andrea Caforio, Daniel Collins, Ognjen Glamocanin, and Subhadeep Banik. Improving First-Order Threshold Implementations of SKINNY. In: *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings*. Ed. by Avishek Adhikari, Ralf Küsters, and Bart Preneel. Vol. 13143. Lecture Notes in Computer Science. Springer, 2021, pp. 246–267.

[50] Andrea Caforio, F. Betül Durak, and Serge Vaudenay. Beyond Security and Efficiency: On-Demand Ratcheting with Security Awareness. In: *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Juan Garay. Vol. 12711. Lecture Notes in Computer Science. Virtual Event: Springer, Heidelberg, Germany, May 2021, pp. 649–677.

[51] D. Canright. A Very Compact S-Box for AES. In: *Cryptographic Hardware and Embedded Systems – CHES 2005*. Ed. by Josyula R. Rao and Berk Sunar. Vol. 3659. Lecture Notes in Computer Science. Edinburgh, UK: Springer, Heidelberg, Germany, Aug. 2005, pp. 441–455.

[52] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. In: *Journal of Cryptology* 31.3 (July 2018), pp. 885–916.

[53] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. Saturnin: a suite of lightweight symmetric algorithms for post-quantum security. In: *NIST Lightweight Cryptography Project* (2019).

[54] Avik Chakraborti, Anupam Chattopadhyay, Muhammad Hassan, and Mridul Nandi. TriviA and uTriviA: two fast and secure authenticated encryption schemes. In: *Journal of Cryptographic Engineering* 8.1 (Apr. 2018), pp. 29–48.

[55] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas Lopez, Mridul Nandi, and Yu Sasaki. LOTUS-AEAD and LOCUS-AEAD. In: *NIST Lightweight Cryptography Project* (2019).

[56] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, and Mridul Nandi. HYENA. In: *NIST Lightweight Cryptography Project* (2019).

[57] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-Based Authenticated Encryption: How Small Can We Go? In: *Journal of Cryptology* 33.3 (July 2020), pp. 703–741.

[58] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. In: *Fast Software Encryption – FSE'97*. Ed. by Eli Biham. Vol. 1267. Lecture Notes in Computer Science. Haifa, Israel: Springer, Heidelberg, Germany, Jan. 1997, pp. 149–165.

[59] Joan Daemen, Pedro Maat Costa Massolino, Alireza Mehrdad, and Yann Rotella. The Subterranean 2.0 Cipher Suite. In: *IACR Transactions on Symmetric Cryptology* 2020.S1 (2020), pp. 262–294.

[60] Joan Daemen and Vincent Rijmen. The Block Cipher Rijndael. In: *Smart Card Research and Applications, This International Conference, CARDIS '98, Louvain-la-Neuve, Belgium, September 14-16, 1998, Proceedings*. Ed. by Jean-Jacques Quisquater and Bruce Schneier. Vol. 1820. Lecture Notes in Computer Science. Springer, 1998, pp. 277–284.

[61] Christophe De Cannière. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In: *ISC 2006: 9th International Conference on Information Security*. Ed. by Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel. Vol. 4176. Lecture Notes in Computer Science. Samos Island, Greece: Springer, Heidelberg, Germany, Aug. 2006, pp. 171–186.

[62] Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: *Cryptographic Hardware and Embedded Systems – CHES 2009*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. Lecture Notes in Computer Science. Lausanne, Switzerland: Springer, Heidelberg, Germany, Sept. 2009, pp. 272–288.

[63] Lauren De Meyer, Begül Bilgin, and Oscar Reparaz. Consolidating Security Notions in Hardware Masking. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.3 (2019), pp. 119–147.

[64] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In: *Advances in Cryptology – EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. Lecture Notes in Computer Science. Cologne, Germany: Springer, Heidelberg, Germany, Apr. 2009, pp. 278–299.

[65] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. In: *NIST Lightweight Cryptography Project* (2022).

[66] Orr Dunkelman and Nathan Keller. Treatment of the initial value in Time-Memory-Data Tradeoff attacks on stream ciphers. In: *Inf. Process. Lett.* 107.5 (2008), pp. 133–137.

[67] Maria Eichlseder, Marcel Nageler, and Robert Primas. Analyzing the Linear Keystream Biases in AEGIS. In: *IACR Transactions on Symmetric Cryptology* 2019.4 (2019), pp. 348–368.

[68] Patrik Ekdahl, Thomas Johansson, Alexander Maximov, and Jing Yang. A new SNOW stream cipher called SNOW-V. In: *IACR Transactions on Symmetric Cryptology* 2019.3 (2019), pp. 1–42.

[69] Muhammed F. Esgin and Orhun Kara. Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks. In: *SAC 2015: 22nd Annual International Workshop on Selected Areas in Cryptography*. Ed. by Orr Dunkelman and Liam Keliher. Vol. 9566. Lecture Notes in Computer Science. Sackville, NB, Canada: Springer, Heidelberg, Germany, Aug. 2016, pp. 67–85.

[70] Martin Feldhofer, J. Wolkerstorfer, and Vincent Rijmen. AES implementation on a grain of sand. In: *Information Security, IEE Proceedings* 152 (Nov. 2005), pp. 13–20.

[71] Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim. Pyjamask v1.0. In: *NIST Lightweight Cryptography Project* (2019).

[72] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying Grover's Algorithm to AES: Quantum Resource Estimates. In: *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*. Ed. by Tsuyoshi Takagi. Fukuoka, Japan: Springer, Heidelberg, Germany, Feb. 2016, pp. 29–43.

[73] Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In: *28th Annual ACM Symposium on Theory of Computing*. Philadephia, PA, USA: ACM Press, May 1996, pp. 212–219.

[74] Chun Guo, Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Romulus v1.3. In: *NIST Lightweight Cryptography Project* (2022).

[75] Matthias Hamann, Matthias Krause, and Willi Meier. LIZARD – A Lightweight Stream Cipher for Power-constrained Devices. In: *IACR Transactions on Symmetric Cryptology* 2017.1 (2017), pp. 45–79.

[76] Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for Three-Subset Division Property Without Unknown Subset - Improved Cube Attacks Against Trivium and Grain-128AEAD. In: *Advances in Cryptology – EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, May 2020, pp. 466–495.

[77] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A Stream Cipher Proposal: Grain-128. In: *Proceedings 2006 IEEE International Symposium on Information Theory, ISIT 2006, The Westin Seattle, Seattle, Washington, USA, July 9-14, 2006*. IEEE, 2006, pp. 1614–1618.

[78] Martin Hell, Thomas Johansson, Willi Meier, Jonathan Sönnerup, and Hirotaka Yoshida. Grain-128AEAD. In: *NIST Lightweight Cryptography Project* (2022).

[79] Martin E. Hellman. A cryptanalytic time-memory trade-off. In: *IEEE Trans. Inf. Theory* 26.4 (1980), pp. 401–406.

[80] Ekawat Homsirikamol, William Diehl, Ahmed Ferozpuri, Farnoud Farahmand, Panasayya Yalla, Jens-Peter Kaps, and Kris Gaj. CAESAR Hardware API. Cryptology ePrint Archive, Report 2016/626. 2016.

[81] Jin Hong and Palash Sarkar. New Applications of Time Memory Data Tradeoffs. In: *Advances in Cryptology – ASIACRYPT 2005*. Ed. by Bimal K. Roy. Vol. 3788. Lecture Notes in Computer Science. Chennai, India: Springer, Heidelberg, Germany, Dec. 2005, pp. 353–372.

[82] Akinori Hosoyamada, Akiko Inoue, Ryoma Ito, Tetsu Iwata, Kazuhiko Mimematsu, Ferdinand Sibleyras, and Yosuke Todo. Cryptanalysis of Rocca and Feasibility of Its Security Claim. In: *IACR Transactions on Symmetric Cryptology* 2022.3 (2022), pp. 123–151.

[83] Kai Hu, Siwei Sun, Meiqin Wang, and Qingju Wang. An Algebraic Formulation of the Division Property: Revisiting Degree Evaluations, Cube Attacks, and Key-Independent Sums. In: *Advances in Cryptology – ASIACRYPT 2020, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. Lecture Notes in Computer Science. Daejeon, South Korea: Springer, Heidelberg, Germany, Dec. 2020, pp. 446–476.

[84] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 463–481.

[85] Arpan Jati, Naina Gupta, Anupam Chattopadhyay, Somitra Kumar Sanadhya, and Donghoon Chang. Threshold Implementations of GIFT: A Trade-Off Analysis. In: *IEEE Trans. Inf. Forensics Secur.* 15 (2020), pp. 2110–2120.

[86] Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. Bit-Sliding: A Generic Technique for Bit-Serial Implementations of SPN-based Primitives - Applications to AES, PRESENT and SKINNY. In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, Sept. 2017, pp. 687–707.

[87] Jérémy Jean and Ivica Nikolic. Efficient Design Strategies Based on the AES Round Function. In: *Fast Software Encryption – FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. Lecture Notes in Computer Science. Bochum, Germany: Springer, Heidelberg, Germany, Mar. 2016, pp. 334–353.

[88] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Quantum Differential and Linear Cryptanalysis. In: *IACR Transactions on Symmetric Cryptology* 2016.1 (2016), pp. 71–94.

[89] Stéphanie Kerckhof, François Durvaux, Cédric Hoquet, David Bol, and François-Xavier Standaert. Towards Green Cryptography: A Comparison of Lightweight Ciphers from the Energy Viewpoint. In: *Cryptographic Hardware and Embedded Systems – CHES 2012*. Ed. by Emmanuel Prouff and Patrick Schaumont. Vol. 7428. Lecture Notes in Computer Science. Leuven, Belgium: Springer, Heidelberg, Germany, Sept. 2012, pp. 390–407.

[90] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In: *Advances in Cryptology – ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, Dec. 2010, pp. 130–145.

[91] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential Cryptanalysis of Trivium and KATAN. In: *SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. Lecture Notes in Computer Science. Toronto, Ontario, Canada: Springer, Heidelberg, Germany, Aug. 2012, pp. 200–212.

[92] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In: *Advances in Cryptology – ASIACRYPT 2020, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. Lecture Notes in Computer Science. Daejeon, South Korea: Springer, Heidelberg, Germany, Dec. 2020, pp. 787–816.

[93] Lars R. Knudsen and David Wagner. Integral Cryptanalysis. In: *Fast Software Encryption – FSE 2002*. Ed. by Joan Daemen and Vincent Rijmen. Vol. 2365. Lecture Notes in Computer Science. Leuven, Belgium: Springer, Heidelberg, Germany, Feb. 2002, pp. 112–127.

[94] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Shorter Linear Straight-Line Programs for MDS Matrices. In: *IACR Transactions on Symmetric Cryptology* 2017.4 (2017), pp. 188–211.

[95] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In: *Fast Software Encryption – FSE 2011*. Ed. by Antoine Joux. Vol. 6733. Lecture Notes in Computer Science. Lyngby, Denmark: Springer, Heidelberg, Germany, Feb. 2011, pp. 306–327.

[96] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of Full Sprout. In: *Advances in Cryptology – CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2015, pp. 663–682.

[97] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY Family of Block Ciphers Engineering an Ultra Low-Latency Cipher from Gate Level for Secure Processor Architectures. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.4 (2021), pp. 510–545.

[98] Julia Len, Paul Grubbs, and Thomas Ristenpart. Partitioning Oracle Attacks. In: *USENIX Security 2021: 30th USENIX Security Symposium*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, Aug. 2021, pp. 195–212.

[99] Shun Li, Siwei Sun, Chaoyun Li, Zihao Wei, and Lei Hu. Constructing Low-latency Involutory MDS Matrices with Lightweight Circuits. In: *IACR Transactions on Symmetric Cryptology* 2019.1 (2019), pp. 84–117.

[100] Stefan Lucks. The Saturation Attack - A Bait for Twofish. In: *Fast Software Encryption – FSE 2001*. Ed. by Mitsuru Matsui. Vol. 2355. Lecture Notes in Computer Science. Yokohama, Japan: Springer, Heidelberg, Germany, Apr. 2002, pp. 1–15.

[101] Alexander Maximov. AES MixColumn with 92 XOR gates. Cryptology ePrint Archive, Report 2019/833. 2019.

[102] Alexander Maximov and Alex Biryukov. Two Trivial Attacks on Trivium. In: *SAC 2007: 14th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Carlisle M. Adams, Ali Miri, and Michael J. Wiener. Vol. 4876. Lecture Notes in Computer Science. Ottawa, Canada: Springer, Heidelberg, Germany, Aug. 2007, pp. 36–55.

