EPFL

Thèse n° 8974

# Attacks on some post-quantum cryptographic protocols: The case of the Legendre PRF and SIKE

Présentée le 5 octobre 2022

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

**Novak KALUĐEROVIĆ**

Acceptée sur proposition du jury

Prof. O. N. A. Svensson, président du jury
Prof. S. Vaudenay, Prof. A. Lenstra, directeurs de thèse
Prof. R. Schoof, rapporteur
Dr J. Bos, rapporteur
Prof. A. Chiesa, rapporteur

École polytechnique fédérale de Lausanne

2022

*Родитељима, који су веровали у мене,*
*и Ђуки, која је све време била уз мене,*
*чак и када нисмо били толико близу.*

# Acknowledgements

First and foremost, I would like to thank Arjen Lenstra for giving me full liberty in sailing the rough seas of a PhD, and for his unconstrained help and insightful remarks, mathematical and otherwise.

My sincere thanks go to Serge Vaudenay for accepting me to LASEC in the last year of my studies, for being a helpful and understanding advisor, and for teaching a great cryptography course whose notes I still use to this day.

When I came to EPFL, I was welcomed to LACAL as a part of the last generation of doctoral students in the lab. I wish to thank Rob Granger, Benjamin Wesolowski as well as Marguerite Delcourt for welcoming me and making me feel like a part of the team. I would further like to thank Dušan Kostić for many hours well spent over the coffee machine, discussing mathematics and other subjects in both Serbian and English, and above all for being a good friend and teaching me how to code. Let me also express my thanks to Thorsten Kleinjung who stayed in the lab until the end, and whom I've had the chance to have many productive discussions and receive insightful feedback. Last but not least I'd like to thank Aymeric Genet, who together with myself, makes part of the last generation of LACAL's PhD students. He introduced me to side-channels, for which I am grateful, and as this thesis shows, our collaboration has been most fruitful. Finally, I would like to mention Monique Amhof who made the administration-induced headaches go away and made my PhD easier.

I would also like to thank all members of LASEC, who have been very welcoming and helpful during the last year of my doctorate: Fatih Balli, Subhadeep Banik, Khashayar Barooti, Andrea Caforio, Daniel Collins, Boris Fouotsa, Loïs Huguenin, Laurane Marco, Abdullah Talayhan, Benedikt Tran and Hailun Yan, as well as Martine Corval who was always there to help.

I will also take this chance to mention all my friends from Switzerland, Italy, Serbia, and Montenegro, who made my journey unforgettable, reminding me where I started and reminding me to always look ahead. I'm also grateful to my friend Bakul Vinchhi for finding an apartment in the best part of the Lausanne and being a great flatmate for the last five years.

Lastly, I am deeply grateful to my whole family, and Snežana and Branislav, my parents, who supported me on my long academic journey away from home. Finally, thanks to Đuka, my girlfriend, for being with me even when the train rides from Schwetzingen to Lausanne were exhausting.

*Lausanne, September 27, 2022* Kolja

# Abstract

Post-quantum cryptography is a branch of cryptography which studies algorithms conjectured to be resistant to attacks performed on quantum computers. This excludes widely deployed public-key algorithms, such as RSA and Diffie-Hellman, whose hardness relies on the difficulty of factoring and solving discrete logarithms, problems known to be solvable by a quantum computer. This thesis treats two cryptographic schemes which are believed to be quantum-resistant and provides theoretical and practical attacks against them.

The first protocol is the generalised Legendre pseudorandom function — a random bit generator computed as the Legendre symbol of the evaluation of a secret polynomial at an element of a finite field. We introduce a new point of view on the protocol by analysing the action of the group of Möbius transformations on the set of secret keys (secret polynomials). We provide a key extraction attack by creating a table which is cubic in the number of the function queries, an improvement over the previous algorithms which only provided a quadratic yield. Furthermore we provide an ever stronger attack for a new set of particularly weak keys.

The second protocol that we cover is SIKE – supersingular isogeny key encapsulation. In 2017 the American National Institute of Standards and Technology (NIST) opened a call for standardisation of post-quantum cryptographic algorithms. One of the candidates, currently listed as an alternative key encapsulation candidate in the third round of the standardisation process, is SIKE. We provide three practical side-channel attacks on the 32-bit ARM Cortex-M4 implementation of SIKE.

The first attack targets the elliptic curve scalar multiplication, implemented as a three-point ladder in SIKE. The lack of coordinate randomisation is observed, and used to attack the ladder by means of a differential power analysis algorithm. This allows us to extract the full secret key of the target party with only one power trace.

The second attack assumes coordinate randomisation is implemented and provides a zero-value attack — the target party is forced to compute the field element zero, which cannot be protected by randomisation. In particular we target both the three-point ladder and isogeny computation in two separate attacks by providing maliciously generated public keys made of elliptic curve points of irregular order. We show that an order-checking countermeasure is effective, but comes at a price of 10% computational overhead. Furthermore we show how to modify the implementation so that it can be protected from all zero-value attacks, i.e., a zero-value is never computed during the execution of the algorithm.

Finally, the last attack targets a point swapping procedure which is a subroutine of the three-point ladder. The attack successfully extracts the full secret key with only one power trace even

if the implementation is protected with coordinate randomisation or order-checking. We provide an effective countermeasure — an improved point swapping algorithm which protects the implementation from our attack.

# Abstract

La crittografia post-quantistica é un ramo della crittografia che studia algoritmi che si ipotizzano essere resistenti ad attacchi eseguiti su computer quantistici. Questo esclude algoritmi a chiave pubblica ampiamente diffusi, quali RSA e Diffie-Hellman, e altri basati sulla fattorizzazione e sul problema del logaritmo discreto, problemi noti per essere risolvibili da un computer quantistico. Questa tesi tratta due schemi crittografici che si ritiene essere resistenti ai computer quantistici, e fornisce attacchi teorici e pratici contro di essi.

Il primo protocollo è la funzione pseudocasuale di Legendre generalizzata — un generatore di bit casuali calcolato come il simbolo di Legendre della valutazione di un polinomio segreto ad un elemento di un campo finito. Nella tesi introduciamo un nuovo punto di vista sul protocollo analizzando l'azione del gruppo delle trasformazioni di Möbius sull'insieme delle chiavi segrete (polinomi segreti). Forniamo un attacco per l'estrazione della chiave segreta creando una tabella di grandezza cubica nel numero di valutazioni della funzione di Legendre, un aumento rispetto ai precedenti algoritmi che fornivano solamente un rendimento quadratico. Inoltre, forniamo un attacco ancora più forte su un nuovo set di chiavi particolarmente deboli.

Il secondo protocollo che trattiamo è SIKE – "supersingular isogeny key encapsulation", tradotto "incapsulamento di chiavi usando isogenie supersingolari". Nel 2017 il National Institute of Standards and Technology americano (NIST) ha aperto un bando per la standardizzazione degli algoritmi crittografici post-quantistici. Uno dei candidati, attualmente elencato come candidato alternativo tra gli algoritmi di incapsulamento di chiavi, è SIKE. Questa tesi fornisce tre attacchi pratici tramite canali laterali (attacchi side-channel) all'implementazione di SIKE sul ARM Cortex-M4 a 32 bit.

Il primo attacco mira alla moltiplicazione scalare sulla curva ellittica, implementata come una scala a tre punti (three-point ladder) in SIKE. La mancanza di randomizzazione delle coordinate viene osservata e utilizzata per attaccare la scala per mezzo di un algoritmo di analisi differenziale del consumo di energia(differential power analysis). Questo ci permette di estrarre l'intera chiave segreta dell'entità attaccata con una sola traccia del consumo di energia (power trace).

Il secondo attacco presuppone che la randomizzazione delle coordinate sia già implementata e fornisce un attacco a valore zero – la entità attaccata è costretta a calcolare l'elemento zero del campo, il quale non può essere protetto dalla randomizzazione. In particolare, vengono attaccati sia la scala a tre punti che il calcolo della isogenia in due attacchi separati, fornendo chiavi pubbliche generate maliziosamente, e fatte di punti di curve ellittiche di ordine irregolare. Mostriamo che una contro misura di controllo dell'ordine dei punti è efficace, ma ha il prezzo di un overhead computazionale del 10%. Inoltre, mostriamo come modificare l'implementazione

in modo che possa essere protetta da tutti gli attacchi a valore zero, cioè, un valore zero non viene mai calcolato durante l'esecuzione dell'algoritmo.
Infine, l'ultimo attacco riguarda una procedura di scambio di punti che è un sottoprogramma della scala a tre punti. L'attacco estrae con successo l'intera chiave segreta con una sola traccia del consumo di energia anche se l'implementazione è protetta con la randomizzazione delle coordinate o con il controllo dell'ordine dei punti. Per combattere l'attacco forniamo una contromisura efficace – un algoritmo di scambio di punti migliorato, che protegge l'implementazione dal nostro attacco.

# Résumé

La cryptographie post-quantique est une branche de la cryptographie qui étudie les algorithmes supposés être résistants aux ordinateurs quantiques. Cela exclut les algorithmes largement déployés tels que RSA et Diffie-Hellman, ou d'autres qui reposent sur les problèmes de factorisation et de logarithme discret, qui sont connus pour être résolus par un ordinateur quantique. Cette thèse traite de deux protocols cryptographiques qui sont supposés être résistants aux ordinateurs quantiques et fournit des attaques théoriques et pratiques contre eux.

Le premier protocole est la fonction pseudo-aléatoire de Legendre généralisée — un générateur de bits aléatoires calculé comme le symbole de Legendre de l'évaluation d'un polynôme secret à un élément d'un corps fini. Nous introduisons un nouveau point de vue sur le protocole en analysant l'action du groupe des transformations de Möbius sur l'ensemble des clés secrètes (polynômes secrets). Nous fournissons une attaque d'extraction de clé en créant une table qui est cubique dans le nombre de requêtes de la fonction, une amélioration par rapport aux algorithmes précédents qui ne fournissaient qu'un rendement quadratique. En outre, nous proposons une attaque toujours plus puissante pour un nouvel ensemble de clés particulièrement faibles.

Le deuxième protocole que nous couvrons est SIKE – «supersingular isogeny key encapsulation», ou «encapsulation de clé basée sur les isogenies supersingulières». En 2017, le National Institute of Standards and Technology (NIST), l'Institut national américain des normes et de la technologie, a ouvert un appel à la standardisation d'algorithmes cryptographiques post-quantiques. L'un des candidats, actuellement répertorié comme un candidat alternatif d'encapsulation de clé dans le troisième tour du processus de normalisation, est SIKE. Nous fournissons trois attaques pratiques par canal latéral (attaques side-channel) sur l'implémentation 32-bit ARM Cortex-M4 de SIKE.

La première attaque vise la multiplication scalaire de la courbe elliptique, implémentée comme une échelle à trois points (three-point ladder) dans SIKE. L'absence de randomisation des coordonnées est observée et utilisée pour attaquer l'échelle au moyen d'un algorithme d'analyse de puissance différentielle. Cela nous permet d'extraire la clé secrète complète de la partie cible avec une seule trace de puissance.

La deuxième attaque suppose que la randomisation des coordonnées est mise en œuvre et fournit une attaque à valeur nulle — la partie cible est forcée de calculer l'élément nul du corps, qui ne peut pas être protégé par la randomisation. En particulier, nous ciblons à la fois l'échelle à trois points et le calcul de l'isogénie dans deux attaques distinctes en fournissant des clés publiques générées de manière malveillante et constituées de points de courbe elliptique d'ordre irrégulier. Nous montrons qu'une contre-mesure de vérification de l'ordre est efficace, mais au prix d'une surcharge de calcul de 10 %. En outre, nous montrons comment modifier l'implémentation de

manière à la protéger contre toutes les attaques à valeur nulle, c'est-à-dire qu'une valeur nulle n'est jamais calculée pendant l'exécution de l'algorithme.
Enfin, la dernière attaque vise une procédure d'échange de points qui est une sous-routine de l'échelle à trois points. L'attaque réussit à extraire la clé secrète complète avec une seule trace de puissance, même si l'implémentation est protégée par la randomisation des coordonnées ou la vérification de l'ordre. Nous proposons une contre-mesure efficace — un algorithme d'échange de points amélioré qui protège l'implémentation de notre attaque.

# Резиме

Пост-квантна криптографија је грана криптографије која проучава алгоритме за које се верује да су отпорни на нападе квантним рачунарима. Најраспрострањенији алгоритми асиметричне криптографије данас, РСА (RSA) и Дифи-Хелман (Diffie-Hellman), а и други чија се тежина заснива на проблему факторизације и дискретних логаритмима, могу бити разбијени помоћу квантних рачунара. Ова теза проучава два криптографска алгоритма који су сматрани отпорним на квантне рачунаре и нуди теоријске и практичне нападе на њих.

Први протокол је генерализована Лежандрова псеудонасумична функција – генератор насумичних битова који се израчунава као Лежандров симбол од вредности евалуације тајног полинома над елементом коначног поља. Представљени су нова тачка гледишта на скуп тајних кључева (тајних полинома), и напад за откривање тајног кључа који се заснива на грађењу табеле која је кубна у броју упита функција. Представљени напад је побољшање у односу на предходне најбоље алгоритме чије су табеле само квадратне у односу на број упита функције.

Други протокол који покривамо је SIKE – „supersingular isogeny key encapsulation", или размена кључева суперсингуларним изогенијама. У 2017. амерички национални институт за стандарде и технологије (NIST) је отворио позив за стандардизацију пост-квантних криптографских алгоритама. Један од кандидата, тренутно на листи алтернативних кандидата за размену кључева, је SIKE. У овој тези представљена су три практична напада споредним каналима (тзв. side-channel напади) на 32-битну имплементацију SIKE-а на ARM Cortex-M4 микропроцесору.

Први напад таргетира операцију скаларног производа у елиптичким кривама, имплементирану кроз лествицу са три тачке (тзв. three-point ladder) у SIKE-у. Примећен је недостатак рандомизације координата и искориштен како би се извршио напад на мердевине помоћу диференцијалне анализе потрошње струје (тзв. differential power analysis).

Други напад полази од претпоставке да је имплементација заштићена рандомизацијом координата и нуди „напад нултих вредности" – нападнута странка је натерана да израчуна елемент нула у пољу који не може бити заштићен рандомизацијом. Прецизније, таргетирани су и лествице са три тачке и изогенија, кроз два одвојена напада. Напади се извршавају тако што се нападнутој странки предаје злонамерно генерисан јавни кључ састављен из тачака на елиптичкој криви које су неправилног реда. Поред тога, доказана је успешност заштитне мере провере реда тачака, која додуше долази уз додатне трошкове у рачунању – око 10% више рачунања у односу на алгоритам који не користи заштитну меру. Такође,

представљен је начин да се додатно измени имплементација тако да буде заштићена од свих напада нултих вредности, т.j., да вредност нула никада не буде израчуната у току извршења алгоритма.

Последњи напад таргетира операцију замене тачака која је сама подрутина унутар лествице са три тачке. Напад успешно открива цео тајни кључ са само једним мерењем напона на микропроцесору у току рада алгоритма, чак и када је имплементација заштићена рандомизацијом координата или провером реда тачака. Представљена је и делотворна контрамера – побољшани алгоритам за замену тачака који успешно брани имплементацију од наведеног напада.

# Contents

# Introduction

What is a quantum computer? "*It's just an excuse to give cryptographers jobs*".

Whether you're a sceptic (or as sceptics say – a *realist*), or if you believe in the inevitable change of the computing paradigm towards quantum, the theory of quantum computing is here to stay. The practice, on the other hand, is lagging behind by about 10 years – the realists' new natural constant. At the time of the writing of this thesis, quantum computers were commercially available [IBM09]. However they were limited to working on only 20 quantum bits, or qubits. There are serious doubts about the scalability of quantum computers, and some researchers such as Kalai [Kal16] argue that there are physical barriers which make quantum computing infeasible. The argument, in a nutshell, is "*Noisy quantum systems will not allow building quantum error-correcting codes needed for quantum computation*" [Kal19].

Practice aside, in theory quantum computers can solve some problems that are classically hard. For example the Deutsch-Jozsa problem [DR92], definition 1, is solved on a quantum computer with only one query, while a classical computer requires at least two queries of a secret function.

**Definition 1** (The Deutsch-Jozsa problem)**.** Let $f : \{0,1\} \to \{0,1\}$ be a function. Given oracle access to $f$, decide if $f$ is constant or not.

The Bernstein-Vazirani problem [BV97], definition 2, increases the gap further – it is solvable with a single query on a quantum computer, but on a classical one a polynomial number of queries is needed.

**Definition 2** (The Bernstein-Vazirani problem)**.** Let $s \in \{0,1\}^n$ be a secret and let $f : \{0,1\}^n \to \{0,1\}$ be a function such that $f((x_1,\ldots,x_n)) = \langle s,x\rangle = s_1x_1 \oplus \cdots \oplus s_nx_n$. Given oracle access to $f$, find $s$.

Finally, an exponential separation is given by Simon's problem [Sim97], definition 3, which is solved in a linear number of queries on a quantum computer, while a classical one requires an exponential number of queries.

**Definition 3** (Simon's problem)**.** Let $s \in \{0,1\}^n$ be a secret and let $f : \{0,1\}^n \to \{0,1\}^*$ be a function such that $f(x) = f(y)$ if and only if $x - y = s$ (i.e., $x_i \oplus y_i = s_i$ for $i = 1,\ldots,n$). Given oracle access to $f$, find $s$.

Similarly, Shor's algorithm [Sho97] solves problems that are believed to be hard, and on which present day cryptography is based. In particular the RSA problem, factoring and the discrete logarithm problem all fall prey to Shor's algorithm.

However, quantum computers are not almighty. So far NP-hard problems have not been solved (in polynomial time) by quantum computers. Similarly, secret-key cryptography is only lightly scratched by quantum computers – Grover's algorithm [Gro96] can solve function inversion (or unsorted search) in time square root of the size of the input space. A quadratic speed up over the best classical algorithms, but not an exponential one. In most cases secret-key cryptography solves the problem of the quantum threat by doubling the security parameters.

For public-key cryptography, however, increasing the parameters is not enough to promise quantum resistance. Instead, new algorithms are being developed, and more importantly, new hardness assumptions are made, in particular ones that, as far as one is able to infer, cannot be solved by quantum algorithms.

**The generalised Legendre pseudorandom function.** The first algorithm that this thesis covers is the generalised Legendre pseudorandom function (PRF). Using the Legendre symbol in a pseudorandom function was originally proposed by Damgård [Dam90]. Initially, the Legendre PRF was defined for some prime $p$ and secret key $k \in \mathbb{F}_p$ as an oracle $\mathcal{O}$ which at input $x$ computes the Legendre symbol $\mathcal{O}(x) = \left(\frac{x+k}{p}\right)$. Later generalisations of the protocol by Russell and Shparlinski [RS04] proposed to use a higher degree polynomial in $x$ as the key. The generalised Legendre PRF is defined for a prime $p$ and a secret key $f \in \mathbb{F}_p[x]$ as an oracle $\mathcal{O}$ which at input $x$ computes $\mathcal{O}(x) = \left(\frac{f(x)}{p}\right)$. The prime $p$ was originally considered to be secret by Damgård, but we assume that it is publicly known. Damgård conjectured that when $f$ is linear (the original Legendre PRF case), given a sequence of Legendre symbols of consecutive elements it is hard to predict the next one, even when the prime $p$ is public. Similar problems conjectured to be hard were also proposed [GRR+16], such as finding the secret polynomial while being given access to $\mathcal{O}$ and distinguishing $\mathcal{O}$ from a random function, for both the linear and the general Legendre PRF. So far no classical polynomial time algorithms were found for either of these problems and it is believed that they are hard. A quantum polynomial time algorithm was given by Russel and Shparlinski [RS04], but it relies on querying a quantum oracle, i.e., querying the oracle on a quantum state. No subexponential algorithms are known when only classical queries to the oracle are allowed. Currently, the Legendre PRF can be considered to be quantum-resistant.

While mathematically simple, the Legendre PRF was considered to be too computationally expensive for traditional cryptographic settings and thus not used in practice as a pseudorandom function. A recent result by Grassi et al. [GRR+16] showed that it can be used in multiparty-computation settings, and that in that paradigm it is orders of magnitude faster than the alternative symmetric-key primitives. This is mainly due to the homomorphic property of the Legendre symbol and the possibility of evaluating it with only three modular multiplications in arithmetic circuit multi-party computations, which makes it a very efficient MPC friendly PRF candidate. In

comparison, SHA256 requires tens of thousands of multiplications while AES needs 290 in the MPC setting [Fei19].

There are plans to incorporate the Legendre PRF into the Ethereum 2.0 protocol [Fei19], and in order to motivate research in this direction, the Ethereum foundation published a number of challenges. In each challenge, a list of $M = 2^{20}$ Legendre symbols of consecutive elements was provided. The difficulty of the challenge was based on the size of the underlying prime field which varied from 64 to 148 bits. The goal was to find the secret key which parametrises the linear Legendre PRF. We provide the solutions for challenges of size 64, 74 and 84 bits.

**Supersingular isogeny key encapsulation.** The second cryptographic algorithm that we cover is SIKE – Supersingular isogeny key encapsulation.

Given the current state of the "quantum threat" and the substantial amount of research done on the topic of quantum computers, the American National Institute for Standards and Technology (NIST) has started a project to solicit, evaluate and standardise post-quantum cryptographic protocols [Div16]. The submissions for the standardisation process are divided into two categories – key encapsulation mechanisms (KEM) and digital signatures. SIKE, as the name suggests, is a key encapsulation mechanism.

The NIST standardisation process started in 2017. Initially there were 69 KEM submissions. The submitted protocols can be classified into four groups depending on the algorithmic approaches that were used:

- Lattice based algorithms,
- Multivariate quadratics based algorithms,
- Code based algorithms,
- Isogeny based algorithms.

Out of all the submissions SIKE was the lone isogeny candidate. In the meanwhile the standardisation process reached the third round. At the time of writing there are nine algorithms left, out of which four are designed as finalists, and five as alternate candidates. SIKE is one of the remaining alternate candidates.