[103] Alexander Maximov and Patrik Ekdahl. New Circuit Minimization Techniques for Smaller and Faster AES SBoxes. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.4 (2019), pp. 91–125.

[104] David A. McGrew and John Viega. The Security and Performance of the Galois/-Counter Mode of Operation. In: *Progress in Cryptology - INDOCRYPT 2004: 5th International Conference in Cryptology in India*. Ed. by Anne Canteaut and Kapalee Viswanathan. Vol. 3348. Lecture Notes in Computer Science. Chennai, India: Springer, Heidelberg, Germany, Dec. 2004, pp. 343–355.

[105] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On Ciphers that Continuously Access the Non-Volatile Key. In: *IACR Transactions on Symmetric Cryptology* 2016.2 (2016), pp. 52–79.

[106] Brice Minaud. Linear Biases in AEGIS Keystream. In: *SAC 2014: 21st Annual International Workshop on Selected Areas in Cryptography*. Ed. by Antoine Joux and Amr M. Youssef. Vol. 8781. Lecture Notes in Computer Science. Montreal, QC, Canada: Springer, Heidelberg, Germany, Aug. 2014, pp. 290–305.

[107] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Tallinn, Estonia: Springer, Heidelberg, Germany, May 2011, pp. 69–88.

[108] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In: *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*. Ed. by Chuankun Wu, Moti Yung, and Dongdai Lin. Vol. 7537. Lecture Notes in Computer Science. Springer, 2011, pp. 57–76.

[109] Yusuke Naito, Mitsuru Matsui, Yasuyuki Sakai, Daisuke Suzuki, Kazuo Sakiyama, and Takeshi Sugawara. SAEAS. In: *NIST Lightweight Cryptography Project* (2019).

[110] Ivica Nikolić. Tiaoxin-346: VERSION 2.0. CAESAR Competition. 2014.

[111] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In: *ICICS 06: 8th International Conference on Information and Communication Security*. Ed. by Peng Ning, Sihan Qing, and Ninghui Li. Vol. 4307. Lecture Notes in Computer Science. Raleigh, NC, USA: Springer, Heidelberg, Germany, Dec. 2006, pp. 529–545.

[112] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-Channel Resistant Crypto for Less than 2,300 GE. In: *Journal of Cryptology* 24.2 (Apr. 2011), pp. 322–345.

[113] Georgios Pouiklis and Georgios Ch. Sirakoulis. Clock Gating Methodologies and Tools: A Survey. In: *Int. J. Circuit Theory Appl.* 44.4 (Apr. 2016), pp. 798–816.

[114] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In: *Advances in Cryptology – CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2015, pp. 764–783.

[115] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. Smashing the Implementation Records of AES S-box. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.2 (2018), pp. 298–336.

[116] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An Efficient AES-based Encryption Scheme for Beyond 5G. In: *IACR Transactions on Symmetric Cryptology* 2021.2 (2021), pp. 1–30.

[117] Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In: *Cryptographic Hardware and Embedded Systems – CHES 2015*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. Lecture Notes in Computer Science. Saint-Malo, France: Springer, Heidelberg, Germany, Sept. 2015, pp. 495–513.

[118] Peter Schwabe and Ko Stoffelen. All the AES You Need on Cortex-M3 and M4. In: *SAC 2016: 23rd Annual International Workshop on Selected Areas in Cryptography*. Ed. by Roberto Avanzi and Howard M. Heys. Vol. 10532. Lecture Notes in Computer Science. St. John's, NL, Canada: Springer, Heidelberg, Germany, Aug. 2016, pp. 180–194.

[119] Adi Shamir. How to Share a Secret. In: *Communications of the Association for Computing Machinery* 22.11 (Nov. 1979), pp. 612–613.

[120] The ZUC Design Team. The ZUC-256 Stream Cipher. `http://www.is.cas.cn/ztzl2016/zouchongzhi/201801/W020180126529970733243.pdf`. 2018.

[121] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, Apr. 2015, pp. 287–314.

[122] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube Attacks on Non-Blackbox Polynomials Based on Division Property. In: *Advances in Cryptology – CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2017, pp. 250–279.

[123] Yosuke Todo, Willi Meier, and Kazumaro Aoki. On the Data Limitation of Small-State Stream Ciphers: Correlation Attacks on Fruit-80 and Plantlet. In: *SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Kenneth G. Paterson and Douglas Stebila. Vol. 11959. Lecture Notes in Computer Science. Waterloo, ON, Canada: Springer, Heidelberg, Germany, Aug. 2019, pp. 365–392.

[124] Yosuke Todo and Masakatu Morii. Bit-Based Division Property and Application to Simon Family. In: *Fast Software Encryption – FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. Lecture Notes in Computer Science. Bochum, Germany: Springer, Heidelberg, Germany, Mar. 2016, pp. 357–377.

[125] Qingju Wang, Yonglin Hao, Yosuke Todo, Chaoyun Li, Takanori Isobe, and Willi Meier. Improved Division Property Based Cube Attacks Exploiting Algebraic Properties of Superpoly. In: *Advances in Cryptology – CRYPTO 2018, Part I*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10991. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2018, pp. 275–305.

[126] Hongjun Wu and Bart Preneel. AEGIS: A Fast Authenticated Encryption Algorithm. In: *SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Tanja Lange, Kristin Lauter, and Petr Lisonek. Vol. 8282. Lecture Notes in Computer Science. Burnaby, BC, Canada: Springer, Heidelberg, Germany, Aug. 2014, pp. 185–201.

[127] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, Heidelberg, Germany, Dec. 2016, pp. 648–678.

[128] Bin Zhang, Xinxin Gong, and Willi Meier. Fast Correlation Attacks on Grain-like Small State Stream Ciphers. In: *IACR Transactions on Symmetric Cryptology* 2017.4 (2017), pp. 58–81.

Appendix

A AEAD Energy Analysis

A.1 NanGate 45 nm and UMC 65 nm Synthesis Results

Table 1: Synthesis measurements of the investigated AEAD schemes for the NanGate 45 nm cell library. The energy consumption corresponds to the encryption of 1024 bits of AD and 1024 plaintext bits.

Cipher	Circuit	Area	Latency	Critical Path	TP	Power	Energy
		GE	Cycles	ns	Mbits/s	μW	nJ
GIFT-COFB	1-Round	5395	680	2.18	1467.9	145.6	9.90
	1-Round-CG	5248	680	2.20	1454.5	140.9	9.58
	2-Round	6049	340	2.49	2570.3	170.2	5.79
	2-Round-CG	5902	340	2.56	2500.0	165.4	5.62
	3-Round	7010	238	3.36	2721.1	201.2	4.79
	3-Round-CG	6862	238	3.47	2634.8	198.2	4.72
	4-Round	7358	170	3.40	3764.7	219.3	3.73
	4-Round-CG	7212	170	3.46	3699.4	214.5	3.65
SUNDAE-GIFT	1-Round	4284	1000	1.99	804.0	126.5	12.65
	1-Round-CG	-	-	-	-	-	-
	2-Round	4939	500	2.45	1306.1	150.5	7.53
	2-Round-CG	-	-	-	-	-	-
	3-Round	5898	350	3.05	1498.8	182.6	6.39
	3-Round-CG	-	-	-	-	-	-
	4-Round	6248	250	3.37	1899.1	198.2	4.95
	4-Round-CG	-	-	-	-	-	-
HyENA	1-Round	4711	680	2.05	1561.0	126.7	8.61
	1-Round-CG	4563	680	1.99	1608.0	122.0	8.30
	2-Round	5365	340	2.75	2327.3	150.9	5.13
	2-Round-CG	5217	340	2.80	2285.7	146.3	4.97
	3-Round	6323	238	2.96	3088.8	183.3	4.36
	3-Round-CG	6175	238	2.98	3068.1	178.6	4.25
	4-Round	6674	170	3.40	3764.7	199.4	3.39

	4-Round-CG	6526	170	3.49	3667.6	194.7	3.31
LOTUS-AEAD	1-Round	8279	1428	1.60	2000.0	196.8	28.10
	1-Round-CG	7544	1428	1.52	2105.3	172.3	24.60
	2-Round	8679	714	2.16	2963.0	196.8	14.05
	2-Round-CG	7942	714	2.19	2922.4	181.7	12.97
	3-Round	9228	485	2.46	3716.6	231.7	11.24
	3-Round-CG	8743	485	2.47	3706.1	208.1	10.09
	4-Round	9474	357	3.00	4266.7	241.6	8.62
	4-Round-CG	8749	357	2.94	4353.7	217.7	7.77
SKINNY-AEAD	1-Round	9326	952	2.27	1409.7	229.6	21.86
	1-Round-CG	8736	952	2.04	1572.5	210.5	20.04
	2-Round	9972	476	3.03	2112.2	256.4	12.20
	2-Round-CG	9383	476	2.93	2184.3	237.2	11.29
	3-Round	11337	323	3.99	2291.4	301.6	9.74
	3-Round-CG	10749	323	3.71	2464.4	282.4	9.12
	4-Round	11782	238	4.75	2694.7	326.7	7.78
	4-Round-CG	11193	238	4.59	2788.7	306.1	7.29
Romulus	1-Round	6892	970	1.55	2064.5	189.5	18.38
	1-Round-CG	-	-	-	-	-	-
	2-Round	7321	494	2.37	2700.4	211.0	10.42
	2-Round-CG	-	-	-	-	-	-
	3-Round	9211	324	3.47	2634.8	268.9	8.71
	3-Round-CG	-	-	-	-	-	-
	4-Round	9206	256	4.09	3129.6	278.3	7.12
	4-Round-CG	-	-	-	-	-	-
ForkAE	1-Round	8543	1144	1.78	1797.8	218.6	25.01
	1-Round-CG	7954	1144	1.80	1777.8	200.0	22.88
	2-Round	9258	576	2.65	2415.1	249.7	14.38
	2-Round-CG	8545	576	2.69	2379.2	231.1	13.31
	3-Round	10861	384	3.88	2356.4	299.2	11.49
	3-Round-CG	10272	384	3.92	2332.4	280.6	10.77
	4-Round	10749	288	4.27	2997.7	304.4	8.77
	4-Round-CG	10160	288	4.29	2983.7	285.7	8.23
Pyjamask	1-Round	18068	285	1.93	4421.4	501.2	14.28
	1-Round-CG	17246	285	2.03	4203.6	467.8	13.33
	2-Round	23076	152	2.64	6060.6	687.1	10.44
	2-Round-CG	22299	152	2.68	5970.1	658.6	10.01
	3-Round	28313	95	2.75	9309.1	888.9	8.44
	3-Round-CG	27482	95	2.78	9208.6	855.7	8.13
	4-Round	33406	76	3.61	8864.3	1077.8	8.19
	4-Round-CG	32568	76	3.63	8815.4	988.9	7.52
Saturnin	1-Round	16981	285	2.05	4162.6	464.8	13.25
	1-Round-CG	15873	285	2.08	4102.6	430.1	12.26
	2-Round	21994	152	1.24	12903.2	602.2	9.15
	2-Round-CG	20892	152	1.31	12213.7	567.5	8.63

	3-Round	-	-	-	-	-	-
	3-Round-CG	-	-	-	-	-	-
	4-Round	23887	76	3.08	10389.6	730.6	5.55
	4-Round-CG	22805	76	3.11	10289.4	530.6	4.03

Table 2: Synthesis measurements of the investigated AEAD schemes for the UMC 65 nm cell library. The energy consumption corresponds to the encryption of 1024 bits of AD and 1024 plaintext bits.

Cipher	Circuit	Area	Latency	Critical Path	TP	Power	Energy
		GE	Cycles	ns	Mbits/s	μW	nJ
GIFT-COFB	1-Round	5155	680	7.24	442.0	35.9	2.44
	1-Round-CG	5147	680	7.83	408.7	34.1	2.32
	2-Round	5805	340	10.23	625.6	54.8	1.86
	2-Round-CG	5767	340	10.72	597.0	52.3	1.78
	3-Round	6675	238	11.16	819.3	84.0	2.00
	3-Round-CG	6573	238	10.80	846.6	82.3	1.96
	4-Round	7279	170	15.66	817.4	131.8	2.24
	4-Round-CG	7285	170	16.54	773.9	131.1	2.23
SUNDAE-GIFT	1-Round	3701	1000	4.55	351.6	32.2	3.22
	1-Round-CG	-	-	-	-	-	-
	2-Round	4482	500	7.79	410.8	59.1	2.95
	2-Round-CG	-	-	-	-	-	-
	3-Round	5390	350	9.40	486.3	82.3	2.88
	3-Round-CG	-	-	-	-	-	-
	4-Round	6006	250	12.63	506.7	136.8	3.42
	4-Round-CG	-	-	-	-	-	-
HyENA	1-Round	4300	680	6.70	477.6	31.9	2.17
	1-Round-CG	4176	680	6.79	471.3	29.2	1.99
	2-Round	4921	340	9.22	694.1	52.1	1.77
	2-Round-CG	4799	340	9.40	680.9	48.8	1.66
	3-Round	6029	238	10.86	841.9	86.5	2.06
	3-Round-CG	5909	238	11.29	809.8	83.8	1.99
	4-Round	6554	170	15.02	852.2	130.5	2.22
	4-Round-CG	6459	170	15.44	829.0	124.3	2.11
LOTUS-AEAD	1-Round	6842	1428	8.87	360.8	31.4	4.48
	1-Round-CG	6733	1428	7.71	415.0	24.1	3.44
	2-Round	7266	714	10.06	636.2	46.3	3.31
	2-Round-CG	6875	714	10.42	614.2	38.2	2.73
	3-Round	7742	485	11.81	774.2	68.7	3.33
	3-Round-CG	7405	485	11.45	798.5	51.4	2.49
	4-Round	8165	357	12.85	996.1	91.8	3.28
	4-Round-CG	7781	357	15.30	836.6	80.8	2.88
SKINNY-AEAD	1-Round	8274	952	7.93	403.5	65.4	6.22

	1-Round-CG	7644	952	7.47	428.4	54.5	5.19
	2-Round	8938	476	10.07	635.6	92.0	4.38
	2-Round-CG	8421	476	9.88	647.8	80.3	3.82
	3-Round	11090	323	14.70	622.0	150.5	4.86
	3-Round-CG	10772	323	15.06	607.1	143.2	4.63
	4-Round	12847	238	19.33	662.2	286.7	6.82
	4-Round-CG	12360	238	20.95	611.0	281.2	6.69
Romulus	1-Round	6167	970	5.23	611.9	50.4	4.89
	1-Round-CG	-	-	-	-	-	-
	2-Round	6654	494	8.85	723.2	69.3	3.42
	2-Round-CG	-	-	-	-	-	-
	3-Round	9185	324	16.68	548.1	137.1	4.44
	3-Round-CG	-	-	-	-	-	-
	4-Round	10569	256	22.54	567.9	261.2	6.69
	4-Round-CG	-	-	-	-	-	-
ForkAE	1-Round	7646	1144	6.30	507.9	61.3	7.01
	1-Round-CG	7238	1144	6.35	503.9	51.9	5.94
	2-Round	8806	576	9.39	681.6	89.1	5.13
	2-Round-CG	7876	576	9.87	648.4	75.4	4.34
	3-Round	10351	384	12.52	730.3	145.7	5.59
	3-Round-CG	9406	384	12.95	706.0	128.7	4.94
	4-Round	12560	288	19.57	654.1	296.6	8.54
	4-Round-CG	11649	288	19.87	644.2	273.2	7.87
Pyjamask	1-Round	16413	285	7.22	1181.9	106.6	3.04
	1-Round-CG	15521	285	7.31	1167.4	87.7	2.50
	2-Round	22932	152	11.62	1376.9	337.5	5.13
	2-Round-CG	22169	152	11.78	1358.2	316.3	4.81
	3-Round	29773	95	13.39	1911.9	528.9	5.02
	3-Round-CG	29174	95	13.47	1900.5	501.4	4.76
	4-Round	37427	76	16.43	1947.7	807.9	6.14
	4-Round-CG	36929	76	16.79	1905.9	764.7	5.81
Saturnin	1-Round	15760	285	9.47	901.1	208.3	5.94
	1-Round-CG	14842	285	9.80	870.7	168.8	4.81
	2-Round	20920	152	6.27	2551.8	259.5	3.94
	2-Round-CG	20031	152	6.35	2519.7	244.5	3.72
	3-Round	-	-	-	-	-	-
	3-Round-CG	-	-	-	-	-	-
	4-Round	23095	76	13.21	2422.4	485.6	3.69
	4-Round-CG	22158	76	13.45	2379.2	480.1	3.65

A.2 NanGate 45 nm and UMC 65 nm TI Synthesis Results

Table 3: Synthesis figures the investigated threshold implementations for the NanGate 45 nm and UMC 65 nm cell libraries. The latency/energy consumption corresponds to the encryption of 1024 bits of AD and 1024 plaintext bits.

Library	Scheme	Shares	Area	Latency	Critical Path	TP	Power	Energy
			GE	Cycles	ns	Mbits/s	μW	nJ
NanGate 45 nm	GIFT-COFB	3	18034	1360	1.97	812.2	456.3	62.05
	GIFT-COFB	4	24932	680	2.79	1147.0	961.3	65.37
	SUNDAE-GIFT	3	16045	2000	1.65	484.8	431.2	86.24
	SUNDAE-GIFT	4	26706	1000	2.41	663.9	920.3	92.03
	HyENA	3	17213	1360	2.16	740.7	440.2	59.86
	HyENA	4	29302	680	2.80	1142.9	934.8	63.57
	LOTUS-AEAD	3	17966	2856	1.37	1167.9	432.0	123.38
	LOTUS-AEAD	4	24030	1428	2.15	1488.4	682.1	97.40
	SKINNY-AEAD	3	22543	3808	2.11	379.1	561.8	212.92
	Romulus	3	15970	3877	1.97	406.1	460.4	178.48
	ForkAE	3	21139	4576	1.92	416.7	552.5	252.83
UMC 65 nm	GIFT-COFB	3	18606	1360	6.92	231.2	92.6	12.59
	GIFT-COFB	4	29049	680	8.35	383.2	210.0	14.28
	SUNDAE-GIFT	3	16968	2000	7.97	100.4	94.7	18.93
	SUNDAE-GIFT	4	25726	1000	8.46	189.1	217.1	21.71
	HyENA	3	17392	1360	6.76	236.7	90.7	12.33
	HyENA	4	27264	680	8.27	386.9	200.2	13.61
	LOTUS-AEAD	3	16664	2856	7.19	222.5	61.5	17.55
	LOTUS-AEAD	4	22624	1428	8.02	399.0	129.0	18.42
	SKINNY-AEAD	3	22365	3808	8.07	102.2	65.4	24.90
	Romulus	3	17251	3877	5.38	148.7	109.2	42.34
	ForkAE	3	21751	4576	5.78	138.4	139.9	64.02

B Perfect Trees

B.1 Proof of Lemma 1

In the following, we prove the lemma by the means of induction.

Base Case. Consider $t_1(r)$ in the original Trivium. We know that for $r = 1 \rightarrow 66 (= X_1^1)$, $t_1(r)$ can be written as functions of depth 0 nodes of the circuit i.e., the state variables $x_1, x_2, x_3, \ldots, x_{288}$, and it is easy to see that all $t_1(r)$, $r \in [1,66]$ are perfect depth 1 trees. For $r = 67$, $t_1(r)$ is expanded as $t_3(1) + x_{27} + x_{28} \cdot x_{29} + x_{105}$. Note that $t_3(1)$ is no longer a depth 0 node, and hence $t_1(67)$ is not a perfect tree. Also consider a sightly modified form of Trivium in which $X_2^f = 62$ (say). In this case the recursive definition of $t_1(r)$ is as follows:

$$t_1(r) = t_3(r-66) + t_3(r-93) + [t_3(r-91) \cdot t_3(r-92)] + t_1(r-62)$$

Now it is easy to see that $t_1(r)$ is a perfect depth 1 tree only upto $r = 62$, as $t_1(63)$ will involve a $t_1(1)$ term which is no longer at depth 0. Thus the number of perfect depth 1 trees for $t_1(r)$ in a generalized Trivium circuit has to be the smaller of 66 and 62, i.e., $\min\left\{X_1^\ell, X_2^f\right\}$. Does this also depend on the tap position of the two AND gates and the final XOR term $t_3(r-93)$? The final XOR term must be tapped from the final location of each register to ensure that the state update function is one-to-one. So numerically, X_j^{op} has to be the length of the register X_j. Since X_j^ℓ is an intermediate location and X_j^{op} is the final location of register j, we always have $X_j^\ell < X_j^{op}$. If we select the tap locations for the AND gates in the range (X_j^ℓ, X_j^{op}), it is easy to see that the perfect depth 1 trees only occur till the smaller of X_1^ℓ and X_2^f. Even if the the tap location of one or both inputs to the AND gate is less than X_j^1, we can simply select the numerically smallest tap location of register X_j as X_j^ℓ, since in terms of the circuit graph it does not make a difference if X_j^ℓ is input to an XOR or an AND gate. However, here we have the AND taps strictly in between X_j^ℓ and X_j^{op} and so the the actual locations do of the AND taps not make a difference. Thus it is pretty easy to see base case for our recursive formula $f_1(X_j) = 0$ and $g_1(X_j) = \min\left\{X_j^\ell, X_{j+1}^f\right\}$.

Inductive Step. Let us assume the inductive hypothesis; g_l, f_l are as defined in the Lemma statement for $t = 1, 2, 3, \ldots, l-1$. Consider the equation for $t_1(r)$ at $r = r_0 = f_{l-1}(X_3) + X_1^{op}$ and $r = r_0 + 1$. For conciseness, denote by the symbol α the value of $f_{l-1}(X_3)$ and $\Delta = X_1^{op} - X_1^\ell$. It holds (note a_1, a_2 are the AND gate taps with $X_1^\ell < a_1, a_2 < X_1^{op}$)

$$\begin{aligned}
t_1(r_0) &= t_3(r_0 - X_1^\ell) + t_3(r_0 - X_1^{op}) + [t_3(r_0 - a_1) \cdot t_3(r_0 - a_2)] + t_1(r_0 - X_2^f) \\
&= t_3(\alpha + \Delta) + t_3(\alpha) + \left[t_3(\alpha + (X_1^{op} - a_1)) \cdot t_3(\alpha + (X_1^{op} - a_2))\right] \\
&\quad + t_1(\alpha + (X_1^{op} - X_2^f)) \\
t_1(r_0 + 1) &= t_3(\alpha + \Delta + 1) + t_3(\alpha + 1) \\
&\quad + \left[t_3(\alpha + 1 + (X_1^{op} - a_1)) \cdot t_3(\alpha + 1 + (X_1^{op} - a_2))\right] + t_1(\alpha + 1 + (X_1^{op} - X_2^f))
\end{aligned}$$

Note that by the inductive hypothesis, $t_3(\alpha)$ corresponds to a depth $l-2$ tree, whereas

$t_3(\alpha+1)$ corresponds to a depth $l-1$ tree. All other t_3 terms in the above expressions are depth $l-1$ trees or greater by the inductive hypothesis. If $t_1(\alpha+(X_1^{op}-X_2^f))$ also corresponds to a depth $l-1$ tree, it is easy to see that r_0+1 is the first value of r for which $t_1(r)$ produces a perfect depth l tree. However that is always not the case. It may so happen that $t_1(\alpha+(X_1^{op}-X_2^f))$ still corresponds to a depth $l-2$ tree for certain specific instances of the generic Trivium circuit. In such cases the value of r has to be equal to $u=f_{l-1}(X_1)+X_2^f+1$ to ensure that the t_1 term in the expression for $t_1(r)$ also produces a depth $l-1$ tree by the inductive hypothesis. This is true since $t_1(u-X_2^f)=t_1(f_{l-1}(X_1)+1)$ which corresponds to a depth $l-1$ tree by the inductive hypothesis.