The development of isogeny-based cryptography protocols started in 1997 by Couveignes [Cou06], only to be independently rediscovered in 2006 by Rostovsev and Stolbunov [RS06]. The main building blocks of their algorithm are isogenies between ordinary elliptic curves. The proposed construction was shown to be resistant to known classical attacks, but a subexponential quantum attack was found by Childs et al. [CJS14]. The attack – finding a secret isogeny between two curves – is based on computing the endomorphism ring of the elliptic curves and observing that isogenous curves are in a one-to-one relation with elements of the class group of said ring. The class group is Abelian and thus finding the hidden isogeny reduces to a hidden shift problem in an Abelian group. This attack was mitigated by Jao and De Feo [DJ11] (and Plût [DJP14]) who propose

to use supersingular instead of ordinary elliptic curves in an algorithm called SIDH (Supersingular isogeny Diffie-Hellman) which later became SIKE. Due to the nature of supersingular elliptic curves (i.e., the non-commutativity of their endomorphism ring), the previously mentioned attack is prevented. In addition to the lack of subexponential attacks in both classical and quantum settings, the new SIKE algorithm stood out in its simple structure reminiscent of the classical Diffie-Hellman protocol, but also, and more importantly, SIKE was more efficient and had lower key sizes. Over the years, SIKE was improved [AJK+16, CLN16, ZSP+18, BF18a, BF18b] and the current implementation stands competitive with respect to other NIST candidates in the third round. One of the main downsides of SIKE is its high run-time which currently qualifies the scheme as the slowest surviving candidate, at the moment being around 300 times slower than the fastest candidate. However, this downside is compensated with the lowest key sizes among all quantum-resistant candidates, with the public key size around 3 times smaller than the key size of second smallest candidate. The trade-off between the cost-effectiveness of the key size and the computational cost was studied in [Lan18, Kwi19, PST20, Wei20].

This thesis follows the NIST recommendation to study side-channel attacks on post-quantum cryptographic schemes [Moo18, Apo20] and consists of the side-channel analysis of the SIKE implementation adapted to the 32-bit ARM Cortex-M4 chip architecture. The Cortex-M4 [Hol] is a low-power and low-cost embedded microcontroller from the ARM Cortex-M family, which is recommended by NIST for post-quantum cryptography evaluation [Moo19, KRSS19]. As such, the Cortex-M4 should be used with care in cryptographic settings. The implementation that we target in all attacks is the Cortex-M4 implementation by Azarderakhsh et al. [SAJA20], which is included in the official third round NIST submission of SIKE [JAC+17]. In addition to the implementation by Azarderakhsh et al., the submission also includes an optimised implementation in C for 32-bit architectures. Both implementations have some level of side-channel resistance as they are both constant-time. The main differences lie in the low-level functions, such as multi-precision additions, multiplications, and modular reductions that have been rewritten in assembly in order to take full advantage of the Cortex-M4 capabilities. In addition to the before mentioned implementations, the submission also contains a reference implementation which is written in C, uses GMP for modular arithmetic, and uses affine coordinates. The reference implementations is orders of magnitude slower than the other two due to the excessive number of costly field inversions that are computed.

This thesis covers three attacks on SIKE in Chapters 4, 5, and 6. The attacks are arranged in order in which they were discovered, and also in order of strength – attacks in later chapters defeat the countermeasures proposed in previous chapters. The attacks are based on side-channels, exploiting physical properties of the microprocessor which is running SIKE in order to extract secret information. Different types of side-channel approaches are used – differential power analysis, electromagnetic attacks, collision power analysis, clustering power analysis. They are explained in the corresponding chapters, however, the emphasis will be on the theoretical side of the attacks rather than the techniques used in side-channel analysis.

In Chapter 4 we give a differential power analysis attack which targets the elliptic curve scalar

multiplication procedure in SIKE. The scalar multiplication operation is computed by means of a three-point ladder – a constant-time ladder which takes for input a triple of elliptic curve points $(Q, P, Q-P)$ and outputs the point $R = Q + [sk]P$, where $sk$ is a secret scalar. The ladder is computed by looping over all bits of the secret scalar $sk$, and at each iteration computing a *double-and-add* function $(Q, P, Q-P) \mapsto ([2]Q, P+Q, Q-P)$ which is preceded by a conditional swap on the last two points of the triple. Our attack is based on the observation that the representation of projective coordinates is not protected by (multiplicative) coordinate randomisation. We exploit the lack of coordinate randomisation by guessing the values of point coordinates and computing the correlation between said values and power consumption in order to confirm our guesses. In particular, the attack consecutively extracts the secret bits of $sk$. For each bit of $sk$, we assume the knowledge of the points triple at the beginning of the corresponding loop iteration, and we make two hypothesis on the conditional swap for the two bit guesses. In such manner we obtain two possible elliptic point triples. We proceed by computing the coordinates of the points in the two triples, and correlating them with the power consumption. High correlation indicates a correct guess. We use this procedure to extract the whole secret key by using only one power trace of the three-point ladder procedure.

In Chapter 5 we provide two attacks on a SIKE implementation which is assumed to be protected by coordinate randomisation. The attacks aim to defeat the countermeasure by forcing the target party to compute projective points which have 0 as one of their coordinates, and which therefore cannot be protected by coordinate randomisation. The first of the two attacks targets the three-point ladder procedure. We create a malicious public key formed of elliptic curve points of an order which is different from the order of honestly generated public keys. The malicious points are such that when the target party computes the three-point ladder with those points as input, they are forced to compute an elliptic curve point equal to $[0:1]$ or $[1:0]$ if and only if a secret bit of their secret key is equal to a hypothesised value. Side-channel analysis in form of electromagnetic measurements is used in order to deduce if a projective coordinate is equal to 0, and therefore to deduce the value of a secret bit. The second attack creates a malicious public key of a similar form, and targets the isogeny computation procedure. When the target is provided with a malicious public key, the isogeny computation leads to irregular behaviour. We characterise such behaviour, and form our public keys so that during the isogeny computation only two things can happen – eventually all field values become 0 or all field values become random. In particular we make the public keys depend on a digit of the secret key so that the former situation happens if the secret key digit is correctly guessed, and the latter if the digit is incorrectly guessed. This allows us to extract the whole secret key of the target party by using a power analysis distinguisher; in particular we used collision power analysis. Furthermore we show that the reference implementation (which is different from the original target – Cortex-M4 implementation) crashes if all field values become 0, which leads to a remote full key extraction attack. The chapter finishes by providing a countermeasure – the order checking procedure, originally suggested by Costello, Longa, Naehrig in [CLN16]. However, the countermeasure comes at a cost of 10% performance overhead, which puts the future of SIKE, which is already the slowest of all post-quantum key encapsulation algorithms, in a difficult position.

In Chapter 6 we provide a clustering power analysis attack on an implementation of SIKE which is protected by coordinate randomisation and order checking. We target the three-point ladder, in particular the point swapping operation. This function swaps two elliptic curve points depending on the value of a secret bit. In particular, the secret bit is expanded into a mask – `0xFFFFFFFF` or `0x00000000` – which is then used in a conditional XOR swap algorithm. The high Hamming distance between the two possible masks allows for the creation of a side-channel distinguisher. The attack is performed by measuring all the executions of the swaps, and assigning them to one of the two clusters based on the power consumption during the corresponding mask computation. Two different clustering algorithms are studied, $k$-cluster and thresholding. Both approaches extracted the whole secret key in each of the 1000 experiments that were performed. Finally, we provide an effective countermeasure based on splitting the mask into two random shares and correspondingly adjusting the conditional XOR swap. The countermeasure successfully protects from the attack in question, and comes at a negligible performance overhead when compared to the overall runtime of SIKE.

**Publications.** The four chapters of this thesis are based on previous work which was written in collaboration with other authors.

- Chapter 3 is based on the paper "*Cryptanalysis of the generalised Legendre pseudorandom function*", [KKK20a], which was written in collaboration with Thorsten Kleinjung and Dušan Kostić. The paper is published in "*ANTS XIV: Proceedings of the Fourteenth Algorithmic Number Theory Symposium*".
- Chapter 4 is based on the paper "*Full key recovery side-channel attack on ephemeral SIKE*", [GdGK21], which was written in collaboration with Aymeric Genêt and Natacha Linard de Guertechin. The paper is published in "*COSADE 2021: Proceedings of the 12th international workshop on Constructive Side-Channel Analysis and Secure Design*".
- Chapter 5 is based on the paper "*SIKE channels: Zero-Value Side-Channel Attacks on SIKE*", [DEG$^+$22], which was written in collaboration with Luca De Feo, Nadia El Mrabet, Aymeric Genêt, Natacha Linard de Guertechin, Simon Pointié and Élisse Tasso. The paper is under revision for TCHES 2022. The preprint of the paper is available at https://eprint.iacr.org/2022/054.
- Chapter 6 is based on the paper "*Single-trace clustering power analysis of the point swapping procedure in the three point ladder of Cortex-M4 SIKE*", [GK22], which was written in collaboration with Aymeric Genêt. The paper is accepted at "*COSADE 2022: Proceedings of the 13th international workshop on Constructive Side-Channel Analysis and Secure Design*".

# Part I Background

# 1 The Legendre pseudorandom function

The usage of Legendre symbols in a pseudorandom function (PRF) is an idea originally proposed by Damgård [Dam90]. Further generalisations with higher degree polynomials were proposed by Russell and Shparlinski [RS04]. In both cases a prime $p$ is given and the Legendre PRF is modelled as an oracle $\mathcal{O}$ that on input $x$ outputs the Legendre symbol $\left(\frac{f(x)}{p}\right)$, where $f(x) \in \mathbb{F}_p[x]$ is a secret key. Damgård conjectured that when $f$ is linear, given a sequence of Legendre symbols of consecutive elements it is hard to predict the next one. Similar problems conjectured to be hard were also proposed [GRR+16], such as finding the secret polynomial while being given access to $\mathcal{O}$ and distinguishing $\mathcal{O}$ from a random function. So far no polynomial time algorithms were found for either of these problems and it is believed that they are hard. Until recently practical applications have been limited, primarily due to availability of much faster alternatives.

A recent result by Grassi et al. [GRR+16] sparked an interest in the linear Legendre PRF because it was found suitable as a multi-party computation (MPC) friendly pseudorandom generator. This is mainly due to the homomorphic property of the Legendre symbol which allows the function to be evaluated very efficiently in the MPC paradigm. The Legendre PRF is order of magnitudes faster then the alternatives which are mostly block-cipher based.

There are plans to use this construction as a PRF in the Ethereum blockchain [Fei19]. For this purpose the Legendre PRF was shown to be a great candidate because of its efficiency. In comparison, SHA256 requires tens of thousands of multiplications, AES needs 290 in the MPC setting, while the Legendre PRF requires only 3 [Fei19].

## 1.1 Mathematical background

We recall the definitions and some properties of the main mathematical objects used to define the Legendre pseudorandom function.

**Definition 1.1.1** (Legendre symbol)**.** The Legendre symbol of an element $x$ in $\mathbb{F}_p$ is defined as:

$$\left(\frac{x}{p}\right) = x^{\frac{p-1}{2}} = \begin{cases} 1 & \text{if } x \in \mathbb{F}_p^* \text{ is a square mod } p \\ 0 & \text{if } x = 0 \text{ mod p} \\ -1 & \text{if } x \in \mathbb{F}_p^* \text{ is not a square mod } p. \end{cases} \tag{1.1}$$

It stems directly from the definition that the Legendre symbol is a multiplicative function, i.e., that the following formula stands:

$$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right). \tag{1.2}$$

This property is crucial for the usage of the Legendre symbol in the MPC setting.

In practical implementations, the Legendre symbol is usually redefined by setting $\left(\frac{0}{p}\right) := 1$. The main reason behind this choice is that this way the Legendre symbol becomes a binary function and its output can be saved in a single bit. The main drawback is that the function loses the multiplicative property.

Independently of the definition of $\left(\frac{0}{p}\right)$ we will assume that the multiplicative property of the Legendre symbol stands. This is a non-problem and the reader should be easily convinced that the algorithms that we provide terminate in the same expected time and with the same probability.

**Definition 1.1.2** (Legendre sequence)**.** The Legendre sequence with starting point $a$ and length $L$ is defined to be the sequence of Legendre symbols evaluated at $L$ consecutive elements starting from $a$. We denote it with $\{a\}_L$:

$$\{a\}_L := \left(\frac{a}{p}\right), \left(\frac{a+1}{p}\right), \left(\frac{a+2}{p}\right), \ldots, \left(\frac{a+L-1}{p}\right). \tag{1.3}$$

Every starting point $a$ fully determines its sequence of length $L$, but not vice versa – that property depends on $L$. In general, these sequences are as well distributed as one can hope them to be. It is well known that when $L = 1$ *"half"* of the sequences with starting point $a$ are equal to 1, and the other *"half"* of the sequences are equal to $-1$. Similar properties are true for larger $L$, and in general, following a theorem of Davenport, around one in $2^L$ elements of $\mathbb{F}_p$ is a starting point of a given sequence of length $L$.

**Theorem 1.1.1** (Davenport, 1933 [Dav33])**.** Let $S$ be a finite sequence of $\pm 1$'s of length $L$. Then the number of elements of $\mathbb{F}_p$ whose sequence is equal to $S$ satisfies

$$\#\left\{a \in \mathbb{F}_p \,\middle|\, \{a\}_L = S\right\} = \frac{p}{2^L} + O(p^\varepsilon) \tag{1.4}$$

where $0 < \varepsilon < 1$ is a constant depending only on $L$.

In Chapter 3 we assume that $L$ is such that $\{a\}_L$ uniquely defines $a$, i.e., that the following holds

$$\{a\}_L = \{b\}_L \text{ if and only if } a = b. \tag{1.5}$$

For (1.5) to hold we need to have $L = \Omega(\log_2 p)$. The only provable upper bound comes from the Weil bound [Wei48] and is $L = O(\sqrt{p}\log p)$ which is exponential.

Our computational results, together with other statistical data on the distribution of Legendre sequences [Dam90], indicate that on average over all sequences $S$ of length $L$, there are $\frac{p}{2^L} + O(1)$ elements whose Legendre sequences are equal to $S$. In other words, for a random $S$ and a random $j$ we have $\{j\}_L = S$ with probability $\frac{1}{2^L}$. A good estimate of $L$ in terms of $p$ is $L = \lfloor 2\log_2 p \rfloor$.

## 1.2 The Legendre pseudorandom function

In this section we define the Legendre pseudorandom function, and its higher degree generalisation.

**Definition 1.2.1** (Pseudorandom Functions)**.** A pseudorandom function family $\{\mathcal{O}_k\}_k$ is a set of functions with the same domain and codomain indexed by a set of keys $k$ such that a function $\mathcal{O}_k$ chosen randomly over the set of $k$-values cannot be distinguished from a random function.

**Definition 1.2.2** (Legendre PRF)**.** The Legendre pseudorandom functions are functions $\mathcal{O}_k$ from $\mathbb{F}_p$ to $\{-1, 1\}$ indexed by $k \in \mathbb{F}_p$ and defined as

$$\mathcal{O}_k(x) = \left(\frac{x+k}{p}\right). \tag{1.6}$$

**Definition 1.2.3** (Higher degree Legendre PRF)**.** The Legendre pseudorandom functions of degree $r$ are a family of functions $\mathcal{O}_f$ from $\mathbb{F}_p$ to $\{-1, 1\}$ indexed by $f = k_r x^r + \ldots + k_1 x + k_0 \in \mathbb{F}_p[x]$ and defined as

$$\mathcal{O}_f(x) = \left(\frac{f(x)}{p}\right). \tag{1.7}$$

The degree $r$ is assumed to be polylogarithmic in $p$.

Two oracles $\mathcal{O}_f(x)$ and $\mathcal{O}_{f/k_r}(x)$ are the same up to multiplication by $\left(\frac{k_r}{p}\right)$ and therefore we can assume the polynomial $f$ to be monic. The case of linear $f(x)$ thus reduces to the standard Legendre PRF as introduced by Damgård which we from now on refer to as the linear Legendre PRF.

The polynomial $f(x)$ is considered up to multiplication by a square in $\mathbb{F}_p[x]$ since the Legendre symbol is invariant under square factors of $f(x)$. This is not entirely true as a square linear factor introduces a zero and may change the output of the oracle at one point, but the reader should be convinced that this can be safely ignored.

The secret key space, i.e., the space from which we choose $f(x)$ is the space of monic polynomials modulo squares. The number of such polynomials equals $p^r - p^{r-1}$ for $r > 1$ (see [Ber15], problem 3.3) and $p$ for $r = 1$.

**Definition 1.2.4** (Generalised Legendre sequence)**.** The length $L$ Legendre sequence of a polynomial $f(x)$ is denoted by $\{f\}_L$ and defined as

$$\{f\}_L := \left(\frac{f(0)}{p}\right), \left(\frac{f(1)}{p}\right), \left(\frac{f(2)}{p}\right), \ldots, \left(\frac{f(L-1)}{p}\right). \tag{1.8}$$

As a generalisation to Theorem 1.1.1 and property (1.5) we assume that $L$ is such that $\{f\}_L$ uniquely defines $f$, i.e., that the following holds

$$\{f\}_L = \{g\}_L \text{ if and only if } f = g. \tag{1.9}$$

With $r$ the degree of $f$ we have $L = \Omega(r \log p)$. We assume that property (1.9) holds for $L = \Theta(r \log p)$. A reasonable estimate is $L = \lfloor 2r \log p \rfloor$. Throughout Chapter 3 we include the dependence on $L$ in the complexity of our algorithms.

## 1.3 Hard Problems

There are three main problems conjectured to be hard, and on which the security of the Legendre PRF is based.

**Definition 1.3.1** (Generalised Legendre Symbol Problem - GLSP)**.** Let $f$ be a uniformly random monic square-free polynomial. Given access to an oracle $\mathcal{O}$ that on input $x \in \mathbb{F}_p$ computes $\mathcal{O}(x) = \left(\frac{f(x)}{p}\right)$, find $f$.

**Definition 1.3.2** (Decisional Generalised Legendre Symbol Problem - DGLSP)**.** Let $f$ be a uniformly random monic square-free polynomial. Let $\mathcal{O}_0$ be an oracle that on input $x \in \mathbb{F}_p$ computes $\mathcal{O}_0(x) = \left(\frac{f(x)}{p}\right)$, and let $\mathcal{O}_1$ be an oracle that on input $x$ outputs a random value in $\{-1, +1\}$. Given access to $\mathcal{O}_b$ where $b$ is an unknown random bit, find $b$.

**Definition 1.3.3** (Next Symbol Problem - NSP)**.** Given a Legendre sequence $\{f\}_M$ of $M = \text{polylog}(p)$ symbols, find $\left(\frac{f(M)}{p}\right)$, or equivalently find $\{f\}_{M+1}$.

The DGLSP trivially reduces to GLSP and NSP. In the other direction, following a theorem of Yao [Kra86] on general pseudorandom functions, predicting the next bit of a pseudorandom function is as hard as distinguishing it from a truly random one. Therefore, under polynomial time reductions, we have that NSP = DGLSP ≤ GLSP.

# 2 SIKE

SIKE, the supersingular isogeny key encapsulation is a post-quantum key encapsulation mechanism (KEM) built on the theory of isogeny graphs of supersingular elliptic curves.

We start by introducing the mathematical background – finite fields, elliptic curves, isogenies. Then we introduce supersingular isogeny Diffie-Hellman (SIDH), the main building block of SIKE. Finally we end by introducing SIKE which is essentially a Fujisaki-Okamoto wrapper of SIDH.

A general overview of isogeny-based cryptography can be found in [DF17]. For a complete tutorial on SIKE, the reader is advised to see [Cos19].

## 2.1 Mathematical background

Let $q = p^r$ be an odd prime power, and denote with $\mathbb{F}_q$ the finite field with $q$ elements. All primes denoted with $p$ are assumed to be strictly greater than 3. We denote with $\overline{\mathbb{F}}$ the algebraic closure of a field $\mathbb{F}$. The projective space of degree $n$ over a field $\mathbb{F}$ is denoted with $\mathbb{P}^n(\mathbb{F})$, and an element $P \in \mathbb{P}^n(\mathbb{F})$ is denoted with $P = [X_0 : \ldots : X_n]$. We call coordinate randomisation of $P$ the process of sampling a non-zero element $\xi \in \mathbb{F}$ at random, and multiplying all coordinates of $P$ with $\xi$, obtaining $P = [\xi X_o : \ldots : \xi X_n]$.

Equivalence of integers $a$ and $b$ modulo $n$ is denoted with $a \underset{n}{\equiv} b$. The ring of integers modulo an integer $n$ is denoted with $\mathbb{Z}/(n)$.

We will use the minimal amount of definitions necessary to cover SIKE. For a more technical introduction to elliptic curves we refer the reader to [ST15, Sil09, Cas91].

**Definition 2.1.1** (Algebraic curve)**.** An algebraic curve $C$ over $\mathbb{F}$ is the set of points $[X : Y : Z] \in \mathbb{P}^2(\mathbb{F})$ that are the solutions to a homogenous polynomial equation $f(X, Y, Z) = 0$. If $f$ is of degree $d$ then we call $C$ a degree $d$ curve. In particular, third degree curves are called *cubics*.

**Definition 2.1.2** (Singular points)**.** Let $C$ be a cubic, $f$ it's defining polynomial and $f_X, f_Y, f_Z$ the derivatives of $f$ with respect to $X$, $Y$ and $Z$ respectively. We call $P = [X_P : Y_P : Z_P]$ a singular

point if

$$f(X_P, Y_P, Z_P) = f_X(X_P, Y_P, Z_P) = f_Y(X_P, Y_P, Z_P) = f_Z(X_P, Y_P, Z_P) = 0. \tag{2.1}$$

**Definition 2.1.3** (Smooth curves)**.** An algebraic curve without singular points is called smooth.

**Definition 2.1.4** (Elliptic curve)**.** On the projective plane $\mathbb{P}^2(\mathbb{F})$, an elliptic curve $E$ is a smooth cubic over $\mathbb{F}$ together with a designated point $\mathcal{O}$ on the curve called the identity.

**Definition 2.1.5** (Short Weierstraß form)**.** When the characteristic of the field $\mathbb{F}$ is different from 2 or 3 we may without loss of generality (see [Sil09]) suppose that the elliptic curve $E$ is defined by the short Weierstraß equation in projective coordinates:

$$Y^2Z = X^3 + aXZ^2 + bZ^3 \tag{2.2}$$

with the identity point being $\mathcal{O} = [0:1:0]$. The curve equation can be written in affine coordinates as

$$y^2 = x^3 + ax + b. \tag{2.3}$$

The curve is smooth if and only if $4a^3 + 27b^2 \neq 0$.