For $t_1(r)$ to definitely correspond to a depth l perfect tree both the depth conditions on the above t_3 and t_1 nodes must be satisfied. This leads us to the easy conclusion that the first value of r for which $t_1(r)$ is a perfect depth r tree is the maximum of $f_{l-1}(X_3)+X_1^{op}+1$ and $f_{l-1}(X_1)+X_2^f+1$. Generalizing over all configurations of n-stage registers, we have

$$f_l(X_j)=\max\left\{f_{l-1}(X_{j-1})+X_j^{op},\ f_{l-1}(X_j)+X_{j+1}^f\right\}.$$

In order to prove the recursive expression for g_l, consider again $t_1(r)$ for the generic Trivium circuit for $r=r_1=g_{l-1}(X_3)+X_1^\ell$ and $r=r_1+1$. For conciseness, denote by the symbol β the value of $g_{l-1}(X_3)$. It holds

$$\begin{aligned}
t_1(r_1) &= t_3(r_1-X_1^\ell)+t_3(r_1-X_1^{op})+[t_3(r_1-a_1)\cdot t_3(r_1-a_2)]+t_1(r_1-X_2^f)\\
&= t_3(\beta)+t_3(\beta-\Delta)+\left[t_3(\beta-\Delta+(X_1^{op}-a_1))\cdot t_3(\beta-\Delta+(X_1^{op}-a_1))\right]\\
&\quad + t_1(\beta-\Delta+(X_1^{op}-X_2^f))\\
t_1(r_1+1) &= t_3(\beta+1)+t_3(\beta+1-\Delta)\\
&\quad +\left[t_3(\beta+1-\Delta+(X_1^{op}-a_1))\cdot t_3(\beta+1-\Delta+(X_1^{op}-a_2))\right]\\
&\quad + t_1(\beta+1-\Delta+(X_1^{op}-X_2^f))
\end{aligned}$$

By the inductive hypothesis $t_3(\beta+1)$, no longer corresponds to a perfect tree of depth $l-1$. Assuming that $t_3(\beta-\Delta)$, $t_3(\beta-\Delta+(X_1^{op}-a_1))$, $t_3(\beta-\Delta+(X_1^{op}-a_1))$ and $t_1(\beta-\Delta+(X_1^{op}-X_2^f))$ do correspond to perfect depth $l-1$ trees, $r_1=g_{l-1}(X_3)+X_1^\ell$ is of course the largest value of r that produces depth l trees.

There are two assumptions made in the above proof which may not always hold for all configurations of the generic Trivium circuit. The first is if $t_3(\beta-\Delta)$ does not correspond to a perfect depth $l-1$ tree (note if $t_3(\beta-\Delta)$ is not a perfect depth $l-1$ tree, neither of the AND terms will correspond to depth $l-1$ trees since their index values are larger than $\beta-\Delta$). The above happens when

$$\beta-\Delta\le f_{l-1}(X_3)\Rightarrow g_{l-1}(X_3)-f_{l-1}(X_3)-(X_1^{op}-X_1^\ell)\le 0.$$

The above condition essentially means that the number of perfect depth $l-1$ trees for $t_3(r)$ is less than or equal to $X_1^{op}-X_1^\ell$. This implies that the terms $t_3(r-X_1^\ell)$ and $t_3(r-X_1^{op})$ can never be both of depth $l-1$, which in turn implies that the expression for $t_1(r)$ can never produce

a depth l tree. In this case we can simply set $g_l(X_1)$ to be some value less than or equal to $f_l(X_1)$ to indicate this impossibility. We can simply pick one of the expressions for $f_l(X_1)$, i.e., $f_{l-1}(X_3) + X_1^{op}$ for this purpose. Combining the two assumptions we can write the new expression as $f_{l-1}(X_3) + X_1^{op} + \left[\left(g_{l-1}(X_3) - f_{l-1}(X_3)\right) - (X_1^{op} - X_1^{\ell})\right]^+$ The second assumption was that t_1 term also produces a perfect depth $l-1$ tree. For a generic Trivium circuit, this assumption may be false. The term $t_1(r - X_2^f)$ will produce a depth $l-1$ tree if $r - X_2^f \leq g_{l-1}(X_1) \Rightarrow r \leq X_2^f + g_{l-1}(X_1)$. Since we need both depth conditions to be satisfied, we take minimum of the above two. Generalizing for all n-stage Trivium circuits we have

$$g_l(X_j) = \min\Big\{ f_{l-1}(X_{j-1}) + X_j^{op} + \Big[\big(g_{l-1}(X_{j-1}) - f_{l-1}(X_{j-1})\big) - (X_j^{op} - X_j^{\ell})\Big]^+, \\ g_{l-1}(X_j) + X_{j+1}^f \Big\}.$$

B.2 Trivium-LE(S) Security Analysis

We inherit the security of Trivium-LE(F) against the TMD tradeoff attack because it does not exploit the tap location. Here, we discuss the security against the correlation attack, the Maximov/Biryukov's guess-and-determine attack, and the cube attack.

Linear Distinguishing Attack. On this parameter, the maximum linear correlation is 2^{-48}, and the required data is about 2^{96} to distinguish the keystream from ideal one. Compared to 2^{144} in Trivium or Trivium-LE(F), the security margin is very narrow. However, it is still enough to achieve the claimed security, i.e., 80 bits, which is the same as Trivium.

Maximov/Biryukov's Guess-and-Determine Attack. Similarly to the case for Trivium-LE(F), we evaluated the number of collectable linear equations after guessing some outputs of AND gates. In the scenario T1, the time complexity is $c \cdot 2^{77.0503}$, where 37, 41, and 44 outputs of AND gates are guessed for each register. Considering $c \approx 2^{16}$, this attack never threatens the claimed security, namely 80 bits.

Cube Attack. We also investigated the increase in algebraic degree by using the bit-based division property. Figure 1 shows the upper bound of the algebraic degree of $f_k(iv)$. Trivium-LE(S) is clearly more vulnerable than Trivium and Trivium-LE(F) against the cube attack. The degree of even 1000 rounds does not reach 80. In other words, we can attack 1000-round Trivium-LE(S) with the use of any 76-dimensional cube. The upper bound reaches the full, i.e., 80, in 1050 rounds although the original Trivium reach the same level in 840 rounds. In other words, the increase of the degree is about 25% slower because $1050/840 = 1.25$. This is the main reason why we increase the number of rounds in the initialization from $288 \times 4 = 1152$ to $288 \times 5 = 1440$.

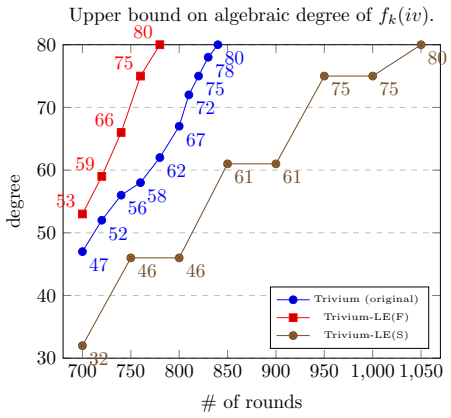
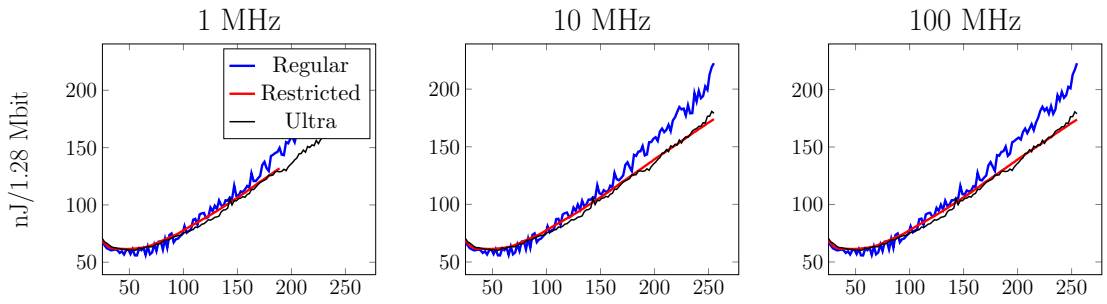
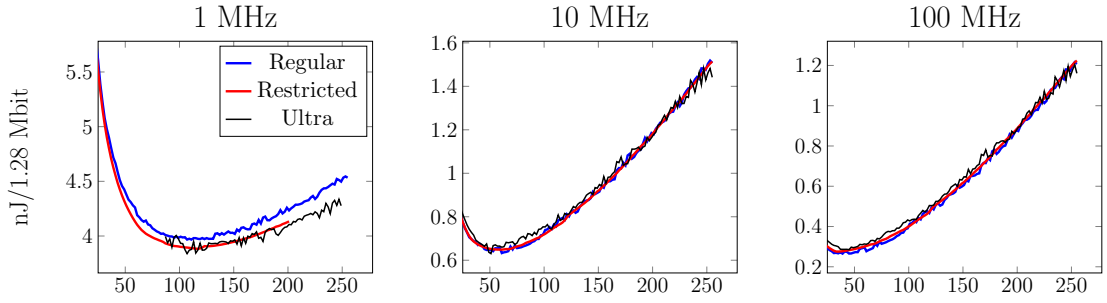


Figure 1: Increase in algebraic degree with respect to the number of initialization rounds.

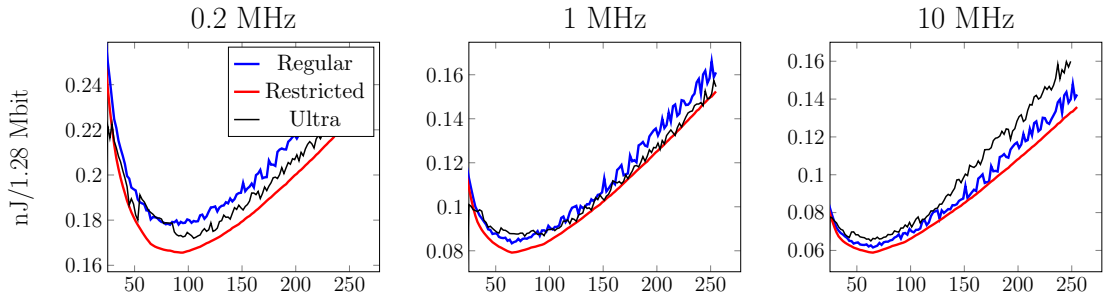
B.3 Supplementary Grain-128 Plots



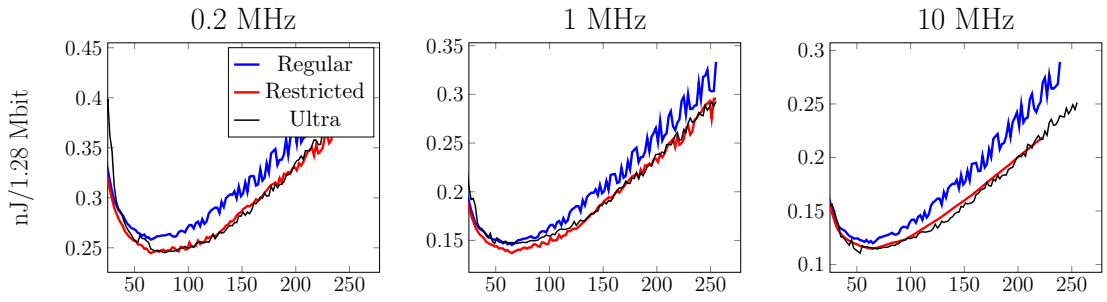
(a) NanGate 15 nm



(b) NanGate 45 nm

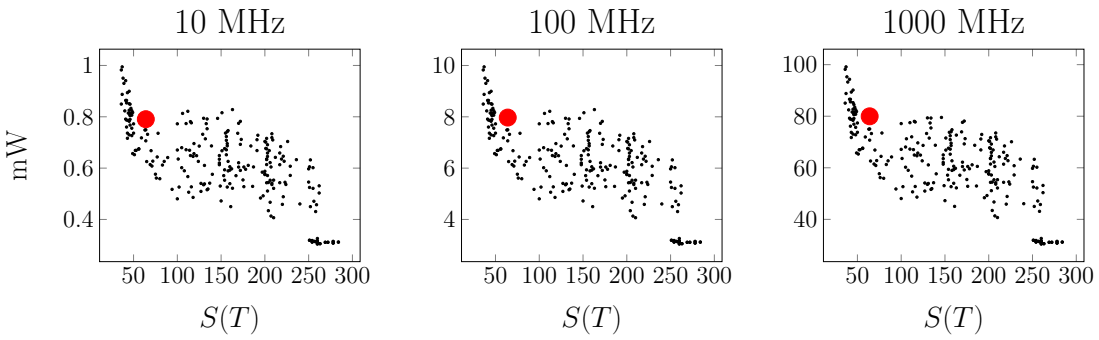


(c) UMC 65 nm

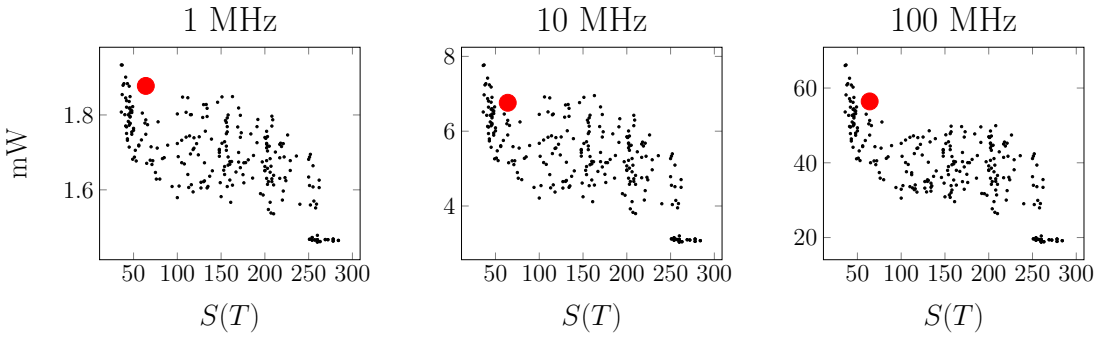


(d) TSMC 90 nm

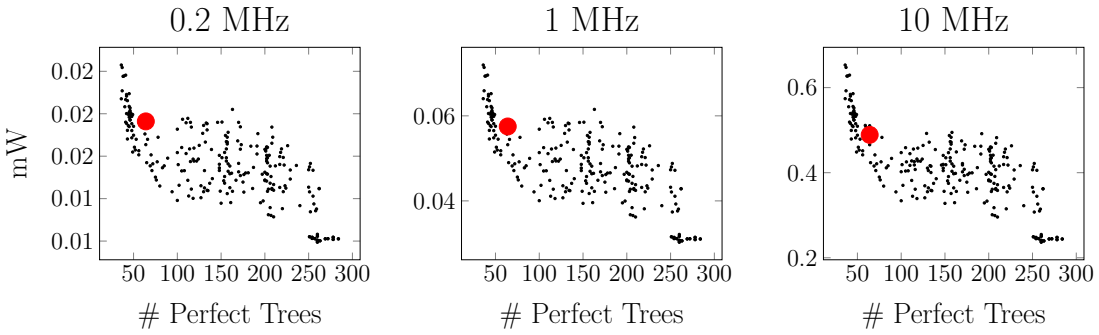
Figure 2: Grain-128 energy measurements for the three synthesis settings for different frequencies and libraries. Note that energy graphs are noisier for the regular/ultra modes which indicates that the synthesizer chooses different mapping strategies for varying r.



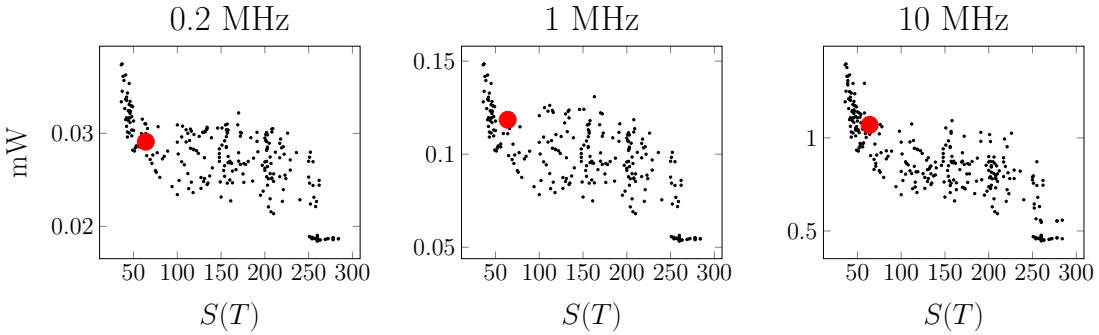
(a) NanGate 15 nm



(b) NanGate 45 nm



(c) UMC 65 nm



(d) TSMC 90 nm

Figure 3: Power measurements of several Grain-128 circuits as a function of $S(T)$.

C Atom

C.1 Banik's Distinguishing Attack on Sprout

Atom is secure against generic Time Memory Data (TMD) Tradeoff attacks as shown in [37], for the same reason that Sprout, Plantlet and Lizard are secure. The reason is that it is not possible to construct a one way function that maps the internal state to any keystream vector that does not additionally require the secret key. Furthermore the key update component in the state update function is completely linear, this ensures that table based special state attacks of [64] do not apply to all post-Sprout constructions. An interesting distinguishing attack against Sprout using slid keystreams was presented in [10] that also applies to Plantlet and Lizard. We will present the attack in context of Atom.

Consider any random initial state $S_R \in \{0,1\}^{159}$. Since the state update function in both the keystream generation and key-IV initialization is bijective and efficiently invertible, we can apply both the Init^{-1} and Update^{-1} algorithms on it. Given the secret key, the former would reverse the entire key-IV initialization on any random string of 159 bits, and the latter inverts one round of the state update during keystream generation. A state S_R is a valid initial state after key-IV initialization, if a) its last 9 bits in decimal representation equals 511 and b) if $\text{Init}^{-1}(S_R)$ has the 22 bit constant used to initialize Atom in bit positions 128 to 149. Thus the probability that a random S_R is a valid initial state is around $2^{-22-9} = 2^{-31}$. Similarly the probability that S_R is a valid t^{th} state after initialization is also 2^{-31} ($S = [\text{Update}^{-1}]^t(S_R)$ and $\text{Init}^{-1}(S)$ must satisfy the required conditions). Hence the probability that for any given key S_R is both the 0^{th} and t^{th} post-initialization state for 2 different IVs is around 2^{-62}. From randomness considerations we can therefore conclude that on average for every key there exists $2^{159-62} = 2^{97}$ IV pairs IV_1, IV_2 that satisfy such a condition. If t is such that the order and sequence of keybits that is used in the state update following the 0-th and t-th clocks are the same, then it is clear that the IV pair IV_1, IV_2 produce t-bit shifted keystream for the given key. So our distinguisher is as follows:

- Generate around $2t$ keystream bits $Z_1||Z_2$ for the unknown Key K and some randomly generated Initial Vector IV (where Z_1 and Z_2 are t-bit vectors each).
- Store the keystream bits in some appropriate data structure such as a hash table keyed with both Z_1 and Z_2 (to help easy detection of collisions).
- Continue the above steps with more randomly generated IVs IV till we obtain two Initial Vectors for K that generate t-bit shifted keystream.

Imagine the space of Initial Vectors as an undirected Graph $G = (W, E)$, where $W = \{0,1\}^{128}$ is the Vertex set which contains all the possible 128 bit Initial vector values as nodes. An edge $(\text{IV}_1, \text{IV}_2) \in E$ if and only if (K, IV_1) and (K, IV_2) produce t-bit shifted keystream sequence. From the above discussion, it is clear that the cardinality of E is expected to be 2^{97}. When we run the Distinguisher algorithm for N different Initial Vectors, we effectively add $\binom{N}{2}$ edges to the coverage and a match occurs when one of these edges is actually a member of the Edge-set E. Since there are potentially $\binom{2^{128}}{2}$ edges in the IV space, by the Birthday bound, a match

will occur when the product of $\binom{N}{2}$ and the cardinality of E which is around 2^{97} is equal to $\binom{2^{128}}{2}$. From this equation solving for N, we get $N \approx 2^{79.5} = \sqrt{2^{159}}$ which is square root of the cardinality of the state space. This gives a bound for the time and memory complexity of the Distinguisher. The time complexity is around $\sqrt{2^{159}}$ encryptions, and the memory required is of the order of $2t \cdot \sqrt{2^{159}}$ bits.

This keystream distinguisher also works for Atom, but we claim that this can not be converted to a key-recovery attack. Consider what happens when the attacker finds two IVs $\mathrm{IV}_1, \mathrm{IV}_2$ that produces 128-bit shifted keystream for some secret key K. This implies that there exists a state S_R which is the 0^{th} and 128^{th} post-initialization state after initialization with key-IV pairs (K, IV_1) and (K, IV_2) respectively. This implies the following two things

- S_R and $[\mathrm{Update}^{-1}]^{128}(S_R)$ are such that the last 9 LFSR bits of both these states is the 9 bit string 1^9.
- $\mathrm{Init}^{-1} S_R$ and $\mathrm{Init}^{-1} \circ [\mathrm{Update}^{-1}]^{128}(S_R)$ are such that the last 31 LFSR bits of both these states is the 31-bit constant used to initialize Atom.

Of these, the latter is not of much use cryptographically, since Init^{-1} is an algebraically complex function, most probably of degree close to $(128 + 159)$. However, Update^{-1} is a linear function on the LFSR part of the state. It implies that the S_R can be denoted as the symbolic variable string $\ell_0, \ell_1, \ldots, \ell_{59}, 1^9$ over GF(2). Furthermore, there is a set of 9 linear equations over the 60 variables ℓ_i. The kernel of this system has dimension 51, which implies that there are 2^{51} possible values that the LFSR part of S_R can have. Hereafter, the attacker may use the equation solving approach used in the previous subsection to solve for the NFSR state and the key. The only difference is that the attacker now has fewer number of LFSR states to try out. This implies that the total complexity required for this approach is faster than the attack complexities from Section 5.3.5 by a factor of 2^9, plus an additive complexity of $2^{79.5}$ required to find the shifted keystreams. This is still worse than exhaustive search and requires memory of $2t \cdot 2^{79.5} \approx 2^{87.5}$ bits.

D Rocca-S

D.1 Finding the Round Function Parameters

In this section, we search for optimal parameters that satisfy the security requirements. Let s, a, and m be #State, #AESENC, and #Message, respectively. Once we select $rate$ and s according to Requirement 1 and Requirement 3, then we can properly choose pairs of a and m by Requirement 2 (see Section 8.2.2). Specifically, we search for all fifteen candidates with parameters such that rate = 2 to 3 and $s = 6, 7, 8$ as shown in Table 4. For each parameter, we try to search for candidates that satisfy Requirement 4 for all patterns of block permutations and the combinations of positions of inserted messages and AESENC/XOR in the target class of Figure 8.2 through a MILP-aided evaluation. Sakamoto et al. estimated the total number of search space as $s! \times \binom{s}{a} \times \binom{s}{m}$ [116]. However, this search space includes equivalent class of round functions. Considering such equivalent classes, we can reduce it by the formula of $\frac{s!}{m!} \times \binom{s}{m} \times \binom{s}{a}$. For example, the candidates of the class of $s = 7$, $a = 4$, and $m = 2$ can be reduced from $2^{21.82}$ in [116], to $2^{20.82}$. In our evaluation, if the total number of candidates in the class exceeds 2^{23}, we randomly choose 2^{20} candidates and evaluate these due to the limitations of the computational power.

Table 4: Candidate of round functions for each class.

#State	#AESENC	#Message	Rate	Total	#Searched	#Found
6	4	2	2.0	$2^{16.31}$	All	0
6	6	3	2.0	$2^{11.23}$	All	0
6	5	2	2.5	$2^{16.98}$	All	0
6	6	2	3.0	$2^{12.40}$	All	0
7	4	2	2.0	$2^{20.82}$	All	0
7	6	3	2.0	$2^{17.65}$	All	0
7	7	3	2.33	$2^{14.84}$	All	0
7	5	2	2.5	$2^{20.08}$	All	0
7	**6**	**2**	**3.0**	$\mathbf{2^{18.50}}$	**All**	**14**
8	4	2	2.0	$2^{25.24}$	2^{20}	0
8	6	3	2.0	$2^{23.33}$	2^{20}	0
8	7	3	2.33	$2^{21.52}$	All	0
8	5	2	2.5	$2^{24.91}$	2^{20}	0
8	8	3	2.67	$2^{18.52}$	All	0
8	**6**	**2**	**3.0**	$\mathbf{2^{23.91}}$	$\mathbf{2^{20}}$	**784**

Table 4 shows the summary of our search. We found 14 candidates in $s = 7$, $a = 6$, and $m = 2$ and 784 candidates in $s = 8$, $a = 6$, and $m = 2$ which satisfy Requirement 4. Due to Requirement 3, we choose 14 candidates of the class of $s = 7$, $a = 6$, and $m = 2$. This evaluation

requires about 45 days on three computers equipped with AMD Ryzen Threadripper 3990X (64-Core) and 256 GB RAMs.

Selecting the Best Round Function for Rocca-S. To determine one round function from 14 candidates of $s = 7$, $a = 6$, and $m = 2$, we evaluate the security and performance of these.

- Table 5 shows the required number of rounds for full diffusion and the lower bound for the number of active S-boxes for forgery setting. We choose seven candidates which attain 46 active S-boxes and achieve the full diffusion after 5 rounds named as RF-1, 2, 3, . . . , 7.

Table 5: The lower bound for the number of differentially active S-boxes and the full diffusion rounds.