**Theorem 2.1.1** (Group structure)**.** An elliptic curve $E$ has a natural group law, known as the *chord-and-tangent* rule. Given two points $P$ and $Q$ we can define the point $P + Q$ as follows:

1 Let $L$ be the line through $P$ and $Q$, or the tangent line if $P = Q$.
2 Let $S$ be the other point of intersection of $E$ with $L$.
3 Let $L2$ be the line through $S$ and $\mathcal{O}$, or the tangent line if $S = \mathcal{O}$.
4 Let $R$ be the other point of intersection of $E$ with $L2$.

Then $P + Q := R$. This operation defines an Abelian group structure on $E$.

*Proof.* See [Sil09]. □

*Remark.* For any field extension $\mathbb{F} \subseteq \mathbb{K}$ we denote with $E(\mathbb{K})$ the group of points $P \in \mathbb{P}^2(\mathbb{K})$ which satisfy the curve equation (2.2).

**Definition 2.1.6** ($E$ as an $\mathbb{Z}$-module)**.** The group structure makes $E$ a $\mathbb{Z}$-module, giving a meaning to multiplication of elements of $E$ by integers. We make the following definition for all $P \in E$, $n \in \mathbb{N}$:

$$[n]P := \underbrace{P + P + \ldots + P}_{n \text{ times}} \tag{2.4}$$

$$[-n]P := -([n]P) \tag{2.5}$$

**Definition 2.1.7** ($n$-torsion)**.** We denote by $E[n]$ the $n$-torsion of the elliptic curve $E$ over $\overline{\mathbb{F}}$, e.g.,

the subgroup of points of $E(\overline{\mathbb{F}})$ such that $[n]P = \mathcal{O}$:

$$E[n] := \{P \in E(\overline{\mathbb{F}}) \mid [n]P = \mathcal{O}\} \tag{2.6}$$

**Theorem 2.1.2** (Torsion subgroups structure)**.** The $n$-torsion is completely characterised, and we have for $\text{char}(\mathbb{F}) \not| n$

$$E[n] \cong \mathbb{Z}/(n) \oplus \mathbb{Z}/(n), \tag{2.7}$$

while if $\text{char}(\mathbb{F}) = p$ there are two cases

$$E[p^r] \cong \mathbb{Z}/(p^r) \qquad \text{for all } r \in \mathbb{N}, \text{ or} \tag{2.8}$$
$$E[p^r] \cong \{\mathcal{O}\} \qquad \text{for all } r \in \mathbb{N}. \tag{2.9}$$

*Proof.* See [Sil09]. □

**Definition 2.1.8** (Ordinary and supersingular curves)**.** Let $E$ be a curve over $\mathbb{F}$, and let $\text{char}(\mathbb{F}) = p$. Then

$$E \text{ is ordinary if } E[p^r] \cong \mathbb{Z}/(p^r), \tag{2.10}$$
$$E \text{ is supersingular if } E[p^r] \cong \{\mathcal{O}\}. \tag{2.11}$$

**Definition 2.1.9** (Isogenies)**.** An isogeny over $\mathbb{F}$ is a non-constant rational map defined over $\mathbb{F}$ from an elliptic curve $E$ to an elliptic curve $E'$ that is also a group morphism from $E(\mathbb{F})$ to $E'(\mathbb{F})$.

*Remark.* All non-constant rational maps sending $\mathcal{O} \in E$ to $\mathcal{O}' \in E'$ are isogenies (see [Sil09]).

**Definition 2.1.10** (Isogeny degree)**.** The degree of a separable[1] isogeny $\phi : E \to E'$ is the order of the kernel subgroup

$$\ker\phi := \{P \in E(\overline{\mathbb{F}}) \mid \phi(P) = \mathcal{O}'\} \tag{2.12}$$

**Definition 2.1.11** (The $j$-invariant)**.** Klein's $j$-invariant of an elliptic curve $E$ defined by equation (2.2) is defined to be:

$$j(E) := 1728\frac{4a^3}{4a^3 + 27b^2}. \tag{2.13}$$

The smoothness condition of the curve equation assures that the denominator is not zero.

*Remark.* Two elliptic curves are isomorphic if and only if they have the same $j$-invariant [Sil09]. Furthermore for any $j_0 \in \mathbb{F}$ there is an elliptic curve $E$ such that $j(E) = j_0$. For $j_0 = 0$ the curve is $y^2 = x^3 + 1$, for $j_0 = 1728$ the curve is $y^2 = x^3 + x$ and for other values the curve is defined the short Weierstraß equation with $a = b = \frac{27j_0}{4(1728-j_0)}$.

[1] The only type of isogeny used in SIKE

**Definition 2.1.12** (Isogenous curves)**.** Two curves are said isogenous if there is an isogeny between them. If the degree of the isogeny is $l$, the curves are called $l$-isogenous. We say that the curves are isogenous over $\mathbb{F}$ if the isogeny is defined over $\mathbb{F}$.

Every separable isogeny has a finite kernel and is uniquely determined by it. More precisely, every finite subgroup $G$ of $E$ gives rise to a unique isogeny $\phi : E \to E'$ with $\ker\phi = G$, where $E'$ is uniquely defined up to an isomorphism. We denote the image curve by $E/G$, so

$$\phi : E \to E' =: E/\ker\phi. \tag{2.14}$$

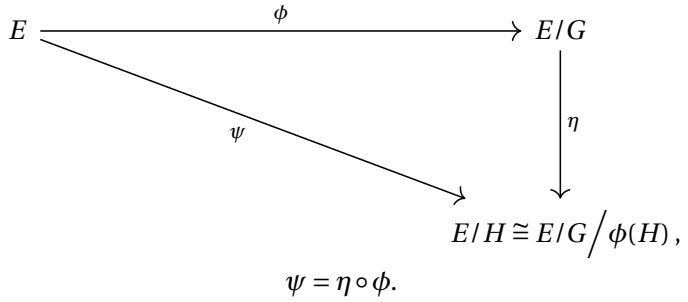
*Remark* (Computing isogenies). Given an isogeny $\phi$ as a rational map, one can compute $\ker\phi$ in time polynomial in the degree of the isogeny by means of a root finding algorithm.

The converse is also true – given a finite group $G$, represented for example by a (pair of) group generator(s), one may compute the associated isogeny using Vélu's formulas [Vél71] in time polynomial in $\#G$.

*Remark.* Every isogeny of degree $d = \prod p_i$ can be written as a composition of isogenies of degrees $p_i$. This property can be used to evaluate large smooth degree isogenies, which is one of the main elements of SIKE.

**Theorem 2.1.3** (Composing isogenies)**.** If we have finite subgroups $G \subseteq H \subseteq E$ giving rise to isogenies $\phi : E \to E/G$ and $\psi : E \to E/H$ then there is a unique isogeny $\eta : E/G \to E/H$ such that the following diagram commutes:

Figure 2.1: Composition of isogenies.




The kernel of $\eta$ is isomorphic to $H/G$ and equal to $\phi(H) \subseteq E/G$.

*Proof.* See [Sil09]. □

**Theorem 2.1.4** (Dual isogeny)**.** Let $E, E'$ be elliptic curves, and $\phi : E \to E'$ an isogeny of degree $d$. Then there exists a unique isogeny $\hat{\phi} : E' \to E$ such that $\hat{\phi} \circ \phi = [d]$ on $E$, and $\phi \circ \hat{\phi} = [d]$ on $E'$, and $\deg(\phi) = \deg(\hat{\phi})$.

*Proof.* See [Sil09]. □

**Definition 2.1.13** ($\ell$-isogeny graph)**.** An isogeny graph is a graph of isomorphism classes of isogenous elliptic curves connected by $\ell$-isogenies. More precisely, every node in the graph represents an isomorphism class of elliptic curves, and edges represent isogenies of degree $\ell$. Due to Theorem 2.1.4, the graph can be made an undirected graph, where the edge assigned to an isogeny and its dual are considered equivalent.

**Theorem 2.1.5.** Every supersingular elliptic curve $E$ over $\overline{\mathbb{F}}_p$ satisfies $j(E) \in \mathbb{F}_{p^2}$. In other words $E$ is isomorphic to a curve defined over $\mathbb{F}_{p^2}$.

*Proof.* See [Sil09]. □

Theorem 2.1.5 shows that there is only a finite number of supersingular elliptic curves over $\overline{\mathbb{F}}_p$ since the number of possible $j$-invariants is bounded by $p^2$. The exact number of supersingular elliptic curves over $\overline{\mathbb{F}}_p$ is actually lower, and given by the following theorem.

**Theorem 2.1.6.** The number of isomorphism classes of supersingular elliptic curves over $\mathbb{F}_{p^2}$ is

$$\left\lfloor \frac{p}{12} \right\rfloor + \begin{cases} 0 & \text{if } p \underset{12}{\equiv} 1, \\ 1 & \text{if } p \underset{12}{\equiv} 5,7, \\ 2 & \text{if } p \underset{12}{\equiv} 11. \end{cases} \tag{2.15}$$

*Proof.* See [Sil09]. □

The $\ell$-isogeny graph of supersingular curves over $\mathbb{F}_{p^2}$ has some nice combinatoric properties. Before we proceed, we recall a couple of graph-theoretic definitions and theorems.

**Definition 2.1.14.** Let $G = (V, E)$ be a $k$-regular graph with $n$ nodes and let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of the associated adjacency matrix. The adjacency matrix is symmetric and real, so its eigenvalues are real, which without loss of generality we can supposed to be ordered. Then, the eigenvalues satisfy

$$k = \lambda_1 \geq \ldots \geq \lambda_n \geq -k. \tag{2.16}$$

**Definition 2.1.15.** Let $G$ be a $k$-regular graph with $\lambda_i$ eigenvalues of the associated adjacency matrix. Then $G$ is called an $\varepsilon$-expander if $\varepsilon < 1$ and

$$|\lambda_i| \leq \varepsilon k \text{ for all } i \neq 1. \tag{2.17}$$

Expander graphs are widely used and have many interesting properties. In particular they have a small diameter and random walks on $\varepsilon$-expanders are rapidly mixing.

**Theorem 2.1.7** (Mixing theorem)**.** Let $G = (V, E)$ be a $k$-regular $\varepsilon$-expander. Let $S \subseteq V$ be a subset of vertices and let $x \in V$ be any vertex in $V$. Then a random walk of length at least

$$\frac{\log((2\#V)/\#S^{1/2})}{\log(1/\varepsilon)} \tag{2.18}$$

starting from $x$ will land in $S$ with probability at least $\frac{1}{2}\frac{\#S}{\#V}$.

*Proof.* See [JMV09]. □

The smaller the $\varepsilon$ value in an expander graph is, the shorter the graph diameter, and random walks are more rapidly mixing. However, the size of $\varepsilon$ is bounded from below as the next theorem shows.

**Theorem 2.1.8** (Alon-Boppana bound)**.** Let $G$ be a $k$-regular graph on $n$ vertices with diameter $m$. Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ be the eigenvalues of the adjacency matrix. Then

$$\lambda_2 \geq 2\sqrt{k-1} - \frac{2\sqrt{k-1}-1}{\lfloor m/2 \rfloor}, \tag{2.19}$$

or in other words $\lambda_2 \geq 2\sqrt{k-1} - o(1)$ for $n \to \infty$.

*Proof.* See [Nil91]. □

**Corollary 2.1.8.1.** Following Theorem 2.1.8 and formula 2.17 we have that all $\varepsilon$-expanders must satisfy $\varepsilon \geq 2\frac{\sqrt{k-1}}{k} - o(1)$.

There is a family of expander graphs that have optimal expansion properties,i.e., satisfy the bound in Theorem 2.1.8 sharply. These graphs are called Ramanujan graphs and are defined as follows.

**Definition 2.1.16** (Ramanujan graphs)**.** Let G be a $k$-regular graph. Then G is called a Ramanujan graph if it satisfies

$$|\lambda_i| \leq 2\sqrt{k-1} \text{ for all } i \neq 1. \tag{2.20}$$

Ramanujan graphs are $\varepsilon$-expanders as can be seen by setting $\varepsilon = 2\frac{\sqrt{k-1}}{k}$.

Finally this brings us to the main theorem on the structure of $\ell$-isogeny graphs of supersingular elliptic curves which proves rapid mixing of random walks along the edges of the graph.

**Theorem 2.1.9** (Pizer [Piz90])**.** For every $p \geq 5$ the graph of $\ell$-isogenies of supersingular elliptic curves over $\overline{\mathbb{F}}_p$ for any $\ell$ coprime to $p$ is a connected and $\ell + 1$-regular graph. Moreover it is a Ramanujan graph.

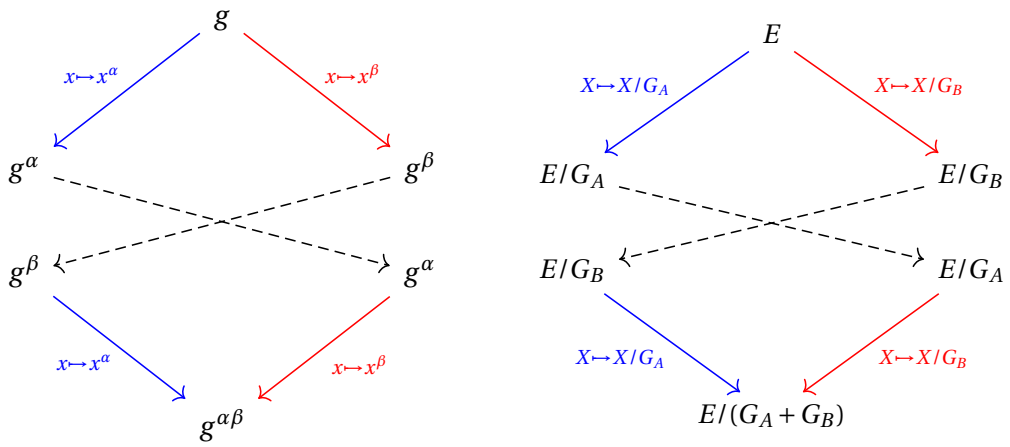
## 2.2 SIDH

Supersingular isogeny Diffie-Hellman, or shorthand SIDH, is a Diffie-Hellman-like key agreement protocol. It is the main building block of SIKE, and in fact we can consider SIKE to be SIDH with a Fujisaki-Okamoto [FO99] wrapper.

SIDH is a protocol in which two parties, call them Alice and Bob, communicate over an integrity-preserving and authenticated channel in such a manner that at the end of the protocol they agree on a shared secret.

The difference from Diffie-Hellman where the two parties compute secret exponentiations which may be represented as random walks in a group is that in SIDH the two parties compute secret isogenies which are represented as quasi-random walks in the graph of isogenous supersingular elliptic curves.

Figure 2.2: Comparison of group Diffie-Hellman and SIDH (simplified).




### 2.2.1 Public parameters

All algebraic computations in SIDH take place in a finite field $\mathbb{F}_{p^2}$, where $p$ is a prime of the form $p = \ell_A^{e_A}\ell_B^{e_B} - 1$, with $\ell_A^{e_A} \approx \ell_B^{e_B} \approx \sqrt{p}$, with $\ell_A \neq \ell_B$ small primes (in practise taken to be 2 and 3).

Curves that are used in SIDH are of cardinality $(p+1)^2$, supersingular, and they all satisfy

$$E(\mathbb{F}_{p^2}) \cong (\mathbb{Z}/(p+1))^2 \cong (\mathbb{Z}/(\ell_A^{e_A}))^2 \oplus (\mathbb{Z}/(\ell_B^{e_B}))^2. \tag{2.21}$$

Since $E[\ell_A^{e_A}] \subseteq E(\mathbb{F}_{p^2})$ and $E[\ell_B^{e_B}] \subseteq E(\mathbb{F}_{p^2})$, the elliptic curve is exactly the product of the $\ell_A^{e_A}$ and $\ell_B^{e_B}$ torsion. Let $P_A, Q_A$ be generators of the $\ell_A^{e_A}$-torsion on $E$, and let $P_B, Q_B$ be generators of the $\ell_B^{e_B}$-torsion on $E$, so

$$\langle P_A, Q_A \rangle = E[\ell_A^{e_A}], \quad \langle P_B, Q_B \rangle = E[\ell_B^{e_B}]. \tag{2.22}$$

Call $E_0$ an elliptic curve with the above properties, for example the elliptic curve with $j$-invariant $j(E_0) = 1728$ when $p \underset{4}{\equiv} 3$.

Finally, the public parameters are a sextuple containing the prime $p$, the starting curve $E_0$, and the $\ell_A^{e_A}$ and $\ell_B^{e_B}$-torsion generators $(P_A, Q_A)$ and $(P_B, Q_B)$:

$$(p, E_0, P_A, Q_A, P_B, Q_B). \tag{2.23}$$

From this point onward let $I \in \{A, B\}$ represent one party and $J \in \{A, B\} \setminus \{I\}$ the other party.

### 2.2.2 Secret key computation

The secret keys of both parties are secret subgroups of $E(\mathbb{F}_{p^2})$. The subgroups are created as follows. Party $I$ chooses a pair of numbers

$$m_I, n_I \in \left\{0, \ldots, \ell_I^{e_I} - 1\right\} \tag{2.24}$$

such that they are not both multiples of $\ell_I$. Then they compute a secret point $R_I$ as follows:

$$R_I = [m_I]P_I + [n_I]Q_I. \tag{2.25}$$

Point $R_I$ is of order $\ell_I^{e_I}$. We define $G_I = \langle R_I \rangle$ to be the cyclic group generated by the secret point. This cyclic group is the secret key. It can be represented as the generating point $R_I$ or the pair $(m_I, n_I)$.

It should be noted that the points $R_I$ and $[\xi_I]R_I$ with $\xi_I \in \mathbb{Z}/(\ell_I^{e_I})^\times$ generate the same group $G_I$. Therefore the pair $(m_I, n_I)$ is defined up to multiplication by an element of $\mathbb{Z}/(\ell_I^{e_I})^\times$. In practice this is avoided by setting $m_I = 1$, and choosing $n_I$ uniformly at random from $\mathbb{Z}/(\ell_I^{e_I})$, so the secret key of party $I$ is simply $n_I$.

The secret group $G_I$ is a randomly sampled cyclic subgroup of $E[\ell_I^{e_I}]$ of order $\ell_I^{e_I}$. The group $G_I$ uniquely defines a secret isogeny $\phi_I$ of degree $\ell_I^{e_I}$ and kernel $G_I$.

### 2.2.3 Public key computation

Both parties compute their secret isogeny $\phi_I : E_0 \mapsto E_0/G_I$, and evaluate the isogeny on the points $P_J, Q_J$, obtaining their public key:

$$pk_I = (E_0/G_I, \phi_I(P_J), \phi_I(Q_J)). \tag{2.26}$$

A straightforward use of Vélu's formulas to compute $\phi_I$ is inefficient. Instead, more efficient algorithms are used where the isogeny is computed as a composition of $e_I$ isogenies of degree $\ell_I$.

The procedure is given in Section 2.5.5.

### 2.2.4 Shared secret computation

After sharing the corresponding public keys, the two parties compute another isogeny, but this time starting from the other party's public key curve, and using that other party's public key points.

Let us rename the elements of the public key of party $J$ as

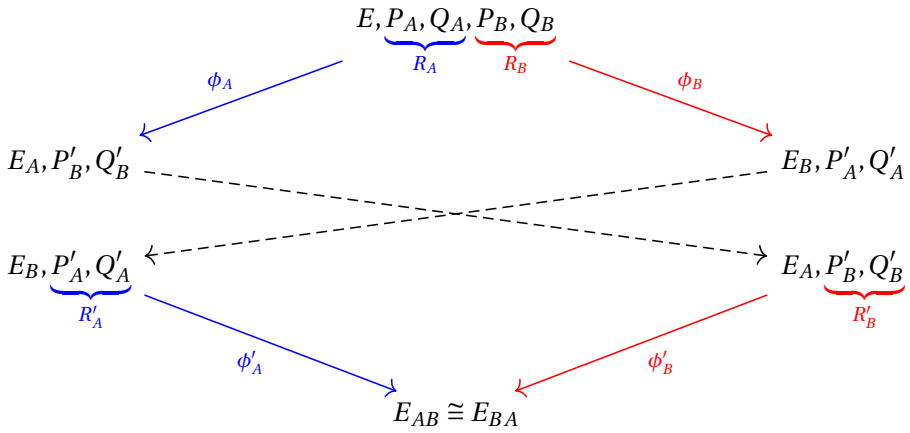
$$E_J, P'_I, Q'_I := E_0/G_J, \phi_J(P_I), \phi_J(Q_I). \tag{2.27}$$

The party $I$ first computes the kernel generating point of the isogeny as

$$R'_I = [m_I]P'_I + [n_I]Q'_I, \tag{2.28}$$

with the same method that was used in Section 2.2.2. The point $R'_I$ is of order $\ell_I^{e_I}$. The party $I$ proceeds by computing the isogeny $\phi'_I : E_J \to E_J/G'_I$ of kernel $G'_I = \langle R'_I \rangle$. Finally both parties reach the same elliptic curve $E_{IJ} = E_J/G'_I \cong E_I/G'_J = E_{JI}$, and their shared secret is the final curve's $j$-invariant.

Figure 2.3: Supersingular isogeny Diffie-Hellman




## 2.3 Hard problems

There are four main problems conjectured to be hard and on which the security of SIDH is based.

**Definition 2.3.1** (Decisional supersingular isogeny problem - DSSI)**.** Let $E$ and $E'$ be two supersingular curves over $\mathbb{F}_{p^2}$. Decide if there is an $\ell^e$-isogeny between them.

**Definition 2.3.2** (Computational supersingular isogeny problem - CSSI)**.** Let $E$ be a supersingular elliptic curve, let $P_A, Q_A$ be generators of $E[\ell_A^{e_A}]$, and let $P_B, Q_B$ be generators of $E[\ell_B^{e_B}]$ with $\ell_A, \ell_B$

coprime. Let $R_A = [m_A]P_A + [n_A]Q_A$ with $m_A, n_A$ chosen at random not both divisible by $\ell_A$. Let $G_A = \langle R_A \rangle$ and let $\phi_A : E \to E/G_A$ be the corresponding isogeny.

Given $(E, P_B, Q_B)$ and $(E/G_A, \phi_A(P_B), \phi_A(Q_B))$, find a generator of $G_A$.

**Definition 2.3.3** (Supersingular computational Diffie-Hellman problem - SSCDH)**.** Assume the conditions of Definition 2.3.2 along with the point $R_B = [m_B]P_B + [n_B]Q_B$ with $m_B, n_B$ chosen at random not both divisible by $\ell_B$, the group $G_B = \langle R_B \rangle$ and the isogeny $\phi_B : E \to E/G_B$.