# Active S-boxes	Full Diffusion Rounds	# Candidates
44	6	7
46	5	7

- Table 6 shows the security of the initialization phase of these candidates against differential attacks and integral attacks by a byte-based MILP, assuming that the adversary can control only nonce. In addition, Table 6 compares the speed of the round function of 7 candidates and Rocca, where the speed is measured as the average of the round function executed $2^{23.25}$ times with 64 kB messages on Intel(R) Core(TM) i7-1068NG7 CPU @ 2.30GHz.

Table 6: Lower bound of differentially active S-boxes, maximum rounds of the integral distinguisher and speeds.

Target	# Active S-boxes					Integral distinguisher	Speed (cycles / Byte)
	6R	7R	8R	9R	10R		
AEGIS-128L	85	86	94	111	120	6R	0.188985
Tiaoxin-346	53	93	99	123	134	15R	0.200404
Rocca	54	62	82	85	93	7R	0.123258
RF-1 (Rocca-S)	**94**	**113**	**122**	**134**	**152**	**5R**	**0.122219**
RF-2	76	88	103	115	131	6R	0.129443
RF-3	96	101	114	129	136	6R	0.118518
RF-4	80	100	108	120	145	5R	0.122185
RF-5	81	86	95	121	141	5R	0.122286
RF-6	97	113	122	139	151	6R	0.129258
RF-7	97	110	128	132	137	6R	0.129523

Considering results of Table 6, we finally adopt RF-1 as shown in Figure 8.1.

D.2 Auxiliary Round-based and 2-Round Unrolled Synthesis Results

Table 7: Rocca-S circuit area comparison for the NanGate 45 nm and UMC 65 nm cell libraries.

(a) Rocca-S

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 45 nm						
μm^2	81409	87312	41075	45093	40947	107530
GE	102016	109414	51472	56508	51312	134749
UMC 65 nm						
μm^2	146945	162474	78839	86212	78563	199989
GE	102045	112829	54749	59869	54558	138881
2-Round Unrolled						
NanGate 45 nm						
μm^2	151865	163937	71463	79524	71208	204390
GE	190307	205435	89553	99654	89233	256128
UMC 65 nm						
μm^2	275923	306846	139575	154317	139019	381893
GE	191613	213088	96927	107165	96541	265203

(b) AEGIS

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 45 nm						
μm^2	60986	65706	31028	34041	30926	80884
GE	76424	82338	38882	42658	38754	101358
UMC 65 nm						
μm^2	109598	121988	59261	64791	59054	150129
GE	76110	84714	41153	44994	41010	104256
2-Round Unrolled						
NanGate 45 nm						
μm^2	116454	124907	55551	61667	55347	155263
GE	145932	156525	69613	77277	69357	194565
UMC 65 nm						
μm^2	208727	232660	107207	118226	106792	288941
GE	144949	161569	74449	82101	74161	200653

(c) AES-256-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 45 nm						
μm^2	36793	37599	30374	31001	30351	46162
GE	46107	47117	38063	38848	38034	57847
UMC 65 nm						
μm^2	68774	71351	58282	59434	58239	86261
GE	47760	49549	40474	41274	40444	59903
2-Round Unrolled						
NanGate 45 nm						
μm^2	70455	72153	57703	58957	57657	89281
GE	88289	90417	72310	73881	72252	111881
UMC 65 nm						
μm^2	132767	137928	111792	114095	111706	167653
GE	92199	95783	77633	79233	77574	116426

(d) SNOW-V-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 45 nm						
μm^2	53496	54878	43319	44323	43285	59934
GE	67038	68769	54284	55543	54242	75105
UMC 65 nm						
μm^2	97934	102018	81109	82953	81040	111384
GE	68010	70846	56326	57606	56278	77350
2-Round Unrolled						
NanGate 45 nm						
μm^2	96639	99438	76319	78328	76251	109511
GE	121102	121102	95638	98155	95553	137232
UMC 65 nm						
μm^2	179742	187886	146069	149755	145931	206618
GE	124821	130476	101437	103997	101341	143485

Table 8: Rocca-S throughput comparison for the NanGate 45 nm and UMC 65 nm cell libraries.

(a) Rocca-S

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 45 nm						
Critical Path (ns)	1.87	1.82	2.41	2.14	2.36	1.55
Max TP (Tbps)	0.137	0.141	0.106	0.120	0.108	0.165
UMC 65 nm						
Critical Path (ns)	5.53	5.12	6.02	5.37	5.87	4.24
Max TP (Tbps)	0.046	0.050	0.043	0.048	0.044	0.060
2-Round Unrolled						
NanGate 45 nm						
Critical Path (ns)	3.43	3.30	4.45	3.93	4.38	2.73
Max TP (Tbps)	0.149	0.155	0.115	0.130	0.117	0.188
UMC 65 nm						
Critical Path (ns)	10.25	9.52	11.27	9.99	11.02	7.70
Max TP (Tbps)	0.050	0.054	0.045	0.051	0.046	0.066

(b) AEGIS

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 45 nm						
Critical Path (ns)	1.60	1.56	2.16	1.89	2.10	1.35
Max TP (Tbps)	0.080	0.082	0.059	0.068	0.061	0.095
UMC 65 nm						
Critical Path (ns)	5.16	4.71	5.62	4.96	5.46	3.84
Max TP (Tbps)	0.025	0.027	0.023	0.026	0.023	0.033
2-Round Unrolled						
NanGate 45 nm						
Critical Path (ns)	3.22	3.12	4.28	3.75	4.20	2.55
Max TP (Tbps)	0.08	0.082	0.06	0.068	0.061	0.10
UMC 65 nm						
Critical Path (ns)	10.11	9.35	11.09	9.81	10.85	7.53
Max TP (Tbps)	0.025	0.027	0.023	0.026	0.024	0.034

(c) AES-256-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 45 nm						
Critical Path (ns)	2.93	2.93	2.93	2.93	2.93	2.93
Max TP (Tbps)	0.0027	0.0027	0.0027	0.0027	0.0027	0.0027
UMC 65 nm						
Critical Path (ns)	7.78	7.78	7.78	7.78	7.78	7.78
Max TP (Tbps)	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010
2-Round Unrolled						
NanGate 45 nm						
Critical Path (ns)	5.55	5.55	5.55	5.55	5.55	5.55
Max TP (Tbps)	0.0029	0.0029	0.0029	0.0029	0.0029	0.0029
UMC 65 nm						
Critical Path (ns)	13.5	13.5	13.5	13.5	13.5	13.5
Max TP (Tbps)	0.0012	0.0012	0.0012	0.0012	0.0012	0.0012

(d) SNOW-V-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
NanGate 45 nm						
Critical Path (ns)	3.61	3.61	3.61	3.61	3.61	3.61
Max TP (Tbps)	0.035	0.035	0.035	0.035	0.035	0.035
UMC 65 nm						
Critical Path (ns)	9.08	9.08	9.08	9.08	9.08	9.08
Max TP (Tbps)	0.014	0.014	0.014	0.014	0.014	0.014
2-Round Unrolled						
NanGate 45 nm						
Critical Path (ns)	6.01	6.01	6.01	6.01	6.01	6.01
Max TP (Tbps)	0.043	0.043	0.043	0.043	0.043	0.043
UMC 65 nm						
Critical Path (ns)	15.00	15.00	15.00	15.00	15.00	15.00
Max TP (Tbps)	0.017	0.017	0.017	0.017	0.017	0.017

Table 9: Rocca-S power/energy comparison for the NanGate 45 nm and UMC 65 nm cell libraries.

(a) Rocca-S

	LUT	DSE	S	F	T	TT
Round-Based						
Lat. Short (Cycles)	44	44	44	44	44	44
Lat. Long (Cycles)	5036	5036	5036	5036	5036	5036
NanGate 45 nm						
Power (mW)	3.695	2.744	1.746	1.932	1.750	3.579
Energy Short (nJ)	16.26	12.07	7.68	8.50	7.70	15.75
Energy Long (nJ)	1860.8	1381.9	879.3	973.0	881.3	1802.4
UMC 65 nm						
Power (mW)	1.137	0.467	0.882	0.864	0.837	0.417
Energy Short (nJ)	5.00	2.05	3.88	3.80	3.68	1.83
Energy Long (nJ)	572.6	235.2	444.2	435.1	421.5	210.0
2-Round Unrolled						
Lat. Short (Cycles)	22	22	22	22	22	22
Lat. Long (Cycles)	2518	2518	2518	2518	2518	2518
NanGate 45 nm						
Power (mW)	7.061	5.117	3.127	3.507	3.134	6.788
Energy Short (nJ)	15.53	11.26	6.88	7.72	6.89	14.93
Energy Long (nJ)	1778.0	1288.5	787.4	883.1	789.1	1709.2
UMC 65 nm						
Power (mW)	5.597	1.667	4.056	3.906	3.480	1.235
Energy Short (nJ)	12.31	3.67	8.92	8.59	7.66	2.72
Energy Long (nJ)	1409.3	419.8	1021.3	983.5	876.3	311.0

(b) AEGIS

	LUT	DSE	S	F	T	TT
Round-Based						
Lat. Short (Cycles)	48	48	48	48	48	48
Lat. Long (Cycles)	10032	10032	10032	10032	10032	10032
NanGate 45 nm						
Power (mW)	2.343	1.895	1.130	1.245	1.131	2.510
Energy Short (nJ)	11.25	9.096	5.424	5.976	5.429	12.05
Energy Long (nJ)	2350.5	1901.1	1133.6	1249.0	1134.6	2518.0
UMC 65 nm						
Power (mW)	0.355	0.210	0.234	0.253	0.230	0.222
Energy Short (nJ)	1.704	1.008	1.123	1.214	1.104	1.066
Energy Long (nJ)	356.14	210.67	234.75	253.81	230.74	222.71
2-Round Unrolled						
Lat. Short (Cycles)	24	24	24	24	24	24
Lat. Long (Cycles)	5016	5016	5016	5016	5016	5016
NanGate 45 nm						
Power (mW)	5.348	3.879	2.384	2.674	2.388	5.131
Energy Short (nJ)	12.835	9.310	5.722	6.418	5.731	12.31
Energy Long (nJ)	2682.6	1945.7	1195.8	1341.3	1197.8	2573.7
UMC 65 nm						
Power (mW)	3.451	1.136	2.566	2.468	2.392	0.875
Energy Short (nJ)	8.282	2.726	6.158	5.923	5.741	2.100
Energy Long (nJ)	1731.0	569.82	1287.1	1237.9	1199.8	438.90

(c) AES-256-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
Lat. Short (Cycles)	266	266	266	266	266	266
Lat. Long (Cycles)	160010	160010	160010	160010	160010	160010
NanGate 45 nm						
Power (mW)	2.057	1.665	1.959	1.925	1.906	2.296
Energy Short (nJ)	54.72	44.29	52.11	51.21	50.70	61.07
Energy Long (nJ)	32914	26642	31346	30802	30498	36738
UMC 65 nm						
Power (mW)	0.378	0.272	0.341	0.335	0.336	0.379
Energy Short (nJ)	10.05	7.24	9.07	8.91	8.94	10.08
Energy Long (nJ)	6048.4	4352.3	5456.3	5360.3	5376.3	6064.4
2-Round Unrolled						
Lat. Short (Cycles)	133	133	133	133	133	133
Lat. Long (Cycles)	80005	80005	80005	80005	80005	80005
NanGate 45 nm						
Power (mW)	5.241	4.421	5.007	4.918	4.894	5.679
Energy Short (nJ)	69.71	58.80	66.59	65.41	65.09	75.53
Energy Long (nJ)	41931	35370	40059	39347	39154	45435
UMC 65 nm						
Power (mW)	0.985	0.770	0.908	0.898	0.898	0.983
Energy Short (nJ)	13.10	10.24	12.08	11.94	11.94	13.07
Energy Long (nJ)	7880.5	6160.4	7264.5	7184.4	7184.4	7864.5

(d) SNOW-V-GCM

	LUT	DSE	S	F	T	TT
Round-Based						
Lat. Short (Cycles)	42	42	42	42	42	42
Lat. Long (Cycles)	10026	10026	10026	10026	10026	10026
NanGate 45 nm						
Power (mW)	2.722	2.290	2.544	2.502	2.481	2.466
Energy Short (nJ)	11.43	9.62	10.69	10.51	10.42	10.36
Energy Long (nJ)	2729.1	2296.0	2550.6	2508.5	2487.5	2472.4
UMC 65 nm						
Power (mW)	0.491	0.371	0.442	0.439	0.434	0.365
Energy Short (nJ)	2.062	1.558	1.856	1.844	1.823	1.533
Energy Long (nJ)	492.28	371.96	443.15	440.14	435.13	365.95
2-Round Unrolled						
Lat. Short (Cycles)	21	21	21	21	21	21
Lat. Long (Cycles)	5013	5013	5013	5013	5013	5013
NanGate 45 nm						
Power (mW)	7.128	5.670	6.830	6.698	6.625	5.948
Energy Short (nJ)	14.97	11.91	14.34	14.07	13.91	12.49
Energy Long (nJ)	3573.3	2842.4	3423.9	3357.7	3321.1	2981.7
UMC 65 nm						
Power (mW)	1.341	0.964	1.204	1.184	1.176	0.921
Energy Short (nJ)	2.816	2.024	2.528	2.486	2.470	1.934
Energy Long (nJ)	672.24	483.25	603.57	593.54	589.53	461.70

D.3 Partially Unrolled Synthesis Results

Table 10: Hardware synthesis figures of the Rocca-S circuit with a partial round function for two cell libraries and a clock frequency of 10 MHz.