Given $(E, P_A, Q_A, P_B, Q_B)$, $(E/G_A, \phi_A(P_B), \phi_A(Q_B))$, and $(E/G_B, \phi_B(P_A), \phi_B(Q_A))$, find the elliptic curve $E/(G_A + G_B)$.

**Definition 2.3.4** (Supersingular decisional Diffie-Hellman problem - SDCDH)**.** Assume the conditions of Definition 2.3.3 along with a point $R_C \in E$ chosen at random among points of order $\ell_A^{e_A}\ell_B^{e_B}$, the group $G_C = \langle R_C \rangle$ and the isogeny $\phi_C : E \to E/G_C$.

Given a tuple sampled with probability 1/2 from one of the following two distributions

1. $(E, P_A, Q_A, P_B, Q_B)$, $(E/G_A, \phi_A(P_B), \phi_A(Q_B))$, $(E/G_B, \phi_B(P_A), \phi_B(Q_A))$, $E/(G_A + G_B)$,
2. $(E, P_A, Q_A, P_B, Q_B)$, $(E/G_A, \phi_A(P_B), \phi_A(Q_B))$, $(E/G_B, \phi_B(P_A), \phi_B(Q_A))$, $E/(G_C)$,

decide from which distribution the tuple was sampled.

So far the best algorithms, classical and quantum, for solving any of the above problems are exponential.

## 2.4 SIKE

SIKE is derived from SIDH, by applying the Hofheinz-Hövelmanns-Kiltz variant of the Fujisaki-Okamoto (FO) transform [HHK17]. The reason for this choice is because the textbook SIDH protocol is insecure in the static or semi-static setting, i.e., when one of the parties has a static public key [GPST16]. The FO transform overcomes this weakness, but at a cost of a performance overhead and only allowing for semi-static keys.

The public parameters are the public parameters of SIDH

$$pp = (p, E_0, P_A, Q_A, P_B, Q_B). \tag{2.29}$$

The protocol is asymmetrical so we assume that Bob is the *server* and Alice is the *client*.

There are three phases to the protocol: public key generation, key encapsulation and key decapsulation.

### 2.4.1 Public key generation

Bob starts by choosing a random string $s \in \{0,1\}^t$ ($t > 0$ public parameter) which will be used to create a random key $K$ if he detects a cheating attempt from Alice. Then, Bob proceeds in the same way as in SIDH. The process is given in Algorithm 1.

**Algorithm 1:** PUBLIC KEY GENERATION

**Procedure** `Public key generation(`$pp$`)`

1 $sk_B \leftarrow_{\$} [0, l_B^{e_B})$
2 $s \leftarrow_{\$} \{0,1\}^t$
3 $R_B = P_B + [sk_B]Q_B$
4 Let $\phi_B : E_0 \rightarrow E_B$ be such that $\mathrm{Ker}(\phi_B) = \langle R_B \rangle$
**Output:** $pk_B = (E_B, \phi_B(P_A), \phi_B(Q_A))$

### 2.4.2 Key encapsulation

Alice generates a random message $m \in \{0,1\}^t$ ($t > 0$ public parameter) which plays the role of the secret in the following. Then, Alice computes her private key by setting

$$sk_A = G(m \,||\, pk_B) \mod \ell_A^{e_A}, \tag{2.30}$$

where $G$ is a public cryptographic hash function (in practice, SHAKE256 is used). She proceeds by computing her secret key $R_A = P_A + [sk_A]Q_A$ and her public key $c_0 = pk_A = (E_A, \phi_A(P_B), \phi_A(Q_B))$. Alice continues by computing the common secret $j(E_{BA})$ from Bob's public key. She sets $c_1 = F(j(E_{BA})) \oplus m$, where $F$ is a cryptographic hash function that may or may not be different from $G$.

Finally, Alice sends the concatenation of $c_0$ and $c_1$ as ciphertext (i.e., $ct = c_0 \,||\, c_1$) to Bob and computes the key $K$ to be used as $K = H(m \,||\, ct)$ (where $H$ is yet another cryptographic hash function which can be the same as $G$ or $F$).

**Algorithm 2:** KEY ENCAPSULATION

**Procedure** `Key encapsulation(`$pp, pk_B$`)`

$m \leftarrow \{0,1\}^t$
$sk_A = G(m \,||\, pk_B) \mod \ell_A^{e_A}$
$R_A = P_A + [sk_A]Q_A$
Let $\phi_A : E_0 \rightarrow E_A$ be such that $\text{Ker}(\phi_A) = \langle R_A \rangle$
$pk_A = (E_A, \phi_A(P_B), \phi_A(Q_B))$
$R'_A = \phi_B(P_A) + [sk_A]\phi_B(Q_A)$
Let $\phi'_A : E_B \rightarrow E_{BA}$ be such that $\text{Ker}(\phi'_A) = \langle R'_A \rangle$
$c_0 = pk_A$
$c_1 = F(j(E_{BA})) \oplus m$
$K = H(m \,||\, ct)$ `// Shared secret` $K$
**Output:** $ct = (c_0 \,||\, c_1)$

### 2.4.3 Key decapsulation

After receiving $ct = (c'_0 \,||\, c'_1)$, Bob sets $pk'_A := c'_0$, computes $j(E'_{AB})$, and extracts $m' = F(j(E'_{AB})) \oplus c'_1$ as shown in Algorithm 3. Bob then computes

$$sk'_A = G(m' || pk_B) \mod l_A^{e_A},$$

and proceeds by computing the corresponding public key $pk''_A$. Bob then checks that $pk''_A = pk'_A$, to confirm the truthfulness of Alice. In case the check passes, he sets $K = H(m' \,||\, ct)$, and $K = H(s \,||\, ct)$ otherwise.

**Algorithm 3:** KEY DECAPSULATION

**Procedure** `Key decapsulation(`$ct$`)`

$(E'_A, P'_B, Q'_B) = c'_0$
$R'_B = P'_B + [sk_B]Q'_B$
Let $\phi'_B : E'_A \rightarrow E'_{AB}$ be such that $\text{Ker}(\phi'_B) = \langle R'_B \rangle$
$m' = F(j(E'_{AB})) \oplus c'_1$
$sk'_A = G(m' \,||\, pk_B) \mod \ell_A^{e_A}$
$R' = P_A + [sk'_A]Q_A$
Let $\phi' : E_0 \rightarrow E''_A$ be such that $\text{Ker}(\phi') = \langle R' \rangle$
$pk''_A = (E''_A, \phi'(P_B), \phi'(Q_B))$
**if** $pk''_A = c_0$ **then**
  $K \leftarrow H(m' \,||\, ct)$
**else**
  $K \leftarrow H(s \,||\, ct)$
**Output:** $K$

A full execution of the SIKE algorithm is shown in Figure B.1.

## 2.5 Practical implementation of SIKE

Currently, SIKE is one of the alternate candidates in the third round of the NIST standardisation process for post-quantum cryptography. The official implementation of SIKE, and the official submission to the standardisation process, can be found in [JAC+17].

In this section we give the parameters and formulas used in the practical implementation of SIKE.

### 2.5.1 Public parameters

For all sets of public parameters, the numbers $\ell_A$ and $\ell_B$ are selected to be 2 and 3 respectively, for efficiency reasons. The starting elliptic curve is

$$E_0 : y^2 = x^3 + 6x^2 + x, \tag{2.31}$$

and the starting points $P_A, Q_A, P_B, Q_B$ are determined by a deterministic process explained in [JAC+17] and hard-coded.

Figure 2.4: Public parameters of SIKE

| prime size (bits) | $e_A$ | $e_B$ | classical security | quantum security |
|---|---|---|---|---|
| 434 | 216 | 137 | 108 | 72 |
| 503 | 250 | 159 | 125 | 83 |
| 610 | 305 | 192 | 152 | 101 |
| 751 | 372 | 239 | 187 | 125 |

### 2.5.2 Choice of representations

**Elliptic curves.** The elliptic curves used in SIKE are Montgomery curves [Mon87]. An elliptic curve $E$ over $\mathbb{F}$ is a Montgomery curve if it is represented by the following formula

$$E : by^2 = x^3 + ax^2 + x, \tag{2.32}$$

for some coefficients $a, b \in \mathbb{F}$ such that $b(a^2 - 4) \neq 0$. The point $\mathcal{O} = [0 : 1 : 0]$ is assumed to be the identity. In SIKE the coefficient $b$ is set to be equal to 1. Note that $b$ only defines the curve $E$ up to it's twist.

Define the projective representation of $a$ as $[A : C] = [a : 1]$. In addition to that set $A_{24}^+ = A + 2C$, $A_{24}^- = A - 2C$, and $C_{24} = 4C$. In SIKE, the elliptic curve with coefficient $a$ is represented in four different ways depending on the subroutine:

- As $a$, the affine curve coefficient,
- As $[A : C]$ such that $C \neq 0$ where $a = \frac{A}{C}$,
- As $[A_{24}^+ : C_{24}]$ such that $C_{24} \neq 0$ where $a = (4A_{24}^+ - 2C_{24})/C_{24}$,

- As $[A_{24}^+ : A_{24}^-]$ such that $A_{24}^+ \neq A_{24}^-$ where $a = 2(A_{24}^+ + A_{24}^-)/(A_{24}^+ - A_{24}^-)$.

**Points.** Montgomery curves allow for a point $P \in E$ to be represented only by its $X$ and $Z$ coordinates; $P = [X_P : Z_P]$.

This representation defines the point up to its inverse, i.e., $[X_P : Z_P]$ defines both $P$ and $-P$. With this representation it is possible to compute multiples of $P$ ($[n]P$ for $n \in \mathbb{Z}$) without the need for the $Y$ coordinate [Mon87].

However, two points in Montgomery coordinates cannot be added without additional information. This problem is resolved by saving a pair of points $P, Q$ in a triple of points in Montgomery coordinates

$$P, Q \quad \longleftrightarrow \quad Q = [X_Q : Z_Q],\ P = [X_P : Z_P],\ Q - P = [X_{Q-P} : Z_{Q-P}]. \tag{2.33}$$

The point $P+Q$ can then be computed from $Q, P, Q-P$. Formulas are given in the next subsection.

**Public keys.** A public key in SIKE is a triple formed from an elliptic curve $E$ and two points on that curve $P$ and $Q$. In practice, this triple is saved as the triple of points $Q, P, Q-P$ in affine Montgomery coordinates, i.e., $Q = [x_Q : 1]$, $P = [x_P : 1]$, $Q - P = [x_{Q-P} : 1]$ and

$$pk = (x_Q, x_P, x_{Q-P}). \tag{2.34}$$

The curve coefficient $a$ can be extracted from the triple with the following formula

$$a = \frac{(1 - x_Q x_P - x_Q x_{Q-P} - x_P x_{Q-P})^2}{x_Q x_P x_{Q-P}} - x_Q - x_P - x_{Q-P}. \tag{2.35}$$

In addition to the public keys, the torsion generating points on the starting curve $(P_A, Q_A)$, $(P_B, Q_B)$ are also represented as triples $(x_{Q_A}, x_{P_A}, x_{Q_A - P_A})$, $(x_{Q_B}, x_{P_B}, x_{Q_B - P_B})$.

**Compressed SIKE.** In uncompressed (i.e., the previously described) SIKE, public keys are formed as triples of coordinates of elliptic curve points $pk = (x_Q, x_P, x_{Q-P})$, from which the curve coefficient $a$ is computed. Compressed SIKE is an alternative implementation of SIKE which represents the public key points $Q, P$ as elements of $\mathbb{Z}/(\ell_I^{e_I}) \oplus \mathbb{Z}/(\ell_I^{e_I})$ through a basis of the $\ell_I^{e_I}$ torsion which is deterministically chosen as a function of the underlying curve. This allows for about 40% smaller public keys, however at a cost of slower algorithms due to the necessity of computing torsion bases and computing coefficients of points in the torsion basis. In particular the overhead is 65% – 75% for key generation, 55% – 65% for key encapsulation, and 5% – 10% for key decapsulation. Throughout the paper we assume that uncompressed SIKE is used, unless stated otherwise.

### 2.5.3 Formulas

The main building blocks of SIKE are doubling and tripling formulas, differential addition, and 2-isogenies, 3-isogenies and 4-isogenies. They are components of a scalar multiplication function used to compute the secret point $R$, and the high-order isogeny computation.

**Doubling.** The doubling of a point $[X : Z]$ on the curve represented by the affine coefficient $a$ is given by the formula

$$[2][X : Z] = \left[(X^2 - Z^2)^2 : 4XZ(X^2 + aXZ + Z^2)\right]. \tag{2.36}$$

In practice the curve is represented as $[A_{24}^+ : C_{24}]$ and the doubling is given by

$$[2][X : Z] = \left[C_{24}(X^2 - Z^2)^2 : 4XZ(C_{24}(X - Z)^2 + A_{24}^+ 4XZ)\right]. \tag{2.37}$$

**Tripling.** The tripling of a point $[X : Z]$ on the curve represented by the affine coefficient $a$ is given by the formula

$$[3][X : Z] = \left[(X^4 - 6X^2Z^2 - 4aXZ^3 - 3Z^4)^2 X : (3X^4 + 4aX^3Z + 6X^2Z^2 - Z^4)^2 Z\right]. \tag{2.38}$$

In practice the curve is represented as $[A_{24}^+ : A_{24}^-]$ and the tripling is given by

$$\begin{aligned}[3][X : Z] = \big[&X(A_{24}^+(X + Z)^4 - A_{24}^-(X - Z)^4 - 2(X^2 - Z^2)(A_{24}^+(X + Z)^2 - A_{24}^-(X - Z)^2))^2 \\ &: Z(A_{24}^+(X + Z)^4 - A_{24}^-(X - Z)^4 + 2(X^2 - Z^2)(A_{24}^+(X + Z)^2 - A_{24}^-(X - Z)^2))^2\big].\end{aligned} \tag{2.39}$$

**Differential addition.** The addition of points $Q = [X_Q : Z_Q]$ and $P = [X_P : Z_P]$, assuming the point $Q - P = [X_{Q-P} : Z_{Q-P}]$ is provided as well, is given by the formula

$$[X_Q : Z_Q] + [X_P : Z_P] = \left[Z_{Q-P}(X_P X_Q - Z_P Z_Q)^2 : X_{Q-P}(X_P Z_Q - X_Q Z_P)^2\right]. \tag{2.40}$$

The differential addition formula is independent of the curve coefficient (it is implicit in the value of $Q - P$ which is curve-dependent).

2-**isogeny.** The 2-isogeny contains two parts - the isogeny computation, which computes the codomain curve of the 2-isogeny, and the isogeny evaluation, which maps points from the codomain to the domain curve. These two parts are treated separately.

The isogeny computation takes as input the kernel point $[X_2 : Z_2]$ and outputs the curve as

$$[A_{24}^+ : C_{24}] = \left[Z_2^2 - X_2^2 : Z_2^2\right]. \tag{2.41}$$

The isogeny evaluation takes as input the kernel point $[X_2 : Z_2]$ and the point to be evaluated $[X : Z]$ and maps it to

$$[X : Z] \mapsto [X(XX_2 - ZZ_2) : Z(XZ_2 - ZX_2)]. \tag{2.42}$$

3-**isogeny.** The 3-isogeny contains two parts - the isogeny computation, which computes the codomain curve of the 3-isogeny, and the isogeny evaluation, which maps points from the codomain to the domain curve. These two parts are treated separately.

The isogeny computation takes as input the kernel point $[X_3 : Z_3]$ and outputs the curve as

$$[A_{24}^+ : A_{24}^-] = \left[(3X_3 - Z_3)^3(X_3 + Z_3) : (3X_3 + Z_3)^3(X_3 - Z_3)\right]. \tag{2.43}$$

The isogeny evaluation takes as input the kernel point $[X_3 : Z_3]$ and the point to be evaluated $[X : Z]$ and maps it to

$$[X : Z] \mapsto \left[X(XX_3 - ZZ_3)^2 : Z(XZ_3 - ZX_3)^2\right]. \tag{2.44}$$

4-**isogeny.** The 4-isogeny contains two parts - the isogeny computation, which computes the codomain curve of the 4-isogeny, and the isogeny evaluation, which maps points from the codomain to the domain curve. These two parts are treated separately.

The isogeny computation takes as input the kernel point $[X_4 : Z_4]$ and outputs the curve as

$$[A_{24}^+ : C_{24}] = \left[X_4^4 : Z_4^4\right]. \tag{2.45}$$

The isogeny evaluation takes as input the kernel point $[X_4 : Z_4]$ and the point to be evaluated $[X : Z]$ and maps it to

$$\begin{aligned}[X : Z] \mapsto \big[&X(XX_4 - ZZ_4)^2(XX_4^2 + XZ_4^2 - 2ZX_4Z_4) \\ &: Z(XZ_4 - ZX_4)^2(ZX_4^2 + ZZ_4^2 - 2XX_4Z_4)\big].\end{aligned} \tag{2.46}$$

$j$-**invariant.** The $j$-invariant of a Montgomery curve $E$ represented by the coefficient $a$ is computed by first representing the curve as $[A : C]$ via

$$[A : C] = [a : 1] = [4A_{24}^+ - 2C_{24} : C_{24}] = [2(A_{24}^+ + A_{24}^-) : A_{24}^+ - A_{24}^-], \tag{2.47}$$

and then computing the $j$-invariant as

$$j(E) = \frac{256(A^2 - 3C^2)^3}{C^4(A^2 - 4C^2)} = \frac{256(a^2 - 3)^3}{(a^2 - 4)}. \tag{2.48}$$

### 2.5.4 Three-point ladder

Computation of the secret point $R = P + [sk]Q$ (we drop the naming subscript $I \in \{A, B\}$) is done via a constant-time *three-point ladder* introduced in [FLOJRH18].

The ladder takes as input a pair of points $Q, P$ represented in Montgomery coordinates, together with the difference $Q - P$, and the affine curve coefficient $a$ (which is computed from the triple via formula 2.35).

The algorithm then loops trough all bits of the secret key, and computes a conditional swap (`cswap`) and a double-and-add at each iteration. The `cswap`$(P, Q,$ `swap`$)$ is a constant-time routine which performs a swap of two points $P$ and $Q$ if the bit `swap` is equal to 1.

The double-and-add, denoted as `xDBLADD` is a routine which combines a doubling with a differential addition.

**Algorithm 4:** DOUBLE-AND-ADD

**Input:** Coefficient $a_{24}^+$, points $R_0, R_1, R_2$.
**Assumes:** An elliptic curve with $a = 4a_{24}^+ - 2$ contains the points $R_0, R_1, R_2$, and $R_2 = R_0 - R_1$.
**Output:** $([2]R_0, R_1 + R_0, R_2)$
**Procedure** `dadd`$(R_0, R_1, R_2, a_{24}^+)$

1 $[A_{24}^+ : C_{24}] = [a_{24}^+ : 1]$
2 $R_1 = R_1 + R_0$ `// Via formula 2.40, page 27.`
3 $R_0 = [2]R_0$ `// Via formula 2.37, page 27.`
**return** $(R_0, R_1, R_2)$

The three-point ladder is finally given in Algorithm 5. We follow the variable naming convention of [JAC+17].

**Algorithm 5:** THREE-POINT LADDER

**Input:** Montgomery curve $E$ with coefficient $a$, points $Q, P, Q - P$, an integer $m = \sum_{i=0}^{\ell-1} m_i 2^i$ with $m_i \in \{0, 1\}$.
**Assumes:** $Q, P, Q - P \in E$, and $P, Q - P \notin \{[1:0], [0:1]\}$.
**Output:** $P + [m]Q$.
**Procedure** `Three-point ladder`$(Q, P, Q - P, m, a)$

1 $a_{24}^+ = (a + 2)/4$
2 $m_{-1} = 0$
3 $R_0, R, R_2 \leftarrow Q, P, Q - P$
4 **for** $i = 0$ **to** $\ell - 1$ **do**
5 `cswap`$(R_2, R, m_i \oplus m_{i-1})$
6 $R_0, R_2, R \leftarrow$ `xDBLADD`$\left(R_0, R_2, R, a_{24}^+\right)$
7 `cswap`$(R_2, R, m_i)$
**return** $R$

### 2.5.5 Isogeny computation

An isogeny with a cyclic kernel can be computed in linear time in the order of the kernel via Vélu's formulas. When the order of the kernel is smooth, as is the case in SIKE, the isogeny can be computed exponentially faster. Let $R$ be a point of order $\ell^e$, and let $G = \langle R \rangle$. In this subsection we show how to compute the isogeny $\phi$ of kernel $G$ and order $\ell^e$ as a composition of isogenies of degree $\ell$.

Let $R_i := [\ell^{e-i}]R$, and let $G_i := \langle R_i \rangle$ for $i = 0, 1, \ldots, e$. The order of $R_i$ and the cardinality $G_i$ are equal to $\ell^i$. Furthermore there is an ascending filtration of $G$

$$\{\mathcal{O}\} = G_0 \subseteq G_1 \subseteq \cdots \subseteq G_{e-1} \subseteq G_e = G, \tag{2.49}$$

with $[G_{i+1} : G_i] = \ell$. This filtration, together with Theorem 2.1.3, allows one to compute the isogeny $\phi$ as the composition of small degree isogenies $\phi_i$ as shown in the following commutative diagram.

Figure 2.5: Splitting the isogeny

$$\begin{array}{ccccccccc} E & & \xrightarrow{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} & & & & & & E/G \\ \updownarrow\cong & & & & & & & & \updownarrow\cong \\ E/G & \xrightarrow{\phi_1} & E/G_1 & \xrightarrow{\phi_2} & E/G_2 & \xrightarrow{\phi_3} & \cdots & \xrightarrow{\phi_e} & E/G_e \end{array}$$

In practice $2^{e_A}$-isogenies are computed as composition of $(e_A \mod 2)$ isogenies of degree 2 and $\lfloor \frac{e_A}{2} \rfloor$ isogenies of degree 4 (for efficiency reasons), while $3^{e_B}$-isogenies are computed as a composition of $e_B$ isogenies of degree 3, so $\ell$ can be thought to be equal to 3 or 4.

#### Isogeny computation via multiply-and-push

One way to compute an $\ell^e$ isogeny generated by $\langle R \rangle$ is to compute the $\ell$-isogeny generated by $[\ell^{e-1}]R$, push the point $R$ through said $\ell$-isogeny, and continue recursively. Each time the order of the pushed kernel point decreases by a factor of $\ell$.

The curve $E_i/H_i$ and the isogeny $\phi_i$ of Algorithm 6 correspond to the curve $E/G_i$ and isogeny $\phi_i$ of Figure 2.1. The cost of Algorithm 6 is $O(e^2)$ point-multiplications by $\ell$ and $O(e)$ degree $\ell$ isogeny computations.

#### Isogeny computation via tree traversal

An $\ell^e$-isogeny can be computed in a more efficient way through a combination of $\ell$-isogenies and multiplications by $\ell$ by using a *tree traversal* algorithm guided by what SIDH calls a *strategy*.

**Algorithm 6:** DEGREE $\ell^e$ ISOGENY COMPUTATION AND EVALUATION - MULTIPLY-AND-PUSH

**Input:** Curve $E$, kernel generator $R$, evaluation points $[T_1, \ldots, T_m]$
**Assumes:** $R, T_1, \ldots, T_m \in E$ and $R$ has order $\ell^e$.
**Output:** Curve $E/\langle R \rangle$ and list of evaluations $[\phi_R(T_1), \ldots, \phi_R(T_m)]$.
**Procedure** `isog_map`$(E, R, [T_1, \ldots, T_m])$

$E_0 = E$
**for** $i = 0$ **to** $e-1$ **do**
  $K = [\ell^{e-(i+1)}]R$ `// point of order` $\ell$
  $H_i = \langle K \rangle$
  $\phi_i : E_i \mapsto E_i / H_i$
  $E_{i+1} = E_i / H_i$
  **for** $i = 1$ **to** $m$ **do**
    $T_i = \phi_i(T_i)$
  $R = \phi_i(R)$ `// point of order` $\ell^{e-(i+1)}$
**return** $E_e, [T_1, \ldots, T_m]$

**Tree traversal as a game.** A tree traversal can be modelled as a game which is played on an $n \times n$ rectangular grid, in the area between the axes and above the line $y = x - n$. The goal of the game is to reach the point $(0, n)$ starting from the point $(0, 0)$.

To this end, the player can perform two kinds of move: a vertical move downwards (corresponding to a multiplication by $\ell$), or a horizontal move rightwards (corresponding to an $\ell$-isogeny computation). Moreover, the player can move to any point that was previously visited (some computed points may be saved). Downwards moves are unrestricted, but horizontal moves to the right, i.e., from the vertical line $x = i$ to $x = i + 1$, are allowed only after the point $(i, -(n - i))$ was reached (the next curve is computed only after computing the kernel point of the corresponding $\ell$-isogeny).

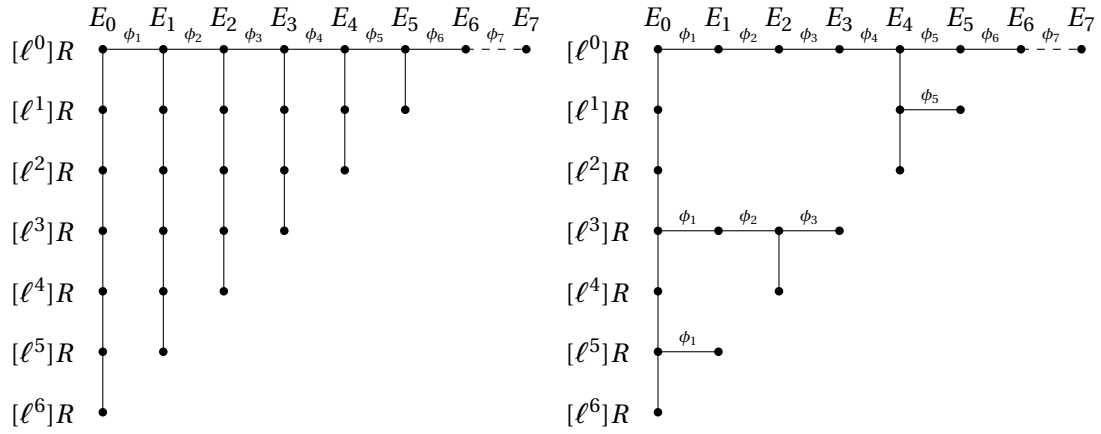
A successfully played game can therefore be represented by a graph of all the visited points, such as Figure 2.6. Such a graph represents an $\ell^e$-isogeny computation; the point at $(0, 0)$ represents the kernel generator $R$, all points on the same vertical line are on the same curve noted on the top. Reaching the point $(n, n)$ is equal to reaching the penultimate curve together with a point of order $\ell$, so just the final $\ell$-isogeny is left to reach the final curve.

**Strategy.** A strategy for computing an $\ell^e$-isogeny is defined as a tuple of $e - 1$ integers, and the set $\mathcal{S}_e$ of valid strategies is defined recursively as

$$\mathcal{S}_1 = \{()\}, \qquad \mathcal{S}_n = \left\{(b) :: S_a :: S_b \;\middle|\; a + b = n,\ S_a \in \mathcal{S}_a,\ S_b \in \mathcal{S}_b\right\}, \tag{2.50}$$

where $()$ denotes the empty tuple and $::$ denotes concatenation. Given a curve $E$, a kernel generator $R$, a list of evaluation points $[T_1, \cdots, T_m]$, and a strategy $S = (s_1, \ldots, s_{e-1})$, the isogeny computation/evaluation proceeds as described in Algorithm 7.

Figure 2.6: Visualisation of two tree traversal games ($\ell^7$-isogeny computation).




The recursive nature of the algorithm is naturally visualised as a binary tree with $e$ leaves. By stretching the tree to draw all the leaves at the same diagonal $y = x - (e-1)$, we even have a visualisation of the computational effort required to compute the isogeny: edges going downwards represent point multiplications by $\ell$, edges going to the right represent $\ell$-isogeny evaluations.

Figure 2.6 shows this representation for the strategy $S = (6,5,4,3,2,1) \in \mathscr{S}_7$, which performs 21 multiplications by $\ell$ and 6 isogeny evaluations and the strategy $S = (3,2,1,1,1,1)$ which performs 9 multiplications by $\ell$ and 11 isogeny evaluations (assuming the list $[T_1,\ldots,T_m]$ is initially empty). The former actually represents Algorithm 6 as a tree traversal game which is in general given by the strategy $S = (e-1, e-2, \ldots, 1)$.

In general, one can find an optimal strategy which computes the $\ell^e$-isogeny in $O(e\log(e))$ multiplications by $\ell$ and $O(e\log(e))$ degree $\ell$ isogenies. Furthermore, given the computational cost of a multiplication by $\ell$, and the cost of an $\ell$-isogeny, one can find, in time $O(e\log(e))$ the most efficient strategy for tree traversal [ZSP+18]. The optimal strategy for each set of SIKE parameters is hard-coded in the algorithm.

**Algorithm 7:** DEGREE $\ell^e$ ISOGENY COMPUTATION AND EVALUATION - TREE TRAVERSAL

**Input:** Curve $E$, kernel generator $R$, strategy $S = (s_1, \ldots, s_{e-1})$,
list of evaluation points $[T_1, \ldots, T_m]$.
**Assumes:** $S \in \mathcal{S}_n$, $R, T_1, \ldots, T_m \in E$ and $R$ has order $\ell^e$.
**Output:** Curve $E/\langle R \rangle$ and list of evaluations $[\phi_R(T_1), \ldots, \phi_R(T_m)]$.
**Procedure** `isog_strategy`$(E, R, S, [T_1, \ldots, T_m])$

**if** $S = ()$ **then**
  $\phi_R : E \mapsto E/\langle R \rangle$
  $E = E/\langle R \rangle$
  **for** $i = 1$ **to** $m$ **do**
    $T_i = \phi_R(T_i)$
**else**
  $K \leftarrow R$
  **for** $i = 1$ **to** $s_1$ **do**
    $K = [\ell]K$
  $S_L = (s_2, \ldots, s_{e-s_1})$
  $E, [R, T_1, \ldots, T_m] \leftarrow$ `isog_strategy`$\big(E, K, S_L, [R, T_1, \ldots, T_m]\big)$
  $S_R = (s_{e-s_1+1}, \ldots, s_{e-1})$
  $E, [T_1, \ldots, T_m] \leftarrow$ `isog_strategy`$\big(E, R, S_R, [T_1, \ldots, T_m]\big)$
**return** $E, [T_1, \ldots, T_m]$

# Part II

# The generalised Legendre pseudorandom function

# 3 Attack on the generalised Legendre pseudorandom function

The generalised Legendre pseudorandom function was originally introduced by Damgård [Dam90] as what we call a *linear Legendre PRF*, and later generalised by Russell and Shparlinski [RS04]. For a public parameter prime $p$, the function is modelled as an oracle $\mathcal{O}_f$ parametrised by a secret key $f \in \mathbb{F}_p[x]$ of degree $r$, which on input an element $x$ of $\mathbb{F}_p$ outputs $\mathcal{O}_f(x) = \left(\frac{f(x)}{p}\right) \in \{-1, 1\}$. Given a sequence of Legendre symbols of $f$ evaluated at consecutive elements, it is conjectured to be hard to predict the symbol of $f$ evaluated at the next element. Similarly, distinguishing $\mathcal{O}_f$ from a random function, or obtaining $f$ while given oracle access to $\mathcal{O}_f$ is assumed to be hard as well (see Chapter 1.3).

While mathematically simple, the Legendre PRF was considered to be too computationally expensive for traditional cryptographic settings and thus not used as a pseudorandom function. A recent result by Grassi et al. [GRR+16] showed that it can be used in multiparty-computation settings, and that in that paradigm it is orders of magnitude faster then the alternative symmetric-key primitives. There are plans to incorporate the Legendre PRF into the Ethereum 2.0 protocol [Fei19], and in order to motivate research in this direction, the Ethereum foundation published a number of challenges. In each challenge, a list of $M = 2^{20}$ Legendre symbols of consecutive elements was provided. The difficulty of the challenge was based on the size of the underlying prime field which varied from 64 to 148 bits. The goal was to find the starting element, or in other words the secret key which parametrises the linear Legendre PRF.

**Previous work.** First attacks on the Legendre PRF were devised in the quantum computing paradigm. In a series of papers [van02, vDH00, vDHI01] van Dam, Hallgreene and Ip provided efficient quantum algorithms for obtaining the secret key given quantum access to a linear Legendre PRF oracle. This was later improved on by Russell and Shparlinski [RS04] who introduced the generalised Legendre PRF, and provided a quantum algorithm for obtaining the secret key when given access to a quantum oracle. The first non-trivial classical attack on the Legendre PRF was given by Khovratovich [Kho19] who devised a memory-less collision attack of time complexity $O(p^{\frac{r}{2}})$ with unconstrained access to the oracle. Further improvements by Beullens et al. [BBUV20] and Kaluđerović et al. [KKK20b] give faster attacks on the linear Legendre PRF

when the number of oracle queries is bounded by $M$. The attack complexities are $O\left(\frac{p\log^2 p}{M^2}\right)$ and $O\left(\frac{p\log p\log\log p}{M^2}\right)$ respectively. So far there are no known polynomial time attacks when access to the oracle is classical.

**Contributions.** We analyse the action of Möbius transformations on monic polynomials of degree $r$, and use it to provide an improved attack on the Legendre pseudorandom function. When $r \geq 3$ we classify the secret key polynomials into three groups, and for the most relevant case we give an $O(p^{r-3})$ attack with $O(p^3)$ precomputation and $p - o(p)$ oracle queries. The other two cases consist of weak keys which we fully characterise while providing faster attacks. For degree $r < 3$ we give an $O(p^{r/2})$ attack with $O(p^{r/4})$ queries, and $O(\frac{p^r \log p}{M^2})$ if the number of queries is limited to $M$. These are improvements with respect to previous algorithms of a factor from $p$ up to $p^3$ in the general case, and even more for a new family of *bad* keys.

**Structure.**

In Section 3.1 we introduce the Möbius group and define the action of group elements on monic polynomials. We then show how this action partitions the key space into three sets, which we call *good*, *bad* and *ugly* keys.

Section 3.2 provides the Algorithm for solving the generalised Legendre symbol problem. The section is divided into three parts providing a different algorithm for each of the three key types, and towards the end provides alternative algorithms for low degree polynomials.

Section 3.4 gives algorithms in the case when the number of queries to the oracle is limited. We also provide practical comparisons with other known algorithms.

Finally we discuss possible future directions and give some security recommendations.

## 3.1 Möbius transformations

Invertible rational projective transformations of the projective space, also known as Möbius transformations, act naturally on rational functions of $\mathbb{P}^1$, changing the argument and preserving their degrees. We show how this action can be exploited in order to connect oracles of monic polynomials that are in the same orbit under the action of the group of Möbius transformations.

**Definition 3.1.1** (Möbius transformations)**.** We call $\mathscr{M}$ the group of $\mathbb{F}_p$-rational automorphisms of $\mathbb{P}^1$. This group is isomorphic to the projective linear group of degree two $\mathrm{PGL}_2(\mathbb{F}_p)$ which has order $p^3 - p$. The elements of $\mathscr{M}$ are called Möbius transformations. Given a matrix $m = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in$

$\mathrm{PGL}_2(\mathbb{F}_p)$ there is a unique Möbius transformation $\varphi_m$ given by

$$\begin{aligned} \varphi_m : \mathbb{P}^1 &\longrightarrow \mathbb{P}^1 \\ [x:y] &\longmapsto [ax+by : cx+dy], \end{aligned} \tag{3.1}$$

and function composition satisfies $\varphi_{m_1} \circ \varphi_{m_2} = \varphi_{m_1 m_2}$. We drop the notation of $\varphi_m$ and only use $m$ from now on.

Recall a classical Lemma on Möbius transformations.

**Lemma 3.1.1.** Given a set of three distinct points $x_1, x_2, x_3 \in \mathbb{P}^1$ and another set of three distinct point $y_1, y_2, y_3 \in \mathbb{P}^1$ there is a unique Möbius transformation $m$ which sends $x_i$ to $y_i$ for $i = 1,2,3$.

*Proof.* We will show that there always exists a transformation $n$ such that $n(x_1) = [0:1]$, $n(x_2) = [1:1]$, $n(x_3) = [1:0]$.

If $x_i$ are all affine points with $x_i = [x_i' : 1]$, then

$$n = \begin{pmatrix} x_2' - x_3' & -x_1'(x_2' - x_3') \\ x_2' - x_1' & -x_3'(x_2' - x_1') \end{pmatrix}.$$

If $x_i = [1:0]$ then the associated matrix is $n_i$:

$$n_1 = \begin{pmatrix} 0 & x_2' - x_3' \\ 1 & -x_3' \end{pmatrix}, \qquad n_2 = \begin{pmatrix} 1 & -x_1' \\ 1 & -x_3' \end{pmatrix}, \qquad n_3 = \begin{pmatrix} 1 & -x_1' \\ 0 & x_2' - x_1' \end{pmatrix}.$$

The transformation which sends $x_i$ to $y_i$ is the composotion of the mapping which sends $x_i$ to $[0:1], [1:1], [1:0]$ with the inverse of the mapping which sends $y_i$ to $[0:1], [1:1], [1:0]$.

Finally, uniqueness follows from the fact that the only map acting as identity on $[0:1], [1:1]$ and $[1:0]$ is the identity itself, which is proved by analysing the formula 3.1. □

**Corollary 3.1.1.1.** It follows from Lemma 3.1.1 that if a Möbius transformation fixes three points, then it must be the identity.

Möbius transformation have a natural action on monic polynomials which is introduced in the following definition.

**Definition 3.1.2** (Action of $\mathscr{M}$ on monic polynomials)**.** The action of a Möbius transformation $m = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in \mathscr{M}$ on a polynomial $f$ is denoted by $m \cdot f = f_m$ and defined as

$$m \cdot f = f_m(x) := \begin{cases} f\left(\frac{ax+b}{cx+d}\right) \frac{(cx+d)^r}{f\left(\frac{a}{c}\right) c^r} & \text{if } c \neq 0, \\ f\left(\frac{ax+b}{cx+d}\right) \left(\frac{d}{a}\right)^r & \text{if } c = 0. \end{cases} \tag{3.2}$$

The corrective factors $(cx+d)^r$ and $f\left(\frac{a}{c}\right)c^r$ (resp. $\left(\frac{d}{a}\right)^r$) are introduced in order to make $f_m$ a polynomial and to make it monic correspondingly.

**Lemma 3.1.2** (Action of $\mathscr{M}$ on roots of polynomials)**.** Let $f$ be a monic polynomial. If $\alpha$ is a root of $f$ then $m^{-1}(\alpha)$ is a root of $f_m$, where $m^{-1} = \left(\begin{smallmatrix} d & -b \\ -c & a \end{smallmatrix}\right)$ is the inverse of the Möbius transformation $m$. Thus, if $f(x) = \prod_{i=1}^{r}(x-\alpha_i)$ then

$$f_m(x) = \prod_{i=1}^{r}(x - m^{-1}\alpha_i) = \prod_{i=1}^{r}\left(x - \frac{d\alpha_i - b}{-c\alpha_i + a}\right). \tag{3.3}$$

*Proof.* Both polynomials in (3.2) and (3.3) are monic of equal degree and they have matching roots with equal multiplicities. Therefore the polynomials are equal. □

We conclude that the group $\mathscr{M}$ of Möbius transformations has left (covariant) action on the roots of polynomials in $\mathbb{F}_p[x]$ and right (contravariant) action on polynomials.

### 3.1.1 Action of $\mathscr{M}$ on oracles

Suppose we are given access to $\mathscr{O}_f$, the oracle of $f$. Following (3.2) we can mimic the oracle of $f_m$ with:

$$\mathscr{O}_{f_m}(x) = \left(\frac{f_m(x)}{p}\right) = \begin{cases} \mathscr{O}_f\left(\frac{ax+b}{cx+d}\right)\left(\frac{cx+d}{p}\right)^r \mathscr{O}_f\left(\frac{a}{c}\right)\left(\frac{c}{p}\right)^r & \text{if } c \neq 0 \text{ and } cx+d \neq 0, \\ \left(\frac{ax+b}{p}\right)^r \mathscr{O}_f\left(\frac{a}{c}\right)\left(\frac{c}{p}\right)^r & \text{if } c \neq 0 \text{ and } cx+d = 0, \\ \mathscr{O}_f\left(\frac{ax+b}{cx+d}\right)\left(\frac{d}{p}\right)^r\left(\frac{a}{p}\right)^r & \text{if } c = 0. \end{cases} \tag{3.4}$$

This allows us to obtain an oracle associated to $\mathscr{O}_{f_m}$ for each $m \in \mathscr{M}$.

In order to compute $\{f_m\}_L$ the Legendre sequence associated to $f_m$, we need to query the oracle at most $L+1$ times and compute at most $L+1$ additional Legendre symbols.

### 3.1.2 Orbits of $\mathscr{M}$

The Möbius group acts on the key space of irreducible polynomials of degree $r$ in $\mathbb{F}_p[x]$. We divide the key space into orbits of $\mathscr{M}$, and characterise polynomials based on the size of the orbit they are in. The following lemma helps characterise these sets.

**Lemma 3.1.3.** Let $\mathscr{M}$ be the Möbius group and $f \in \mathbb{F}_p[x]$ an irreducible polynomial of degree $r$ with $3 \leq r < p$. Then, the stabiliser of $f$ is a cyclic group of order $r'$ for some $r' \mid r$. Furthermore $r' \mid p^2 - 1$.

*Proof.* Let $\mathrm{Stab}(f) = \{m \in \mathscr{M} \mid f = f_m\}$ be the stabiliser of $f$, and let $m \in \mathrm{Stab}(f)$. By (3.3) the roots

of $f_m$ are $m^{-1}\alpha_i$ implying that $m$ permutes the roots of $f$. Let $\mathrm{Gal}(f) = \{\phi_i := x \mapsto x^{p^i} \mid i \in \mathbb{Z}/r\}$ be the Galois group of $f$, and let $\alpha$ be any root of $f$.

Since $m$ permutes roots of $f$, we must have $m\alpha = \phi_i(\alpha)$ for some $i \in \mathbb{Z}/r$. Each root of $f$ can be written as $\phi_j(\alpha)$ for some $j \in \mathbb{Z}/r$. Then $m(\phi_j(\alpha)) = \phi_j(m\alpha) = \phi_j(\phi_i(\alpha)) = \phi_i(\phi_j(\alpha))$, the first equivalence following since $m$ is rational and it commutes with the Frobenius. Hence $m$ acts on the roots equivalently to $\phi_i$. This gives rise to a homomorphism from $\mathrm{Stab}(f)$ to $\mathrm{Gal}(f)$. This homomorphism is injective since two Möbius transformations with the same action on a set of $r \geq 3$ are equal, following from Lemma 3.1.1.

Therefore $\mathrm{Stab}(f)$ is a subgroup of $\mathrm{Gal}(f) \cong \mathbb{Z}/r$, so it is isomorphic to $\mathbb{Z}/r'$ for some $r' \mid r$. The stabiliser is naturally a subgroup of $\mathscr{M}$, so its order divides $\#\mathscr{M} = p(p^2-1)$. Since $r' < p$ we have $r' \mid p^2 - 1$. □

We divide the key space of monic square-free polynomials of degree $r$ into three subsets depending on their reducibility and the size of their orbit.

**Definition 3.1.3** (*Good* polynomials)**.** A polynomial of degree $r \geq 3$ is called *good* if it is irreducible and one of the following equivalent properties is true

- The $\mathscr{M}$-stabilizer of $f$ is trivial (reduced to the neutral element).
- The $\mathscr{M}$-orbit of $f$ is of size $p^3 - p$.

**Definition 3.1.4** (*Bad* polynomials)**.** A polynomial of degree $r \geq 3$ is called *bad* if it is irreducible and one of the following equivalent properties is true

- The $\mathscr{M}$-stabilizer of $f$ is of size $r' > 1$.
- The $\mathscr{M}$-orbit of $f$ is of size $\frac{p^3-p}{r'}$ with $r' > 1$.

**Definition 3.1.5** (*Ugly* polynomials)**.** A polynomial of degree $r \geq 3$ is called *ugly* if it is reducible.

Algorithms for *good*, *bad* and *ugly* polynomials are different, so the three types of polynomials are treated individually. Furthermore the polynomials of degree $r < 3$ are handled separately.

## 3.2 Algorithm for $r \geq 3$

We give an algorithm for solving the Generalised Legendre Symbol Problem for polynomials of degree $r \geq 3$. The general idea is to do a table-based collision search. The algorithm is divided into two stages – the precomputation stage and the search stage.