	LUT	DSE	S	F	T	TT
Lat. Short (cycles)	88	88	88	88	88	88
Lat. Long (cycles)	10072	10072	10072	10072	10072	10072
NanGate 15 nm						
Area (μm^2)	16767	16881	9532	10145	9464	20399
Area (GE)	85281	85861	48482	51600	48136	103755
Critical Path (ns)	0.237	0.236	0.283	0.268	0.269	0.202
Throughput (Tbps)	0.540	0.542	0.452	0.478	0.476	0.634
Power (mW)	1.379	0.791	1.261	1.313	1.199	0.849
Energy Short (nJ)	12.14	6.960	11.10	11.55	10.55	7.471
Energy Long (nJ)	1388.9	796.7	1270.1	1322.5	1207.6	855.1
NanGate 45 nm						
Area (μm^2)	59359	62969	33798	36309	33721	75329
Area (GE)	74385	78909	42353	45500	42257	94397
Critical Path (ns)	2.47	2.34	2.94	2.68	2.89	2.07
Throughput (Tbps)	0.052	0.055	0.044	0.048	0.044	0.062
Power (mW)	2.587	2.022	1.390	1.501	1.392	2.540
Energy Short (nJ)	22.77	17.79	12.23	13.21	12.25	22.35
Energy Long (nJ)	2605.6	2036.6	1400.0	1511.8	1402.0	2558.3
UMC 65 nm						
Area (μm^2)	105810	115499	63227	67835	63054	138866
Area (GE)	73479	80208	43908	47108	43788	96435
Critical Path (ns)	7.54	7.35	8.27	7.62	8.11	6.36
Throughput (Tbps)	0.017	0.017	0.015	0.017	0.016	0.020
Power (mW)	1.084	0.523	0.848	0.842	0.810	0.474
Energy Short (nJ)	9.54	4.60	7.46	7.41	7.13	4.17
Energy Long (nJ)	1091.8	526.8	854.1	848.1	815.8	477.4
TSMC 90 nm						
Area (μm^2)	192141	212413	121994	132261	121984	254592
Area (GE)	68077	75260	43223	46861	43220	90204
Critical Path (ns)	4.36	4.36	4.97	4.68	4.73	4.36
Throughput (Tbps)	0.029	0.029	0.026	0.027	0.027	0.029
Power (mW)	1.952	1.034	1.831	1.802	1.688	0.933
Energy Short (nJ)	17.178	9.099	16.113	14.854	8.210	20.940
Energy Long (nJ)	1966.1	1041.4	1844.2	1815.0	1700.2	939.7

D.4 Software Reference Implementation

```
#include <memory.h>
#include <immintrin.h>
#include <stdlib.h>
#include <stdint.h>

#define ROCCA_KEY_SIZE       (32)
#define ROCCA_IV_SIZE        (16)
#define ROCCA_MSG_BLOCK_SIZE (32)
#define ROCCA_TAG_SIZE       (32)
#define ROCCA_STATE_NUM      ( 7)

typedef struct ROCCA_CTX {
   uint8_t key[ROCCA_KEY_SIZE/16][16];
   uint8_t state[ROCCA_STATE_NUM][16];
   size_t size_ad;
   size_t size_m;
} rocca_context;

#define load(m)    _mm_loadu_si128((const __m128i *)(m))
#define store(m,a) _mm_storeu_si128((__m128i *)(m),a)
#define xor(a,b)   _mm_xor_si128(a,b)
#define and(a,b)   _mm_and_si128(a,b)
#define enc(a,k)   _mm_aesenc_si128(a,k)
#define setzero()  _mm_setzero_si128()

#define ENCODE_IN_LITTLE_ENDIAN(bytes, v) \
  bytes[ 0] = ((uint64_t)(v) << (    3)); \
  bytes[ 1] = ((uint64_t)(v) >> (1*8-3)); \
  bytes[ 2] = ((uint64_t)(v) >> (2*8-3)); \
  bytes[ 3] = ((uint64_t)(v) >> (3*8-3)); \
  bytes[ 4] = ((uint64_t)(v) >> (4*8-3)); \
  bytes[ 5] = ((uint64_t)(v) >> (5*8-3)); \
  bytes[ 6] = ((uint64_t)(v) >> (6*8-3)); \
  bytes[ 7] = ((uint64_t)(v) >> (7*8-3)); \
  bytes[ 8] = ((uint64_t)(v) >> (8*8-3)); \
  bytes[ 9] = 0; \
  bytes[10] = 0; \
  bytes[11] = 0; \
  bytes[12] = 0; \
  bytes[13] = 0; \
  bytes[14] = 0; \
  bytes[15] = 0;

#define FLOORTO(a,b) ((a) / (b) * (b))

#define S_NUM      ROCCA_STATE_NUM
#define M_NUM      ( 2)
#define INIT_LOOP  (16)
#define TAG_LOOP   (16)
```

```
50
51  #define VARS4UPDATE \
52    __m128i k[2], state[S_NUM], stateNew[S_NUM], M[M_NUM];
53
54  #define VARS4ENCRYPT \
55    VARS4UPDATE \
56    __m128i Z[M_NUM], C[M_NUM];
57
58  #define COPY_TO_LOCAL(ctx) \
59    for(size_t i = 0; i < S_NUM; ++i) \
60    { state[i] = load(&((ctx)->state[i][0])); }
61
62  #define COPY_FROM_LOCAL(ctx) \
63    for(size_t i = 0; i < S_NUM; ++i) \
64    { store(&((ctx)->state[i][0]), state[i]); }
65
66  #define COPY_TO_LOCAL_IN_TAG(ctx) \
67    COPY_TO_LOCAL(ctx)    for(size_t i = 0; i < 2; ++i) \
68    { k[i] = load(&((ctx)->key[i][0])); }
69
70  #define COPY_FROM_LOCAL_IN_INIT(ctx) \
71    COPY_FROM_LOCAL(ctx)  for(size_t i = 0; i < 2; ++i) \
72    { store(&((ctx)->key[i][0]), k[i]); }
73
74  #define UPDATE_STATE(X) \
75    stateNew[0] = xor(state[6], state[1]); \
76    stateNew[1] = enc(state[0],     X[0]); \
77    stateNew[2] = enc(state[1], state[0]); \
78    stateNew[3] = enc(state[2], state[6]); \
79    stateNew[4] = enc(state[3],     X[1]); \
80    stateNew[5] = enc(state[4], state[3]); \
81    stateNew[6] = enc(state[5], state[4]); \
82    for(size_t i = 0; i < S_NUM; ++i) \
83    {state[i] = stateNew[i];}
84
85  #define INIT_STATE(key, iv) \
86    k[0] = load((key) + 16*0); \
87    k[1] = load((key) + 16*1); \
88    state[0] = k[1]; \
89    state[1] = load(iv); \
90    state[2] = load(Z0); \
91    state[3] = k[0]; \
92    state[4] = load(Z1); \
93    state[5] = xor(state[1], state[0]); \
94    state[6] = setzero(); \
95    M[0] = state[2]; \
96    M[1] = state[4]; \
97    for(size_t i = 0; i < INIT_LOOP; ++i) { \
98      UPDATE_STATE(M) \
99    } \
100   state[0] = xor(state[0], k[0]); \
101   state[1] = xor(state[1], k[0]); \
```

```
102   state[2] = xor(state[2], k[1]); \
103   state[3] = xor(state[3], k[0]); \
104   state[4] = xor(state[4], k[0]); \
105   state[5] = xor(state[5], k[1]); \
106   state[6] = xor(state[6], k[1]);
107
108 #define MAKE_STRM \
109   Z[0] = enc(xor(state[3], state[5]), state[0]); \
110   Z[1] = enc(xor(state[4], state[6]), state[2]);
111
112 #define MSG_LOAD(mem, reg) \
113   reg[0] = load((mem) +  0); \
114   reg[1] = load((mem) + 16);
115
116 #define MSG_STORE(mem, reg) \
117   store((mem) +  0, reg[0]); \
118   store((mem) + 16, reg[1]);
119
120 #define XOR_BLOCK(dst, src1, src2) \
121   dst[0] = xor(src1[0], src2[0]); \
122   dst[1] = xor(src1[1], src2[1]);
123
124 #define MASKXOR_BLOCK(dst, src1, src2, mask) \
125   dst[0] = and(xor(src1[0], src2[0]), mask[0]); \
126   dst[1] = and(xor(src1[1], src2[1]), mask[1]);
127
128 #define ADD_AD(input) \
129   MSG_LOAD(input, M) \
130   UPDATE_STATE(M)
131
132 #define ADD_AD_LAST_BLOCK(input, size) \
133   uint8_t tmpblk[ROCCA_MSG_BLOCK_SIZE] = {0}; \
134   memcpy(tmpblk, input, size); \
135   MSG_LOAD(tmpblk, M) \
136   UPDATE_STATE(M)
137
138 #define ENCRYPT(output, input) \
139   MSG_LOAD(input, M) \
140   MAKE_STRM \
141   XOR_BLOCK(C, M, Z) \
142   MSG_STORE(output, C) \
143   UPDATE_STATE(M)
144
145 #define ENCRYPT_LAST_BLOCK(output, input, size) \
146   uint8_t tmpblk[ROCCA_MSG_BLOCK_SIZE] = {0}; \
147   memcpy(tmpblk, input, size); \
148   MSG_LOAD(tmpblk, M) \
149   MAKE_STRM \
150   XOR_BLOCK(C, M, Z) \
151   MSG_STORE(tmpblk, C) \
152   memcpy(output, tmpblk, size); \
153   UPDATE_STATE(M)
```

```
154
155 #define DECRYPT(output, input) \
156   MSG_LOAD(input, C) \
157   MAKE_STRM \
158   XOR_BLOCK(M, C, Z) \
159   MSG_STORE(output, M) \
160   UPDATE_STATE(M)
161
162 #define DECRYPT_LAST_BLOCK(output, input, size) \
163   uint8_t tmpblk[ROCCA_MSG_BLOCK_SIZE] = {0}; \
164   uint8_t tmpmsk[ROCCA_MSG_BLOCK_SIZE] = {0}; \
165   __m128i mask[M_NUM]; \
166   memcpy(tmpblk, input, size); \
167   memset(tmpmsk, 0xFF , size); \
168   MSG_LOAD(tmpblk, C   ) \
169   MSG_LOAD(tmpmsk, mask) \
170   MAKE_STRM \
171   MASKXOR_BLOCK(M, C, Z, mask) \
172   MSG_STORE(tmpblk, M) \
173   memcpy(output, tmpblk, size); \
174   UPDATE_STATE(M)
175
176 #define SET_AD_BITLEN_MSG_BITLEN(sizeAD, sizeM) \
177   uint8_t bitlenAD[16]; \
178   uint8_t bitlenM [16]; \
179   ENCODE_IN_LITTLE_ENDIAN(bitlenAD, sizeAD); \
180   ENCODE_IN_LITTLE_ENDIAN(bitlenM , sizeM ); \
181   M[0] = load(bitlenAD); \
182   M[1] = load(bitlenM );
183
184 #define MAKE_TAG(sizeAD, sizeM, tag) \
185   SET_AD_BITLEN_MSG_BITLEN(sizeAD, sizeM) \
186   state[1] = xor(state[1], k[0]); \
187   state[2] = xor(state[2], k[1]); \
188   for(size_t i = 0; i < TAG_LOOP; ++i) { \
189     UPDATE_STATE(M) \
190   } \
191   __m128i tag128a = setzero(); \
192   for(size_t i = 0; i <= 3; ++i) { \
193     tag128a = xor(tag128a, state[i]); \
194   } \
195   __m128i tag128b = setzero(); \
196   for(size_t i = 4; i <= 6; ++i) { \
197     tag128b = xor(tag128b, state[i]); \
198   } \
199   store((tag)   , tag128a); \
200   store((tag)+16, tag128b);
201
202 static const uint8_t Z0[] = {0xcd,0x65,0xef,0x23,0x91, \
203 0x44,0x37,0x71,0x22,0xae,0x28,0xd7,0x98,0x2f,0x8a,0x42};
204 static const uint8_t Z1[] = {0xbc,0xdb,0x89,0x81,0xa5, \
205 0xdb,0xb5,0xe9,0x2f,0x3b,0x4d,0xec,0xcf,0xfb,0xc0,0xb5};
```

```
206
207 void rocca_init(rocca_context * ctx, const uint8_t * key, \
208 const uint8_t * iv) {
209     VARS4UPDATE
210     INIT_STATE(key, iv);
211     COPY_FROM_LOCAL_IN_INIT(ctx);
212     ctx->size_ad = 0;
213     ctx->size_m  = 0;
214 }
215
216 void rocca_add_ad(rocca_context * ctx, const uint8_t * in, size_t size)
217  {
218     VARS4UPDATE
219     COPY_TO_LOCAL(ctx);
220     size_t i = 0;
221     for(size_t size2 = FLOORTO(size, ROCCA_MSG_BLOCK_SIZE); \
222     i < size2; i += ROCCA_MSG_BLOCK_SIZE) {
223         ADD_AD(in + i);
224     }
225     if(i < size) {
226         ADD_AD_LAST_BLOCK(in + i, size - i);
227     }
228     COPY_FROM_LOCAL(ctx);
229     ctx->size_ad += size;
230 }
231
232 void rocca_encrypt(rocca_context * ctx, uint8_t * out, \
233 const uint8_t * in, size_t size) {
234     VARS4ENCRYPT
235     COPY_TO_LOCAL(ctx);
236     size_t i = 0;
237     for(size_t size2 = FLOORTO(size, ROCCA_MSG_BLOCK_SIZE); \
238     i < size2; i += ROCCA_MSG_BLOCK_SIZE) {
239         ENCRYPT(out + i, in + i);
240     }
241     if(i < size) {
242         ENCRYPT_LAST_BLOCK(out + i, in + i, size - i);
243     }
244     COPY_FROM_LOCAL(ctx);
245     ctx->size_m += size;
246 }
247
248 void rocca_decrypt(rocca_context * ctx, uint8_t * out, \
249 const uint8_t * in, size_t size) {
250     VARS4ENCRYPT
251     COPY_TO_LOCAL(ctx);
252     size_t i = 0;
253     for(size_t size2 = FLOORTO(size, ROCCA_MSG_BLOCK_SIZE); \
254     i < size2; i += ROCCA_MSG_BLOCK_SIZE) {
255         DECRYPT(out + i, in + i);
256     }
257     if(i < size) {
```

```
258         DECRYPT_LAST_BLOCK(out + i, in + i, size - i);
259     }
260     COPY_FROM_LOCAL(ctx);
261     ctx->size_m += size;
262 }
263 
264 void rocca_tag(rocca_context * ctx, uint8_t *tag) {
265     VARS4UPDATE
266     COPY_TO_LOCAL_IN_TAG(ctx);
267     MAKE_TAG(ctx->size_ad, ctx->size_m, tag);
268 }
```