In the precomputation stage we create a table containing $\{f_m\}_L$ for many $m \in \mathscr{M}$ together with descriptions for the $m$'s. This is done by using the oracle $\mathscr{O}_f$ and exploiting property (3.4).

In the search stage we do a collision search and compute random polynomials $g$ until $\{g\}_L = \{f_m\}_L$ for some $m$ in the table. This allows us to compute $f$ as $f = g_{m^{-1}}$.

The tables and the trials differ for different polynomial types, so we give three separate algorithms for *good*, *bad* and *ugly* polynomials.

All algorithms start by querying the oracle $\mathcal{O}(x)$ at all $x \in \mathbb{F}_p$, and computing $\left(\frac{x}{p}\right)$ for all $x \in \mathbb{F}_p$. These results are then saved in a table and whenever we need an oracle query or a Legendre symbol we read them instead of computing an expensive symbol or querying the oracle multiple times.

### 3.2.1 Good polynomials algorithm

Recall that $f$ is *good* if it is an irreducible polynomial of degree $r \geq 3$ and the stabiliser of $f$ is trivial or the $\mathcal{M}$-orbit of $f$ has size $p^3 - p$.

#### Precomputation

In the precomputation stage we generate a table $T$ containing $\{f_m\}_L$ and a description of $m$ for all Möbius transformations $m$ as described in Section 3.1.1. Since $f$ is *good*, the table $T$ contains $p^3 - p$ different sequences.

#### Search

The search is done by trying random $g(x)$ of degree $r$ and computing $\{g\}_L$ until we find a collision, which we expect to find after $O(p^{r-3})$ trials. For each trial $g$ is evaluated at $L$ points, and $L$ Legendre symbols are extracted, so the run-time can be measured in the number of Legendre symbols extracted which is $O(p^{r-3}L)$.

### 3.2.2 Bad polynomials algorithm

Recall that $f$ is *bad* if it is an irreducible polynomial of degree $r \geq 3$ and the stabiliser of $f$ is non-trivial or the $\mathcal{M}$-orbit of $f$ has size $\frac{p^3-p}{r'}$ for some $r' > 1$. It follows from Lemma 3.1.3 that $\mathrm{Stab}(f)$ is isomorphic to $\mathbb{Z}/r'$.

The first step is to find $\mathrm{Stab}(f)$, the stabiliser of $f$. A straightforward way to find it in $O(p^3)$ is by enumerating $\mathcal{M}$ and isolating the matrices that fix $f$. Appendix A.1 describes a non-trivial way to find it in $O(p^2 \log r)$ steps.

Call $m$ any generator of $\mathrm{Stab}(f)$. The matrix $m$ is rational so it has a Jordan canonical form of one

of the following three types:

$$\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \qquad \begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix} \qquad \begin{pmatrix} a & 1 \\ 0 & a \end{pmatrix}$$

Rational Irrational Non-diagonal

where $a, b \in \mathbb{F}_p \setminus \{0\}$ and $\lambda, \mu \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$, conjugates of each other. We can exclude non-diagonal matrices since they have order $p$, while $m$ has order $r' < p$.

Let $D$ be a diagonal matrix of order $r'$ and $P$ a change of basis matrix (these can be chosen uniquely from a set of representatives given in Appendix A.1) such that

$$m = P\,D\,P^{-1}.$$

The polynomial $f_P$ is stabilised by $D$ as the following formula shows

$$D \cdot f_P = (PD) \cdot f = (mP) \cdot f = P \cdot f_m = P \cdot f = f_P.$$

Therefore $f_P$ satisfies $f_P(\frac{u}{v}x)(\frac{v}{u})^r = f_P(x)$ where $(u, v) = (a, b)$ or $(\lambda, \mu)$. This sets the following constraints on the coefficients of $f_P$

$$f_P(x) = x^r + k_{r-1}x^{r-1} + \ldots + k_2x^2 + k_1x + k_0 = x^r + \sum_{i=0}^{r-1} k_i x^i$$

$$(D \cdot f_P)(x) = x^r + k_{r-1}(\tfrac{u}{v})^{r-1}x^{r-1} + \ldots + k_1(\tfrac{u}{v})x + k_0 = x^r + \sum_{i=0}^{r-1} k_i(\tfrac{u}{v})^i x^i$$

from which it follows that

$$k_i = k_i(\tfrac{u}{v})^i \text{ for } i = 0, 1, \ldots, r-1. \tag{3.5}$$

Since $\frac{u}{v}$ has order $r'$ we have $k_i = 0$ for all $i$ that are not multiples of $r'$.

The goal is to search polynomials which satisfy (3.5) in order to reduce the search space from $O(p^r)$ to $O(p^{r/r'})$.

**Precomputation**

The precomputation done by creating a table $T$ which satisfies the following two constraints:

- For each $\{t\}_L \in T$, the polynomial $t$ is in the orbit of $f$.
- For each $\{t\}_L \in T$, the polynomial $t_P$ satisfies (3.5).

The Legendre sequence $\{f\}_L$ of the polynomial $f$ is not the only sequence in $T$. There is a family of $O(p)$ polynomials $t$ in the orbit of $f$ such that $t_P$ satisfy (3.5). However, the two classes differ for the two types of polynomials, so we treat them separately.

**Rational.** When $D$ is rational, $P$ is rational too, so the polynomial $f_P$ is in the orbit of $f$. If $C$ is a rational diagonal matrix, $C \cdot f_P$ is another polynomial in the orbit of $f$ satisfying (3.5). The total number of such polynomials is $\frac{p-1}{r'}$ since matrices $C$ can be chosen up to stabiliser of $f_P$ which is $\langle D \rangle$. A set of representatives is

$$\mathscr{C}_1 = \left\{ \begin{pmatrix} g^i & 0 \\ 0 & 1 \end{pmatrix} \middle| \, g \in \mathbb{F}_p^* \text{ generator}, 0 \le i < \tfrac{p-1}{r'} \right\}. \tag{3.6}$$

We define the table $T$ to be the table containing $\{PCP^{-1} \cdot f\}_L$ together with a description of $C$ for all $C$ in $\mathscr{C}_1$. It has $\frac{p-1}{r'}$ elements, and for all polynomials $t$ in the table, $t_P$ satisfies (3.5).

**Irrational.** When $D$ is irrational, $P$ is too, so $f_P$ is not in the orbit of $f$. There are additional constraints on $f_P$ following from the rationality of $m$. For any element $x \in \mathbb{F}_p^2$ we denote with $\overline{x}$ the conjugate of $x$. We also denote with $\overline{n}$ the element-wise conjugate of $n \in \mathscr{M}$. Then:

$$m = P \begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix} P^{-1} = \overline{m} = \overline{P} \, \overline{\begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix}} \, \overline{P^{-1}} = \overline{P} \begin{pmatrix} \mu & 0 \\ 0 & \lambda \end{pmatrix} \overline{P}^{-1}.$$

Let $A_P := P^{-1}\overline{P}$. From the definition of $A_P$ and the above formulas it follows that

$$A_P^{-1} = \overline{A}_P$$
$$\begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix} A_P = A_P \begin{pmatrix} \mu & 0 \\ 0 & \lambda \end{pmatrix}.$$

These constraints imply that $A_P = \left(\begin{smallmatrix} 0 & \alpha \\ 1/\overline{\alpha} & 0 \end{smallmatrix}\right)$ for some $\alpha \in \mathbb{F}_{p^2}$. The action of $A_P$ is the same as the action of $\left(\begin{smallmatrix} 0 & s \\ 1 & 0 \end{smallmatrix}\right)$ where $s = \alpha\overline{\alpha} \in \mathbb{F}_p$. Note that $s$ can be computed and, up to choosing a different representative for $P$, can be set to be equal to 1. We further have

$$A_P \cdot f_P(x) = f_{PA_P}(x) = f_{\overline{P}}(x) = \overline{P} \cdot f(x) = \overline{P} \cdot \overline{f(x)} = \overline{P \cdot f(x)} = \overline{f_P(x)},$$

which gives new constraints on the coefficients of $f_P(x)$:

$$\overline{f_P(x)} = x^r + \overline{k}_{r-1}x^{r-1} + \ldots + \overline{k}_2 x^2 + \overline{k}_1 x + \overline{k}_0 = x^r + \sum_{i=0}^{r-1} \overline{k}_i x^i$$
$$(A_P \cdot f_P)(x) = x^r + \frac{k_1 s}{k_0}x^{r-1} + \ldots + \frac{k_{r-1}s^{r-1}}{k_0}x + \frac{s^r}{k_0} = x^r + \sum_{i=0}^{r-1} \frac{k_{r-i}s^{r-i}}{k_0}x^i.$$

This translates to

$$\begin{aligned} k_0^{p+1} &= s^r \\ k_{r-i} &= \frac{k_0 \overline{k}_i}{s^{r-i}} \\ k_{r/2}^{p-1} &= \frac{s^{r/2}}{k_0} \text{ if } r \text{ is even.} \end{aligned} \tag{3.7}$$

The polynomial $f_P$ is not the only polynomial satisfying (3.5) and (3.7). Certainly (3.5) is satisfied for every $C \cdot f_P$ where $C$ is a diagonal matrix. In order for $C \cdot f_P$ to satisfy (3.7) we need $A_P \cdot f_{PC} = \overline{f_{PC}}$, which implies

$$(C A_P \overline{C}^{-1}) \cdot f_P(x) = \overline{f_P(x)}.$$

This condition, together with $C$ being diagonal implies that $C$ is contained in

$$\left\{ \begin{pmatrix} c & 0 \\ 0 & \overline{c} \end{pmatrix} \middle| c \in \mathbb{F}_{p^2}^* \right\}.$$

Multiplying $C$ on the right by a rational scalar matrix or by an element of $\mathrm{Stab}(f_P) = \langle D \rangle$ does not change the polynomial $C \cdot f_P$. Therefore $C$ can be chosen from a reduced set of representatives, for example the following:

$$\mathscr{C}_2 = \left\{ \begin{pmatrix} g^i & 0 \\ 0 & \overline{g}^i \end{pmatrix} \middle| g \text{ generator of } \mathbb{F}_{p^2}^*, 0 \leq i < \tfrac{p+1}{r''} \right\},$$

where $\frac{p+1}{r''} = \gcd(p+1, \frac{p^2-1}{r'})$, in other words $r'' = \frac{r'}{\gcd(r', p-1)}$. The choice of $r''$ follows from the exponents of $g$ being chosen modulo $p+1$ (action of $\mathbb{F}_p^*$) and modulo $\frac{p^2-1}{r'}$ (action of $r'$'th roots of unity).

The table $T$ contains $\{PCP^{-1} \cdot f\}_L$ together with a description of $C$ for all $C$ in $\mathscr{C}_2$ (note that $PCP^{-1}$ is rational). It has $\frac{p+1}{r''}$ elements, and for all polynomials $t$ in the table, $t_P$ satisfies (3.5) and (3.7).

### Search

In the search phase we consider $g(x) = x^r + \sum_{i=0}^{r/r'-1} g_i x^i$ that satisfy (3.5) and compute $\{g_{P^{-1}}\}_L$ until we find a hit in $T$. In that case $f = g_{(PC)^{-1}}$. We analyse the run-time separately for the two types of polynomials.

**Rational.** For rational polynomials, the coefficients $g_i$ are in $\mathbb{F}_p$. The total number of polynomials $g$ is $p^{r/r'}$ and we expect to find a hit after $O(p^{r/r'-1} r')$ trials.

**Irrational.** For irrational polynomials, the coefficients $g_i$ are in $\mathbb{F}_{p^2}$ and they satisfy (3.7). Therefore there are $p+1$ choices for $g_0$, the $g_i$ with $1 \le i < r/2$ can be chosen freely, giving $p^2$ choices each, and the $g_j$ for $r/2 < j$ are constrained to one value for each choice of the previous coefficients. If $r$ is even, $g_{r/2}$ has $p-1$ choices. The total number of polynomials $g$ is $O(p^{r/r'})$ and we expect to find a hit after $O(p^{r/r'-1}r'')$ trials.

### 3.2.3 Ugly polynomials algorithm

We recall that $f$ is *ugly* if it is a reducible polynomial of degree $r \ge 3$. Write $f(x) = l(x)h(x)$ where $r_h = \deg(h(x)) \ge r/2$.

The Legendre symbol is multiplicative, and Möbius transformations are homomorphic with respect to polynomial multiplication, so we have $\{f_m\}_L = \{l_m\}_L\{h_m\}_L$, where the multiplication is element-wise. It follows that $\{f_m\}_L\{l_m\}_L = \{h_m\}_L$.

#### Precomputation

We create two tables, $T_1$ containing $\{f_m\}_L$ for all $m \in \mathscr{M}$, and $T_2$ containing sequences of all polynomials $g(x)$ of degree $r - r_h$ (the candidates for $l_m(x)$). The main table $T$ is a product of $T_1$ and $T_2$, i.e., a table of size $O(p^{r-r_h+3})$ containing $\{f_m\}_L\{g\}_L$ for all $m \in \mathscr{M}$ and all $g$.

#### Search

The search phase consists of trying random polynomials $t(x)$ of degree $r_h$ until we find a hit in $T$. This gives $\{t\}_L = \{f_m\}_L\{g\}_L$, and implies that $t(x) = h_m(x)$, $g(x) = l_m(x)$, and finally $f(x) = g_{m^{-1}}(x)\,t_{m^{-1}}(x)$. We expect to find a solution in $O(p^{r_h-3})$ trials.

The above description glosses over a number of minor details that one needs to be careful about. The run-time is actually $p^{r_h}$ divided by the size of the orbit of $h(x)$.

If $h$ is *good*, then its orbit is maximal and we are good.

If $h$ is *bad*, we can test all *bad* $h$ in time $O(p^{r_h/r'_h}L)$ for each $r'_h \mid r_h$, so in total $O(p^{r_h/2}L)$. For both Type 1 and Type 2 we can enumerate all polynomials $h$ in time $O(p^{r_h/r'_h-1}r''_hL)$ with $r''_h$ defined as in (3.2.2).

If $h$ is *ugly*, we analyse two cases:

1.) $h$ has an irreducible factor of degree at least 3.

Suppose $h = h_1h_2$ of degrees $r_1$ and $r_2$. We select a set of $O(p^{r_1-3})$ representatives for

$h_1$, multiply them with polynomials of degree $r_2$ and search for $\{h\}_L = \{h_1\}_L\{h_2\}_L$ in $T$, achieving an $O(p^{r_h-3})$ run time.

2.) $h$ has all factors of degree $\leq 2$.

There are three subcases to consider:

- $h$ is divisible by a product of three linear polynomials. Then at least one $h_m$ is divisible by $x(x-1)(x-2)$, so we test for $h = x(x-1)(x-2)h_2$ where $h_2$ are of degree $r_h - 3$.
- $h$ is divisible by a linear and quadratic polynomial. Then one of $h_m$ is divisible by $x(x^2-u)$ where $u$ is a chosen non-square, so we test for $h = x(x^2-u)h_2$ where $h_2$ are of degree $r_h - 3$.
- $h$ is divisible by two quadratic polynomials. Then one of them can be considered to be $x^2-u$ where $u$ is a non-square, and the other one has only one degree of freedom. We test for $h = (x^2-u)h_1h_2$ where $h_1$ is selected from $O(p)$ quadratic polynomials and $h_2$ is of degree $r_h - 4$.

Therefore if $f$ is *ugly* we can find it in $O(p^{r_h-3})$ trials irrespective of the type of $h$.

### 3.2.4 Time-memory tradeoff for low degrees

The run-time of the algorithm depends mainly on the search stage. However for some low degree polynomials, the precomputation may take longer than the search stage. In some cases a time-memory tradeoff allows to reduce the complexity further.

#### Good polynomials

For $r \geq 6$, the table-based collision search with an $O(p^3)$ table and $O(p^{r-3})$ trials is optimal, given the approach as presented. For $3 \leq r \leq 5$, a tradeoff with an $O(p^{r/2})$ table and $O(p^{r/2})$ trials is better.

#### Bad polynomials

If $r/r' - 1 < 2$ then the bottleneck is the precomputation phase that takes $O(p^2 \log r)$ steps. This can happen when $r' = r/c$ for $c = 1, 2$. Not much can be done to reduce the precomputation cost since testing *badness* costs $O(p^2 \log r)$. For $r = 3$ we can lower the attack complexity to $O(p^{1.5})$ with table-based collision search for *good* polynomials.

#### Ugly polynomials

We test if $f$ is ugly by trying to find it using the *ugly* polynomials algorithm for each $r_h = \lceil r/2 \rceil, \ldots, r-1$. The precomputation cost is $O(p^{r-r_h+3})$ and the search cost is $O(p^{r_h-3})$.

If $r - r_h + 3 > r_h - 3$, i.e., $r_h < r/2 + 3$, then we can do a tradeoff. Call $\varepsilon := r_h - r/2 < 3$. We compute only the action of $p^\varepsilon$ many matrices on $f$, and after multiplying with the table $T_2$ of $p^{r-r_h}$ sequences, obtain a table of size $p^{r/2}$. We expect to finish the search phase in $O(p^{r_h-\varepsilon}) = O(p^{r/2})$ if a collision exists. Otherwise we assume that $f$ does not have a factor of degree $r_h$ and move to $r_h + 1$.

### 3.2.5 Algorithm comparison

In Table 3.1 we provide a comparison of the best known algorithms for solving the degree $r \geq 4$ generalised Legendre symbol problem. The run-times are given in big-$O$'s. Size of the stabiliser of $f$ is denoted with $r'$, and $r'' = r'$ if $r' \mid p-1$ and $r'' = r'/\gcd(r', p-1)$ otherwise. We denote with $r_h$ the degree of a factor of $f$ which is at least $r/2$. Complexity is given in the number of Legendre symbols computed/extracted. In all cases we need $p$ queries.

Table 3.1: Comparisons of GLSP solving algorithms.

| *good* polynomials | search | precomputation | memory |
|---|---|---|---|
| Khovratovich [Kho19] | $p^{r-1} r \log p$ | $r \log p$ | $r \log p$ |
| Beullens et al. [BBUV20] | $p^{r-2} r^2 \log^2 p$ | $p^2$ | $p^2$ |
| Our algorithm | $p^{r-3} r \log p$ | $p^3 r \log p$ | $p^3 r \log p$ |
| *bad* polynomials | search | precomputation | memory |
| Khovratovich [Kho19] | $p^{r-1} r \log p$ | $r \log p$ | $r \log p$ |
| Beullens et al. [BBUV20] | $p^{r-2} r^2 \log^2 p$ | $p^2$ | $p^{r-r_h} r \log p$ |
| Our algorithm | $p^{r/r'-1} r'' r \log p$ | $p^2 r \log p$ | $(p/r'') r \log p$ |
| *ugly* polynomials | search | precomputation | memory |
| Khovratovich [Kho19] | $p^{r-1} r \log p$ | $r \log p$ | $r \log p$ |
| Beullens et al. [BBUV20] | $p^{r_h} r \log p$ | $p^{r-r_h} r \log p$ | $p^{r-r_h} r \log p$ |
| Our algorithm | $p^{r_h-3} r \log p$ | $p^{r-r_h+3} r \log p$ | $p^{r-r_h+3} r \log p$ |

## 3.3 Algorithm for $r = 2$

If $r = 2$ all polynomials are *bad* or *ugly*. There is a deterministic $O(p)$ algorithm for finding $f$ in this case – we first precompute the action of $\{\left(\begin{smallmatrix} 1 & a \\ 0 & 1 \end{smallmatrix}\right) \mid a \in \mathbb{F}_p\}$ on the polynomial $f$, which assures that the precomputed table contains the Legendre sequence of a polynomial of the form $x^2 - c$:

$$\left(\begin{smallmatrix} 1 & a \\ 0 & 1 \end{smallmatrix}\right) \cdot (x^2 - tx + n) = x^2 - (t - 2a)x + (n + a^2 - ta).$$

Then we test all $p$ such polynomials until we find $f$.

## 3.4 Algorithm for $r = 1$ and the limited query case

In the Section 3.2 we query the oracle at all elements of $\mathbb{F}_p$ and then extract up to $p^3 - p$ sequences. With $p - o(p/L)$ queries we still have access to $\Omega(p^3)$ sequences, so the same algorithms work. In this chapter we show how to solve the problem when the number of queries is limited.

When the secret polynomial is linear doing more than $O(p^{1/2}L)$ queries is wasteful. In fact, there is a straightforward algorithm - we create a table with $O(p^{1/2})$ sequences by doing $L+1$ queries per sequence. This allows us to find the secret polynomial after $O(p^{1/2})$ trials. This is essentially the algorithm in [Kho19], where the author further provides a memory-less approach.

The main difference in the linear case with respect to the higher degree case is that we are allowed $M \leq \sqrt{p}L$ queries to the oracle. A natural question arises – how many different group actions can we obtain from only $M$ queries? The same question can be asked in the higher degree case, and the algorithm we provide can be directly applied in that scenario. One would expect a cubic increase, as with full access to the oracle, but this seems to be out of reach.

### 3.4.1 Linear shifts subgroup

Let $G$ be the subgroup of $\mathscr{M}$ consisting only of linear Möbius transformations,

$$G = \left\{ \begin{pmatrix} d & i \\ 0 & 1 \end{pmatrix} \middle| d \in \mathbb{F}_p^*, i \in \mathbb{F}_p \right\} \leqslant \mathrm{PGL}_2(\mathbb{F}_p).$$

An element $(i,d) := \left(\begin{smallmatrix} d & i \\ 0 & 1 \end{smallmatrix}\right)$ sends $f(x)$ to $f_{i,d}(x)$. In order to extract $\{f_{i,d}(x)\}_L$ from the oracle $\mathscr{O}$ of $f$, we compute

$$\left(\frac{f_{i,d}(x)}{p}\right) = \mathscr{O}\left(\frac{dx+i}{0x+1}\right)\left(\frac{0x+1}{p}\right)^r\left(\frac{d}{p}\right)^r = \mathscr{O}\,(dx+i)\left(\frac{d}{p}\right)^r$$

for all $x \in [0, L)$. If $\mathscr{O}$ is queried in $[0, M)$, then we can extract all $f_{i,d}$ such that $dx + i \in [0, M)$ for all $x \in [0, L)$. This creates the following constraints on $i, d$:

$$\begin{cases} d = 1, 2, \ldots, \lfloor \frac{M-1}{L-1} \rfloor \\ i = 0, 1, \ldots, M - 1 - (L-1)d \end{cases} \quad \text{or} \quad \begin{cases} d = -1, -2, \ldots, -\lfloor \frac{M-1}{L-1} \rfloor \\ i = (L-1)(-d), \ldots, M-1. \end{cases}$$

The total number of eligible $(i,d) \in G$ is

$$\sum_{d=1}^{\lfloor \frac{M-1}{L-1} \rfloor} 2(M - (L-1)d) = \frac{M^2}{L-1} - M + O(L)$$

with the constant in $O(L)$ being at most 2.

The limited query algorithm works as follows:

#### Precomputation

Query $\mathcal{O}$ at $[0, M)$. Extract $O(\frac{M^2}{L})$ Legendre sequences $\{f_{i,d}\}_L$ and save them in a table $T$ together with descriptions of $(i, d)$.

#### Search

Search is done by trying random polynomials until we find a hit in the table, which is expected after $O(\frac{p^r L}{M^2})$ trials, in particular $O(\frac{pL}{M^2})$ for the linear PRF.

#### Further improvements

The cost of the precomputation is $M$ queries and $O(\frac{M^2}{L})$ *sequence extractions*. The cost of the search is $O(\frac{pL}{M^2})$ *trials*. A straightforward way to do a *sequence extraction* is to read the pre-saved queries $L$ times. Due to the nature of the sequences, this cost can be amortised to $O(1)$ per sequence. Doing a *trial* constitutes of evaluating the polynomial in $L$ places and computing $L$ Legendre symbols. Again, this cost can be amortised to $O(\log L)$ per trial. These implementational improvements are not within the scope of this thesis, and they are explained in detail in [KKK20b].

### 3.4.2 Algorithm comparison

The first algorithm by Khovratovich [Kho19] computes sequences with on-the-go queries, and directly computes Legendre symbols. The main benefit of this approach is that it is memory-less, but it requires $O(\sqrt{p})$ oracle queries. This was improved on in [BBUV20] by extracting sequences rather than querying/computing symbols, and increasing the sequence yield to $M^2/L^2$. In our terminology, the authors of [BBUV20] use the same group $G$ but only elements $(i, d)$ such that $i < d$ , leading them to a table which is a factor of $L$ smaller with respect to ours. Using the full group $G$ as in 3.4.1 comes with cheaper sequence extraction in the precomputation stage, but more expensive sequence extraction in the search stage thus the $\log\log p$ factor in Table 3.2. A more detailed analysis is given in [KKK20b].

Table 3.2: Comparisons of best known algorithms for the linear Legendre PRF challenge, in big-$O$'s and $\Theta(\log p)$-bit word operations. We denote with $t$ the time to compute a Legendre symbol

| Algorithm | search | precomputation | memory | best obtainable run-time |
|---|---|---|---|---|
| Khovratovich [Kho19] | $\frac{p\,t\log^2 p}{M}$ | $M$ | $\log p$ | $\sqrt{p}\,t\log p$ |
| Beullens et al. [BBUV20] | $\frac{p\log^2 p}{M^2}$ | $M^2$ | $\frac{M^2}{\log p}$ | $\sqrt{p}\log p$ |
| Our algorithm | $\frac{p\log p\log\log p}{M^2}$ | $\frac{M^2}{\log p}$ | $M^2$ | $\sqrt{p\log\log p}$ |

### 3.4.3 Experiments

Ethereum research posted a number of challenges [Fei19] for breaking the linear Legendre PRF. In each challenge we are given a prime $p$ of size varying from 64 to 148 bits, and $M = 2^{20}$ bits of the sequence $\{k\}_M$ as defined in (1.3). The challenge is to recover the key $k$. In each case we were able to precompute a table with $\sim 2^{34}$ sequences. The most interesting is of course challenge #2 since it had not been solved before. The actual number of trials performed in challenge #2 is $2^{46.97} = 1.38e14$ which is far less than expected. This can be explained by large variance and by sheer luck. The two most difficult challenges (#3 and #4) are out of reach with the proposed attack and its implementation. An in-depth explanation of the experiments is given in [KKK20b]. The code and the keys of the first three challenges can be found at

https://github.com/nKolja/LegendrePRF.

Table 3.3: Results and estimates for solving the Legendre PRF challenges [Fei19].

| Challenge | Prime bit size | Expected # trials | Observed # trials | Expected core-hours | Observed core-hours |
|---|---|---|---|---|---|
| 0 | 64 | $2^{30}$ | $2^{30.78}$ | 290 sec | 490 sec |
| 1 | 74 | $2^{40}$ | $2^{39.53}$ | 82 | 59 |
| 2 | 84 | $2^{50}$ | $2^{46.97}$ | 1.4e5 | 1.72e4 |
| 3 | 100 | $2^{66}$ | - | 9.1e9 | - |
| 4 | 148 | $2^{114}$ | - | 2.5e24 | - |

## 3.5 Conclusion

### 3.5.1 Security recommendations

Following our argumentation, the most secure PRFs are the ones coming from *good* polynomials. The best way to start is to make sure that the secret key is an irreducible polynomial.

The number of *bad* polynomials can be shown to be small. There are at most $p^{r/r'+2}r'$ polynomials with a stabiliser of size $r'$, since each of them is conjugate to a polynomial satisfying (3.5). By using the inclusion-exclusion principle we see that the number of *bad* polynomials is bounded by

$$\sum_{\substack{r'|\gcd(r,p^2-1)\\ r'>1}} -\mu(r')p^{r/r'+2}r' = O(p^{r/3+2}r). \tag{3.8}$$

Therefore a random polynomial is unlikely to be *bad*. The easiest way to assure that our secret polynomial is not *bad* is to choose $p$ and $r$ such that $\gcd(r, p^2-1) = 1$.

We also recommend using higher degree polynomials in order to increase security as opposed to

increasing the size of the field and using the linear Legendre PRF. The main reason behind this choice is efficiency. Let $M(n)$ be the cost of multiplying two $n$-bit numbers. On a field of size $n$ bits, the Legendre symbol can be computed in $O(M(n)\log n)$ operations [BZ10], and degree-$r$ polynomial evaluation can be computed in $O(rM(n))$ with Horner's method.

Generalised Legendre PRF with a polynomial of degree $r$ and a field of size $n$ provides at least as much security as the linear Legendre PRF on a field of size $rn$. The cost comparison is given in Table 3.4.

Table 3.4: Cost comparison of linear and high degree Legendre PRF

| Protocol | Multiplication | Legendre symbol | Oracle evaluation |
|---|---|---|---|
| degree 1 | $O(n)$ | $O(M(rn)\log(rn))$ | $O(M(rn)\log(rn))$ |
| degree $r$ | $O(rM(n))$ | $O(M(n)\log(n))$ | $O(M(n)(r+\log n))$ |

### 3.5.2 Future directions

As it currently stands, all classical attacks on the Legendre PRF are based on collision-searching methods. This bounds the attack from below by the square root of the key space, and no better attacks for finding the secret key are known.

There are quantum algorithms which solve the problem in polynomial time, however they rely on a quantum oracle. So far there are no quantum algorithms that solve the classical oracle problem in subexponential time, even though some improvements have been done in that direction [FS21]. Given that Legendre symbol based cryptographic schemes are becoming increasingly popular, this subject would certainly be of interest.

Another interesting research direction would be to give an algorithm which enumerates all Möbius transforms which can be used with a predetermined set of queries.

When $M = p - o(p/L)$, then we can extract $\Omega(p^3)$ sequences, and when $M$ is "small" we showed how to extract $M^2/L$ sequences. Can one obtain more than a quadratic yield for small $M$? At what point does the yield increase from quadratic to cubic? Can the yield increase by changing the definition of sequences? All of these are interesting questions which can be studied, but so far seemed out of reach.

# Part III Supersingular isogeny key encapsulation

# 4 Full key recovery side-channel attack against ephemeral SIKE on the Cortex-M4

In this chapter we present a full secret key extraction attack using a side-channel power analysis of recommended implementation of SIKE on the ARM Cortex-M4 microcontroller. The target of our attack is the three-point ladder given in Algorithm 5, page 29. We firstly target the three-point ladder with an attack which processes multiple traces obtained from computations done with the same secret key. Then, we show how to extend the attack to a fully ephemeral setting where a single trace is enough to fully extract the secret key.

Because the attack requires just a single trace it can be applied at any stage of the protocol: key generation, key encapsulation and key decapsulation.

In addition to the attack, we argue that many countermeasures that were mentioned in the literature [ZYD+20] are defeated by the attack. Finally, we recommend the well-known projective point coordinate randomisation, which stops our attack with a negligible performance overhead.

Side-channel analysis of supersingular isogeny protocols was initially conducted in [KAJ17] in which the authors address concerns about power analysis without carrying out a practical experiment. The first paper to practically evaluate the side-channel vulnerabilities of SIKE is due to Zhang et al. [ZYD+20]. In their study, the authors fully describe a practical differential power analysis on the three point ladder of the key decapsulation procedure which requires thousands of traces, and discuss potential countermeasures. However, since the authors rely on the fact that the private key is fixed across the measurements, the attack is applicable only to the semi-static settings of the SIKE protocol. We extend these results and target SIKE in ephemeral settings.

In the past, many papers have already mounted horizontal attacks against the classical Montgomery ladder in the case of elliptic curve cryptography, such as [CFG+10], [PZS17], and [APS19]. We apply similar techniques, but on the variant of the ladder with three points used in SIKE.

Template attacks have also been explored against the elliptic curve scalar multiplication, for instance in [MO09], [ZWMZ14], and [DPN+16]. As opposed to horizontal correlation power analyses, template attacks require control over the input of the targeted procedure and sometimes

even further interactions with the targeted device. Online template attacks [BCP+14] against classical elliptic curve cryptography require only one power trace of the target device, but additional power measurements on the same or a similar template device are needed. Our horizontal correlation power analysis does not rely on such a requirement and is executed purely offline. In particular, our attack is based on an entirely different setup where, instead of correlating power traces with each others, we correlate Hamming weights of processed values. Our results show much stronger correlations due to the reliance on a specific leakage model, unlike the online template attack which is leakage-agnostic.

Other post-quantum algorithms have been targeted by power analyses. In a similar fashion, Aysu et al. [ATT+18] have attacked the lattice-based key exchanges of Frodo and NewHope with a horizontal correlation power analysis. Bos et al. have addressed this attack in [BFM+19] and proposed a profiled extend-and-prune approach. Recently, Sim et al. [SKL+20] have shown a single-trace ephemeral-key recovery against various lattice-based key exchanges.

Finally, let us mention the work of Primas et al. [PPM17] in which the first single-trace attack on lattice-based encryption was described using belief propagation. This work was recently extended by Kannwischer et al. [KPP20] to a single-trace power analysis of the Keccak hash function, used in various applications, including the hash-based signature scheme SPHINCS+.

## 4.1 Correlation power analysis

A Correlation power analysis (CPA) [BCO04] is a statistical known-text side-channel power analysis that aims to deduce a portion of a secret value across multiple power measurements. A CPA aims to use a correlation coefficient to quantify the link between power consumption and the values processed by a processing unit. In the scope of this paper, we consider two types of CPA:

- *Vertical* CPA, which targets a *fixed* secret value across different executions of the attacked algorithm by collecting *multiple* power traces that correspond to multiple executions of the *same* operation.
- *Horizontal* CPA, which targets an *ephemeral* secret value using a *single* power trace that correspond to *multiple* operations. These operations must be similar to allow the segmentation of the power trace into multiple ones thereby simulating a vertical CPA.

In a typical threat model for CPA, the adversary has the capability of measuring the power consumption of a target device which acts as a black-box key decapsulating device. The algorithm inputs are not required to be manipulated but are supposed to be accessible by the target device. As a result, a CPA attack is completely passive (i.e., non-intrusive) and can be mounted even during a trusted communication between two honest parties.

**Definition 4.1.1** (Pearson's correlation coefficient)**.** To assess correlation between the processed values and the power samples, the Pearson's correlation coefficient (PCC) is computed. Let $n > 0$ be the number of measurements, each of which consists of $S > 0$ power samples. Then, let

$T(s) \in \mathbb{R}^n$ be a vector of power samples synchronised at a same instant $0 \leq s < S$, and $M \in \mathbb{N}^n$ a vector of the Hamming weights of the processed values.

$$\mathrm{PCC}(M, T(s)) = \frac{\mathrm{Cov}(M, T(s))}{\sqrt{\mathrm{Var}(M)\,\mathrm{Var}(T(s))}}. \tag{4.1}$$

The overall attack consists of the following steps:

1. Find an operation in the attacked procedure which involves:
   (a) A (small) portion of a secret value which is the same across all measurements.
   (b) A known input (resp. output).

   In the following, we refer to the result of this operation as the *intermediate value*.
2. Collect $n > 0$ power traces consisting of $S > 0$ power samples each, i.e., $T(s)$ for $0 \leq s < S$, that correspond to the computation of the intermediate value with different inputs (resp. outputs).
3. Take a guess for the portion of the secret value involved in the intermediate value computation.
4. Compute the vector of intermediate values from the known inputs (resp. outputs) and the secret value guess, and derive its corresponding vector of Hamming weight $M$.
5. For each vector of power samples at a same time, i.e., $T(s)$ for each $0 \leq s < S$, compute $\mathrm{PCC}(M, T(s))$.
   This results in a vector of PCC at each moment in time.

Using a large enough $n > 0$ given the signal-to-noise ratio of the power consumption, a strong PCC at any point in time indicates a valid guess, while a weak PCC at every point in time can rule out said guess.

Figure 4.1 gives a visual example of a CPA. In this example, the portion of the secret value is only one bit, resulting thus in two possible intermediate values. The PCC computation takes one of the two Hamming weight vectors $M$ sketched on the left of the figure, and each vector of power samples at a same timing instant shown on the right, to produce each point in the corresponding PCC plot below. Since the PCC plot for the bit guess of one shows a spike, the corresponding bit for the secret value is successfully recovered.

## 4.2 Side-channel analysis

In this chapter, we explain how to exploit the link between power consumption and processed data in order to recover private key bits.

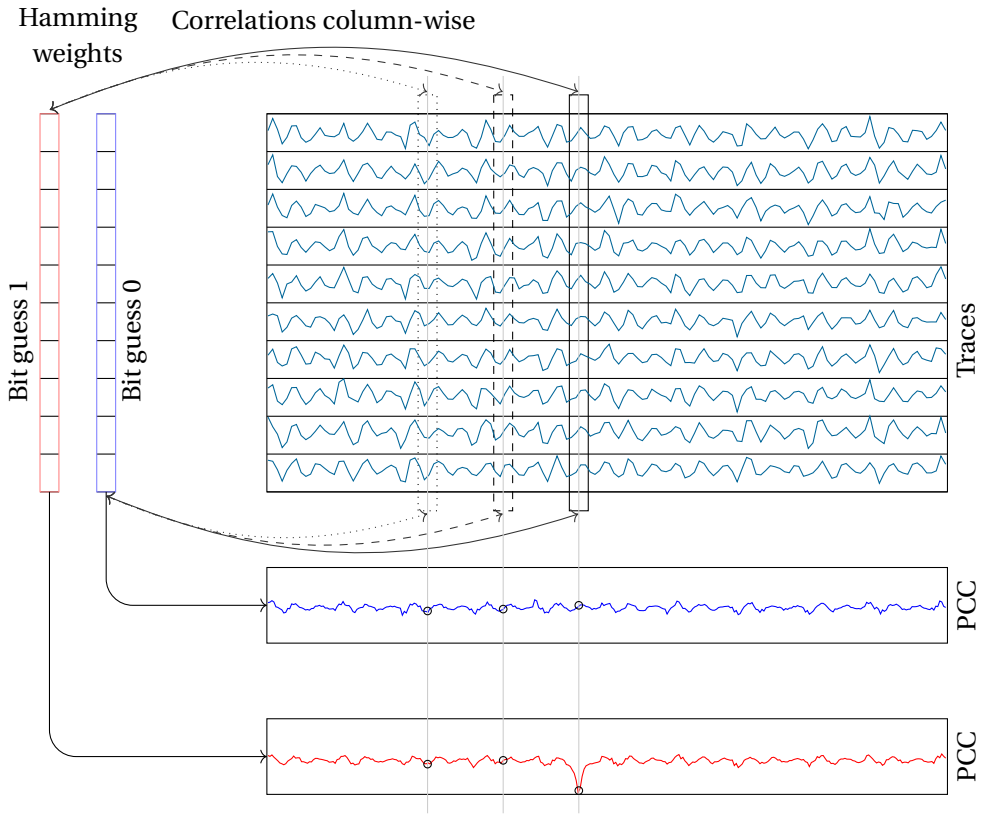



Figure 4.1: Visual example of a CPA. Correlations between two arrays of Hamming weights and the power traces are plotted in the bottom. A strong correlation indicates that the bit value associated to these power traces is 1.

### 4.2.1 Point of attack

The main point of attack is the three-point ladder (see Algorithm 5). This is a function which takes as input an elliptic curve $E$, two points $P$ and $Q$ on that curve, and a secret key $sk$. The "curve and points" triple may be thought of as a part of the public parameters $(E_0, P_I, Q_I)$, or a public key $pk_I = (E_I, \phi_I(P_J), \phi_I(Q_J))$ with $\{I, J\} = \{A, B\}$. The three-point ladder computes the point $R = P + [sk]Q$ where $sk$ is the private key of the computing party.

Recalling Chapter 2.5.2, the triple $(E, P, Q)$ containing a curve and two points is represented as three field elements $(x_Q, x_P, x_{Q-P})$, where $Q = [x_Q : 1]$, $P = [x_P : 1]$, $Q - P = [x_{Q-P} : 1]$. The coefficient $a$ defining the curve $E$ can be obtained from these values with a small number of modular multiplications, squarings and a single inversion, as shown in formula 2.35.

The main ingredient of the three-point ladder is a double-and-add function `xDBLADD` given in Algorithm 4 on page 29. This function takes as input a triple of points $R_0, R_1, R_2 \in E$ in Montgomery coordinates such that $R_2 = R_0 - R_1$, the coefficient $a$ of the curve, and outputs $([2]R_0, R_0 + R_1, R_2)$.

We show how the computation of $R = P + [sk]Q$ is done in practice in Algorithm 8.

**Algorithm 8:** SECRET POINT COMPUTATION

**Input:** A public key or public parameters $x_Q, x_P, x_{Q-P}$,
an integer $sk = \sum_{i=0}^{\ell-1} sk_i 2^i$ with $sk_i \in \{0,1\}$.
**Assumes:** $[x_P : 1], [x_{Q-P} : 1] \notin \{[1:0], [0:1]\}$.
**Output:** $P + [sk]Q$, where $Q = [x_Q : 1]$, $P = [x_P : 1]$.
**Procedure** `sec-point`$(x_Q, x_P, x_{Q-P}, sk)$

1 $a =$ `get_a`$(x_Q, x_P, x_{Q-P})$ `// See formula 2.35, page 26.`
2 $Q, P, Q-P \leftarrow [x_Q : 1], [x_P : 1], [x_{Q-P} : 1]$
3 $R =$ `three-point ladder`$(Q, P, Q-P, sk, a)$ `// See Algorithm 5, page 29.`
**return** $R$

The goal of the attack is to measure the power consumption of the `xDBLADD` operation and to deduce if the function was executed with or without the `swap` at step 5 of Algorithm 5. We may assume that we know the private key up to bit $i-1$, by induction. We also know the starting points $Q, P, Q-P$ since they are public. Therefore, we may obtain the two possible inputs for `xDBLADD`, and we know how they relate to the value of the $i^{\text{th}}$ bit of the private key. The two inputs and their Hamming weights are computed and the power trace of certain instructions within `xDBLADD` is correlated with the Hamming weights. Thanks to CPA, this allows us to distinguish when the $i^{\text{th}}$ bit is zero or one.

### Double-and-add

Despite the involvement of a (random) bit of the private key, `xDBLADD` is a deterministic function. The inputs and outputs of each subprocedure in `xDBLADD` depend only on the original inputs of the function. As a result, an educated guess on the original inputs allow us to infer the results of all the operations involved in `xDBLADD`.

The function consists of 7 multiplications and 4 squarings of $\mathbb{F}_{p^2}$ elements, and multiple field additions, subtractions, and modular reductions. Each $\mathbb{F}_{p^2}$ multiplication and each squaring contain two multi-precision additions of $\mathbb{F}_p$ elements, referred to as "`mp_addfast`". This multi-precision addition is the operation on which our attack is focused. In total, there are $11 \times 2 = 22$ `mp_addfast` functions, out of which only 10 have inputs which differ in case of a `swap` at step 6 of the three-point ladder. The code of `xDBLADD` can be found in Figure C.1, and the code of the $\mathbb{F}_{p^2}$ multiplication and squaring function in Figure C.2, C.3.

### Multi-precision addition

In the Cortex-M4 implementation of SIKE, the `mp_addfast` is written in assembly. The function computes the addition of two $\mathbb{F}_p$ elements. Depending on the size of $p$, each field element is saved in an array of $n \in \{14, 16, 20, 24\}$ words of 32-bit. Each `mp_addfast` executes $2n$ load instructions (`LDMIA`), $n$ store instructions (`STMIA`), and $n$ additions (`ADDS`, `ADCS`). These are executed in batches of four consecutive additions, due to the limited number of available registers

on the Cortex-M4. The code of the `mp_addfast` function can be found in Figure C.4.

### 4.2.2 Vertical attack

In a vertical attack against SIKE, we measure multiple executions of the three-point ladder in which the target's private key is fixed, but the client public key inputs are different. From these traces, we concentrate only on a single `mp_addfast` instruction per `xDBLADD`, i.e., per bit of the private key. Within the `mp_addfast`, we can decide to focus even further on the first addition instruction. We can thus compute the two possible outputs of the first `ADDS` depending on the `cswap`, for each public key, and then correlate the two vectors of Hamming weights of these outputs with the power traces using the CPA procedure from Section 4.1. This process can be repeated for each bit of the target's private key, as the correctness of each guess depends on the correctness of previous ones, resulting thus in an *extend-and-prune* attack.

### 4.2.3 Horizontal attack

In the horizontal attack scenario, we can measure only one power trace for a single execution of the three-point ladder. The same approach as in the vertical attack cannot be used because there would not be enough data to obtain strong correlations. We can work out this issue and re-obtain "*verticality*" by combining the power traces of all 10 `mp_addfast` functions within each `xDBLADD`. This way, we obtain 10 power traces with which we can correlate pairs of inputs – similarly as in a vertical attack with 10 power traces.