E A Small GIFT-COFB

E.1 ANF Equations of the 3-Share GIFT-128 S-Box

Below we list the exact ANF equations for all component functions of the 3-share first-order threshold implementation of the GIFT S-box as proposed in [85].

$$
\begin{aligned}
\mathrm{S}_{G_1}(a_2,b_2,c_2,d_2,a_3,b_3,c_3,d_3) &= a_3 + b_3 + b_2c_2 + b_2c_3 + b_3c_2,\\
&\quad c_3 + 1,\\
&\quad b_3 + a_2c_2 + a_2c_3 + a_3c_2,\\
&\quad a_3 + b_3 + c_3 + d_3 + a_2b_2 + a_2b_3 + a_3b_2;\\
\mathrm{S}_{G_2}(a_1,b_1,c_1,d_1,a_3,b_3,c_3,d_3) &= a_1 + b_1 + b_1c_3 + b_3c_1 + b_3c_3,\\
&\quad c_1,\\
&\quad b_1 + a_1c_3 + a_3c_1 + a_3c_3,\\
&\quad a_1 + b_1 + c_1 + d_1 + a_1b_3 + a_3b_1 + a_3b_3;\\
\mathrm{S}_{G_3}(a_1,b_1,c_1,d_1,a_2,b_2,c_2,d_2) &= a_2 + b_2 + b_1c_1 + b_1c_2 + b_2c_1\\
&\quad c_2,\\
&\quad b_2 + a_1c_1 + a_1c_2 + a_2c_1,\\
&\quad a_2 + b_2 + c_2 + d_2 + a_1b_1 + a_1b_2 + a_2b_1;\\
\mathrm{S}_{F_1}(a_2,b_2,c_2,d_2,a_3,b_3,c_3,d_3) &= d_3 + a_2b_2 + a_2b_3 + a_3b_2,\\
&\quad b_3 + c_3 + d_3 + a_2d_2 + a_2d_3 + a_3d_2 + 1,\\
&\quad a_3 + b_3,\\
&\quad a_3 + 1;\\
\mathrm{S}_{F_2}(a_1,b_1,c_1,d_1,a_3,b_3,c_3,d_3) &= d_1 + a_1b_3 + a_3b_1 + a_3b_3,\\
&\quad b_1 + c_1 + d_1 + a_1d_3 + a_3d_1 + a_3d_3,\\
&\quad a_1 + b_1,\\
&\quad a_1;\\
\mathrm{S}_{F_3}(a_1,b_1,c_1,d_1,a_2,b_2,c_2,d_2) &= d_2 + a_1b_1 + a_1b_2 + a_2b_1,\\
&\quad b_2 + c_2 + d_2 + a_1d_1 + a_1d_2 + a_2d_1,\\
&\quad a_2 + b_2,\\
&\quad a_2.
\end{aligned}
$$

Curriculum Vitae

Andrea Caforio

Date of Birth	18.09.1993
Place of Birth	Zürich
Nationality	Swiss

Education

2019-2023	PhD, Cryptography Supervision: Prof. Serge Vaudenay Area: Symmetric Cryptography LASEC, Ecole Polytechnique Fédérale de Lausanne
2017-2019	MSc, Computer Science Ecole Polytechnique Fédérale de Lausanne
2014-2017	BSc, Computer Science Ecole Polytechnique Fédérale de Lausanne

Work Experience

2018	Software Engineer Taurus SA, Geneva
2013-2014	Development Aid Swiss Embassy, Bishkek, Kyrgyzstan

Languages

German	Mother Tongue
French	Fluent
English	Fluent
Russian	Intermediate

Teaching Assistantships

2022	CS-210, Functional Programming Prof. Martin Odersky & Prof. Viktor Kuncak
2021	COM-402, Information Security and Privacy Prof. Jean-Pierre Hubaux
2021	CS-119(g), Information, Calcul, Communication Prof. Jamila Sam
2020	CS-112(i), Programmation Orientée Objet Prof. Jamila Sam
2020	CS-438, Decentralised Systems Engineering Prof. Bryan Ford

Awards

Best Paper Award	Energy Analysis of Lightweight AEAD Circuits 19th International Conference on Cryptology and Network Security 2020 Vienna, Austria
Best Paper Award	A Study of Persistent Fault Analysis 9th International Conference on Security, Privacy and Applied Cryptographic Engineering 2019 Gandhinagar, India
EDIC Fellowship	One year PhD grant awarded for student excellency

Publications

1. Andrea Caforio and Subhadeep Banik. A Study of Persistent Fault Analysis. In: *Security, Privacy, and Applied Cryptography Engineering - 9th International Conference, SPACE 2019, Gandhinagar, India, December 3-7, 2019, Proceedings*. Ed. by Shivam Bhasin, Avi Mendelson, and Mridul Nandi. Vol. 11947. Lecture Notes in Computer Science. Springer, 2019, pp. 13–33 [46]

2. Andrea Caforio, Fatih Balli, and Subhadeep Banik. Energy Analysis of Lightweight AEAD Circuits. In: *Cryptology and Network Security - 19th International Conference, CANS 2020, Vienna, Austria, December 14-16, 2020, Proceedings*. Ed. by Stephan Krenn, Haya Shulman, and Serge Vaudenay. Vol. 12579. Lecture Notes in Computer Science. Springer, 2020, pp. 23–42 [43]

3. Fatih Balli, Andrea Caforio, and Subhadeep Banik. The Area-Latency Symbiosis: Towards Improved Serial Encryption Circuits. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.1 (2021), pp. 239–278 [9]

4. Andrea Caforio, Fatih Balli, and Subhadeep Banik. Melting SNOW-V: improved lightweight architectures. In: *J. Cryptogr. Eng.* 12.1 (2022), pp. 53–73 [44]

5. Subhadeep Banik, Andrea Caforio, Takanori Isobe, Fukang Liu, Willi Meier, Kosei Sakamoto, and Santanu Sarkar. Atom: A Stream Cipher with Double Key Filter. In: *IACR Transactions on Symmetric Cryptology* 2021.1 (2021), pp. 5–36 [20]

6. Andrea Caforio, F. Betül Durak, and Serge Vaudenay. Beyond Security and Efficiency: On-Demand Ratcheting with Security Awareness. In: *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II*. ed. by Juan Garay. Vol. 12711. Lecture Notes in Computer Science. Virtual Event: Springer, Heidelberg, Germany, May 2021, pp. 649–677 [50]

7. Andrea Caforio, Fatih Balli, Subhadeep Banik, and Francesco Regazzoni. A Deeper Look at the Energy Consumption of Lightweight Block Ciphers. In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*. IEEE, 2021, pp. 170–175 [45]

8. Andrea Caforio, Fatih Balli, and Subhadeep Banik. Complete Practical Side-Channel-Assisted Reverse Engineering of AES-Like Ciphers. In: *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*. Ed. by Vincent Grosso and Thomas Pöppelmann. Vol. 13173. Lecture Notes in Computer Science. Springer, 2021, pp. 97–117 [42]

9. Andrea Caforio, Daniel Collins, Ognjen Glamocanin, and Subhadeep Banik. Improving First-Order Threshold Implementations of SKINNY. in: *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings*. Ed. by Avishek Adhikari, Ralf Küsters, and Bart Preneel. Vol. 13143. Lecture Notes in Computer Science. Springer, 2021, pp. 246–267 [49]

10. Andrea Caforio, Subhadeep Banik, Yosuke Todo, Willi Meier, Takanori Isobe, Fukang Liu, and Bin Zhang. Perfect Trees: Designing Energy-Optimal Symmetric Encryption Primitives. In: *IACR Trans. Symmetric Cryptol.* 2021.4 (2021), pp. 36–73 [47]

11. Andrea Caforio, Daniel Collins, Subhadeep Banik, and Francesco Regazzoni. A Small GIFT-COFB: Lightweight Bit-Serial Architectures. In: *Progress in Cryptology - AFRICACRYPT 2022 - 13th International Conference on Cryptology in Africa, Fes, Morocco, July 18-20, 2022, Proceedings*. Ed. by Abderrahmane Nitaj and Lhoussain El Fadil. Vol. 13143. Lecture Notes in Computer Science. Springer, 2022, pp. 246–267 [48]

12. Subhadeep Banik, Andrea Caforio, Kazuhide Fukushima, Takanori Isobe, Shisaku Kiyomoto, Fukang Liu, Yuto Nakano, Kosei Sakamoto, Nobuyuki Takeuchi, and Ravi Anand. Rocca-S: Ultra High-Throughput and Quantum-Secure Authenticated Encryption. 2023 [19]

Banked on memory
Mummified circuitry
Skin graft machinery
Sputnik sickles found in the seats
Self-destruct sequence
This station is non-operational
Species growing
Bubbles in an IV loitering
Unknown origin
Is this the comfort of being afraid
Solar eclipsed
Black out the vultures
As they wait
Unknown, unknown
Unknown, unknown, yeah

(*One Armed Scissor - At the Drive-In*)

’Cause I walk onto the water
Buzzing with electrolytes and sacrilege
To be praised for my plastic lack of inspiration
I get that this is heinous
To burn up on re-entry and call the state a traitor
But I guess that’s only half the accusation, in hindsight
I was lost and didn’t have a map to recreate it
The home you said you came with
A moment of weakness labeled revelation

(*Star Baby - The Callous Daoboys*)