We can further improve this attack. A multi-precision addition takes two $\mathbb{F}_p$ elements as input and gives one as output. Each one of the $2n$ input words of 32-bits is loaded once and then used in the addition instruction, and the $n$ output words of 32-bits are stored. In total, there are $3n$ words of 32-bits which pass through the pipeline registers and whose Hamming distance from the previous word in the pipeline are related to the power consumption.

For each of the $3n$ words, we compute the PCC between the 10 power traces and the 10 pairs of hamming weights of 32-bit words accounting for the two guesses of the current bit of the private key. For each word, a spike in the correlation is expected at a different position depending on the instruction which uses this particular word. The locations of spikes can be deduced from the shape of the power traces. Once the $3n$ pairs of correlations are computed, we can add them up such that the locations of the expected spikes are aligned. We expect to end up with two correlations for each guess of the private-key bit, with a clear spike in the correlation plot of the correctly guessed value.

In presence of noise in power measurement, the private key guesses may be erroneous. A single wrong guess of a bit of the private key leads to completely inconclusive results, because the following guesses depend on the correctness of the previous bits. Therefore, it is of particular importance that no erroneous guesses are made in the process of key extraction. We propose two

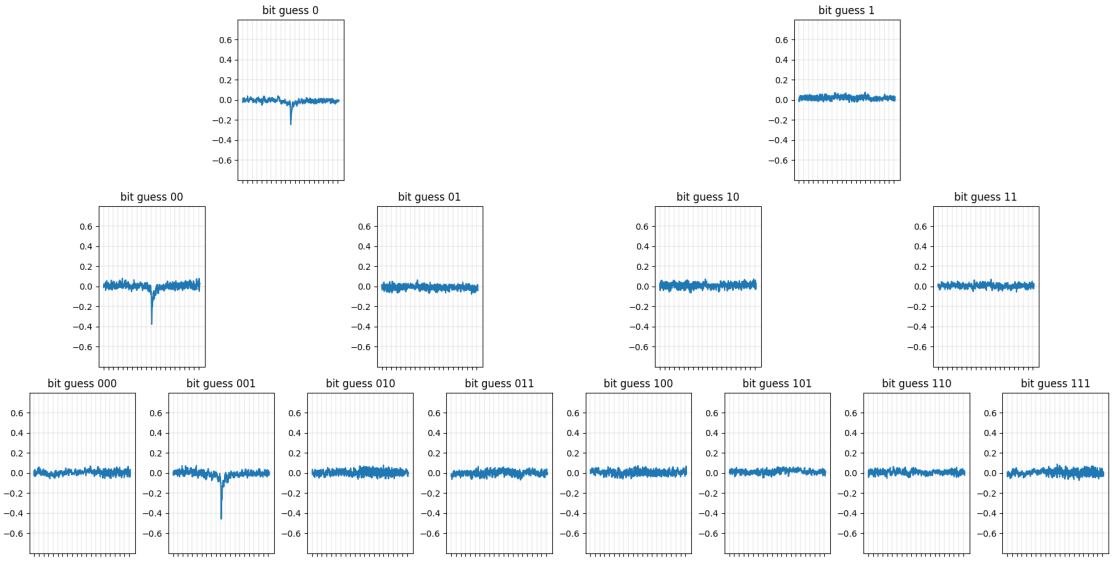



Figure 4.2: Depth search.

measures to approach this problem.

### Depth search

When the guess of a single bit gives inconclusive results, we can proceed by making four guesses for the next two bits in hope of finding a correlation coefficient with a notable spike. In particular we can make a guess for $k$ consecutive bits, obtaining in total $2^k$ different combinations. For each combination we compute a PCC for each of the $k$ bits. In total there are $2(2^k - 1)$ correlation coefficients. We then add up all the PCCs for each $k$-bit combination and we guess the current $sk$ bit to be the trailing bit of the combination with the strongest correlation.

### Increasing verticality

We can increase verticality (i.e., the amount of power traces in the horizontal settings) by computing correlations for bits in windows of size $k$. If, for one bit, 10 `mp_addfast` functions can be measured from a single `xDBLADD`, then, for $k$ bits, there will be $k \times 10$ traces of `mp_addfast` functions from the $k$ consecutive `xDBLADD` functions. In total, $2^k$ hypotheses need to be made (one per bit), and $2^k$ correlation coefficients are computed for $10k$ power traces.

Finally, rather than performing the attack on contiguous windows of $k$ bits, we select only one bit of the target's private key to be the trailing bit of the $k$-bit combination with the strongest correlation. This way, we can re-run the process starting from the bit right afterwards as a way to correct errors due to the potential proximity of strong correlations. This process resembles the error-correction procedure introduced in [DPN+16].

Also, we mention that other operations, such as `fpmul_mont` and `fpsub`, can be measured and combined to increase verticality. While these are dissimilar operations and may leak information differently than `mp_addfast`, they may still add information to the overall selection of target's private bits.

## 4.3 Experimental results

In order to validate the horizontal attack described in Section 4.2, we reproduced the key recovery on a programmable board which includes the ARM Cortex-M4 microcontroller (the target) and runs the reference Cortex-M4 implementation of SIKE [SAJA20]. Details on the hardware setup, the equipment, and the specifications can be found in [GdGK21].

### 4.3.1 Target implementation

The attacked implementation is the official SIKE implementation adapted for (32-bit) ARM Cortex-M4 microcontrollers [SAJA20], which is part of the official submission package. The implementation already has some levels of side-channel resistance in that it is constant-time. We attacked SIKE instantiated with a prime of 434 bits (i.e., `SIKEp434`); a choice that we elaborate on in this section.

We made modifications in the SIKE implementation to ease the collection and the pre-processing of the traces. Note that these modifications were made for efficiency purposes and are by no means necessary for our attack to work. In other words, we emphasise that the attack can be mounted on the original implementation of SIKE presented in [SAJA20] without any difficulty.

The list of modifications are the following:

- A trigger is toggled when the double-and-add operation of the three-point ladder enters into an `mp_addfast` procedure that depends on the swap.
- An idle delay of about 1 millisecond was introduced in between each `mp_addfast` call, and of about 1 second after each loop iteration of the three-point ladder.

**Limitations of the software.** While the introduction of a trigger and multiple delays results in an unrealistic attack scenario, we emphasise on the fact that the attack is still possible on an unmodified SIKE implementation. The process of segmenting the power traces, as well as the correlation and Hamming weights computations can be done *offline*, after the power traces have been sampled. In a plain attack, as opposed to our experiment, the traces acquisition will be synchronised on serial communication. Then, the targeted operations need to be identified within the full resulting power trace (e.g., using cross-correlation techniques, as in [DPN+16]), so the sub-power traces corresponding to the attacked instructions can be manually segmented and carefully aligned to perform the CPA. This cumbersome process is not the main focus of our

study and was therefore duly skipped.

#### Other SIKE instances

To achieve various levels of security, the original SIKE submission [JAC+17] presents four different parameter sets, each of which with a prime of different size ranging from 434 to 751 bits (see Figure 2.4). While instantiating SIKE with a larger prime offers stronger security guarantees against theoretical cryptanalysis, larger instances present a wider attack surface in a single-trace power analysis. This property was also observed by Bos et al. [BFM+19], and is due to the increased number of instructions executed which, therefore, yield more power measurements. As a result, our attacked instance (`SIKEp434`) is expected to be the hardest to attack with a single trace.

Also, the compressed instances of SIKE are prone to the same horizontal attack, because the starting points of the three-point ladder are deterministically obtained from the compressed public key.

### 4.3.2 Collection of traces

Our experiment simulated a portion of the SIKE key exchange between Alice (ephemeral party represented by the computer) and Bob (static party, the target, represented by the Cortex-M4); namely, the key decapsulation procedure. Our attack scenario can be summarised with the following steps:

1. On the computer, generate Bob's key pair at random, and send Bob's private key to the Cortex-M4.
2. Given Bob's public key, generate Alice's key pair at random.
3. Send a public key to the Cortex-M4 and measure the power consumption of:
    - only the *second* `mp_addfast` call involved in steps 6 and 8,
    - and both `mp_addfast` call involved in steps 16, 17, 18, and 19,

This attack scenario was repeated a total of 460 times. Each of these experiments includes the power traces of the 10 `mp_addfast` calls from the loop iterations for all the 217 bits of Bob's private key. Hence, $460 \times 10 \times 217 =$ 998,200 different power traces were acquired during that experiment.

**Power traces.** Power traces are measured with an oscilloscope at a rate of 2.5e8 samples/second during a period of 20$\mu$s. As a result, a power trace for a single execution of `mp_addfast` includes 5,000 power samples. A filter was applied to the power traces in order to down-sample them. In addition to down-sampling, the high frequencies were removed because they were deemed unnecessary. More information on the filtering of the traces in this attack can be found in [GdGK21]. The figure 4.3 shows the average power consumption of an `mp_addfast` execution before and after filtering.

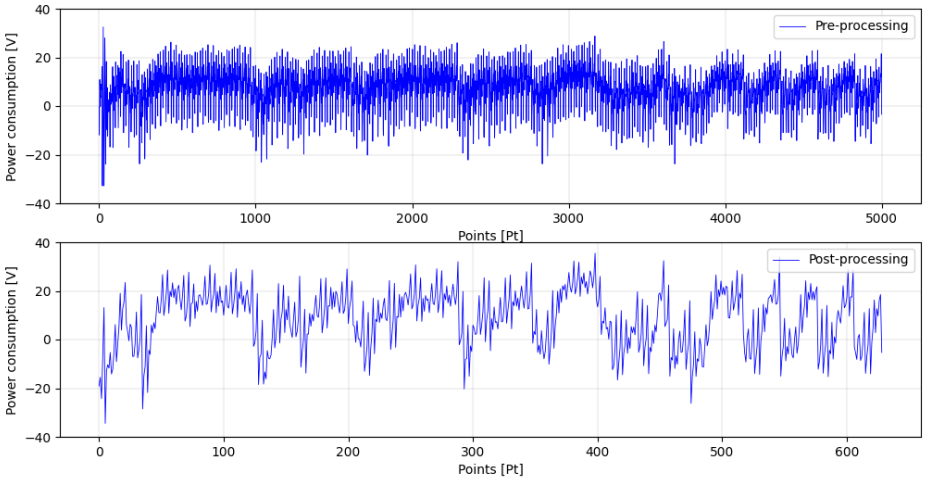



Figure 4.3: An `mp_addfast` power trace before and after filtering.

The traces and the attack are made accessible at https://github.com/nKolja/SIKE-HPA-2021.

### 4.3.3 Horizontal CPA procedure

Using the power traces, we performed a horizontal CPA on each iteration of the loop in the three-point ladder (Algorithm 5, page 5). Each time, a single bit of the target's private key is attacked, with the assumption that the previous bits are known. This process can then be repeated across all the bits of the key. Since a single bit is hypothesised at each step of the horizontal attack, there are only two hypotheses to consider:

- The points $R$ and $R_2$ were swapped (the bit is different from the previous bit).
- The points $R$ and $R_2$ were left un-swapped (the bit is the same as the previous bit).

A strong correlation between the power traces for one loop iteration and the values corresponding to one of the two hypotheses indicates the correctness of the hypothesised bit. As the attack moves forward, a successful recovery of the first bits allows the recovery of the next ones. Therefore, a full-key recovery can be incrementally mounted in an extend-and-prune manner.

#### Power traces segmentation

Due to the ephemeral settings of the protocol, we have access to only a single trace per loop iteration involving a single bit of the target's private key. Therefore, in order to apply a classical CPA, we need to obtain verticality, i.e., find a way to obtain a certain amount of multiple different power samples which are linked to a same portion of the private key. In our case study, we segmented the power trace that corresponds to an iteration of the three-point ladder into 10 different power traces,

each of which corresponding to an `mp_addfast` execution, for which, given either hypothesis, the full input and output (and thus, relevant Hamming distances information) are known. As a result, our horizontal CPA will amount to a vertical CPA with 10 power traces and 2 hypotheses.

#### CPA enhancements

To further improve the success of our attack, we have inspected the targeted function for which the power traces were collected. Because, in our experiment, $p$ is 434-bit long, each element is saved as an array of $\lceil 434/32 \rceil = 14$ words of 32 bits. This results in exactly 14 addition instructions, hence 14 leakage points, in a single `mp_addfast` power trace.

Moreover, we considered the leakage model from a Cortex-M4 microcontroller as explained in [CGD18]. Because the power consumption leaks in the Hamming distance between the pipeline registers, we actually obtain *three* leakage points on a power trace per instruction, coming from the Hamming distance between:

(1) the first inputs of the current and the previous instruction,
(2) the second inputs of the current and the previous instruction, and
(3) the output of the current and the previous instruction.

This results in an additional segmentation of $3 \times 14 = 42$ points of leakage. For each point of leakage, a PCC is computed with the 10 `mp_addfast` power traces and the 10 Hamming distances.

We expect each of these PCCs to produce a spike at a different point in time in the correlation plot which we try to recover. The location of the spike corresponds to the position at which the associated 32-bit word is processed by a pipeline register. Each of these leakage points is constant throughout the `mp_addfast` executions and the three-point ladder loop (assuming the power traces are properly aligned, which can be automated using basic peak alignment methods). These positions can even be identified by analysing the spike structure of the power trace (using, e.g., cross-correlation techniques).

Finally, the 42 PCCs at each point of leakage are added together to produce a larger spike. This consists of aligning all correlation plots on their leakage points and adding them together. We expect the difference of added correlation coefficients to be large enough to correctly validate the private bit.

### 4.3.4 Results

Among the 460 trials, our experimental results returned a resounding success rate of 100% in recovering the full key. None of the improvements described in Section 4.2.3 were required. An example of the corresponding CPA is shown in Figure 4.4 where six bits are shown to be successfully recovered.

This proof of concept shows that, even in ephemeral settings, the official ARM implementation of SIKE is vulnerable to classical power analysis techniques.

All the code used to derive our results is shared on https://github.com/nKolja/SIKE-HPA-2021.

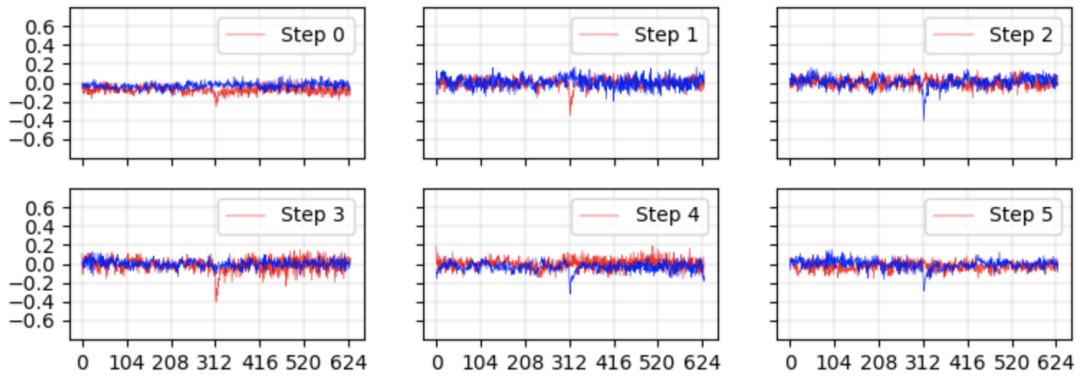



Figure 4.4: Addition of shifted PCC results with 10 segments of a single power trace. Each step corresponds to a different bit. The blue curve corresponds to a bit hypothesis of zero, while the red curve corresponds to bit hypothesis of one.

## 4.4 Countermeasures

The attack arises as a consequence of the three-point ladder being a deterministic function with public inputs. Each value going through the pipeline registers can be reduced to only two cases. These inputs depend on the public triple $x_Q, x_P, x_{Q-P}$ (which define the points $Q = [x_Q : 1]$, $P = [x_P : 1]$, $Q - P = [x_{Q-P} : 1]$), the bits of Bob's private key up to the step at which the instruction in question is being executed (which we may assume to be known by induction), and the two possibilities for the current bit of the private key.

### 4.4.1 Recommended countermeasure

A simple and low-cost countermeasure, which was also mentioned in [FGD+10, Cor99, ZYD+20], consists of randomising the coordinates that define the starting points, i.e., generate three random non-zero field elements $r_Q, r_P, r_{Q-P}$ and set

$$Q = [x_Q r_Q : r_Q],\ P = [x_P r_P : r_P],\ Q - P = [x_{Q-P} r_{Q-P} : r_{Q-P}]. \tag{4.2}$$

The increase in complexity comes from generating three random $\mathbb{F}^*_{p^2}$ elements and three field multiplications. This is negligible with respect to the overall cost of the three-point ladder. The execution of the protocol is still correct because the points $Q, P, Q - P$ are not changed, but the input of `xDBLADD`, seen as three pairs of $\mathbb{F}_{p^2}$ elements is now randomised. Since the values $r_Q, r_P, r_{Q-P}$ are secret, we cannot predict the loaded and stored values in the pipeline registers,

and thus cannot apply the same attack.

### 4.4.2 Other countermeasures

In addition to the randomised projective coordinates described above, the authors of [ZYD+20] proposed a series of countermeasures (based on [FGD+10, JT01]) against CPA on SIKE that we aim to evaluate in the case of a horizontal attack. However these countermeasures are either too expensive, or do not offer additional protection against horizontal attacks. We also comment on atomic elliptic curve algorithms.

1. **Masking the base point $Q$**
   The starting point $Q$ is masked with a random point $R$ in order to obtain $Q \leftarrow Q + R$. The final point $P + [sk](Q + R)$ of the three-point ladder is then adjusted by subtracting $[sk]R$.

   Masking the base point prevents both a vertical and a horizontal attack but cannot be done without leaving the Montgomery representation. As a result, such a countermeasure requires at least a square root computation over the field $\mathbb{F}_{p^2}$, which is very expensive.

2. **Random isomorphic elliptic curve**
   The point $Q$ is mapped to a random elliptic curve $E'$ where the scalar multiplication is computed. The result is then mapped back to the original curve $E$ in order to obtain $[sk]Q$ which is then added to $P$.

   Such a countermeasure is unfortunately limiting, since the number of curves isomorphic to $E$ is low, and finding a non-trivial isomorphism is a non-trivial task.

3. **Masking the scalar sk**
   The secret key $sk$ is masked with a random value $r$ by setting $sk \leftarrow sk + r \cdot \mathrm{ord}(Q)$.

   If the masking is different at each execution and big enough, the vertical attack can be conceivably prevented with this countermeasure. However, the horizontal attack is simply extended by $r \cdot \mathrm{ord}(Q)$ bits and recovers a value congruent to the actual $sk \pmod{\mathrm{ord}(Q)}$.

4. **Random key splitting**
   The private key $sk$ is divided randomly as $sk = sk_1 + sk_2$. Then two three-point ladders are computed in order to obtain $(P + [sk_1]Q) + [sk_2]Q$.

   While splitting $sk$ differently across executions produces measurements of dissimilar operations in a vertical attack, this countermeasure is not effective against a horizontal attack, as both shares can be independently recovered.

5. **Window-based countermeasure**
   Instead of making a binary choice for swapping at each step of the three-point ladder, a 3-bit window is used, and two additions and three doublings are computed per window.

   While a window-based method increases the complexity of a vertical attack, such a countermeasure is ineffective in the settings of a horizontal attack, as the number of guesses

per CPA iteration simply increases from $2^1$ to $2^3$. Besides, similarly as with the base point masking, this countermeasure is not cost-efficient, as the new ladder will require to leave the Montgomery representation, requiring at least one computation of a square root over $\mathbb{F}_{p^2}$.

6. **Atomic three-point ladder**
   The authors of [CCJ04] propose atomic algorithms for preventing simple side-channel analysis. An atomic algorithm consists of a sequence of instructions which are indistinguishable from a side-channel point of view.

   At the first look, the three-point ladder might seem to be atomic, however the assumption in [CCJ04] that modular operations are side-channel equivalent fails in the Cortex-M4 environment. While we are not able to distinguish a single pair of modular additions with two different inputs, we are able to distinguish 10 tuples of modular additions with two different 10-tuples of inputs, which breaks indistinguishability.

## 4.5 Conclusion

This chapter describes a CPA on SIKE in ephemeral settings that recovers the target's entire private key using a single power trace of the three-point ladder in the key decapsulation procedure. The attack was experimentally verified. A countermeasure based on point randomisation is finally suggested.

The impact of this attack on the security of SIKE is critical when the reference implementation is used in an unprotected manner on a Cortex-M4 microcontroller. This is especially important, because of the exceptionally leaky nature of such microcontrollers, thanks to the findings of [CGD18]. Due to the simplicity of the CPA, countermeasures are required to be deployed when the reference implementation of SIKE is used in an embedded environment.

We emphasise on the fact that the three-point ladder attacked in the key decapsulation is not the only point of attack of the SIKE protocol and that *each* use of the three-point ladder (even in the key generation, and key encapsulation) requires to be protected when exposed to power analyses.

# 5 Zero-value side-channel attacks on SIKE

In this chapter we provide two side-channel attacks against the official SIKE implementation protected with coordinate randomisation. Two of the attacks target the three-point ladder procedure, and the third attack targets the isogeny computation. All attacks assume a static key setting, i.e., the target party uses the same secret key to decrypt multiple ciphertexts.

The attacks are executed in an adaptive fashion — we extract consecutive bits of the target's secret key, each time assuming that the previous bits are known and adapting the attack according to the value of the extracted part of the secret key. Malicious public keys are computed and provided to the target at each step of the attack. The public keys force the target to compute a projective point with one of its coordinates equal to equal to 0 depending on a bit of the targets secret key. We shall refer to these attacks as *zero-value attacks*. A computation of the field element 0 can be noticed by side-channel, even if the projective coordinates have been randomised. We used electromagnetic emission measurements for the three-point ladder attack and power consumption measurements for the isogeny attack. In all of our experiments we managed to extract the whole secret key.

SIDH is known to be mathematically broken in a static or semi-static key environment. This is why SIKE relies on the Fujisaki-Okamoto transform to allow for semi-static keys. The purpose of the transform is precisely to validate that the ciphertext (public key) was generated honestly and to abort if not. However, the side-channel leakage that we exploit happens before the Fujisaki-Okamoto transform can prevent the attack.

Full validation of SIDH ciphertexts (and public keys) is believed to be as hard as breaking SIDH itself. Luckily, our attack can be blocked by a partial ciphertext validation, which however, comes at a price of around 10% performance overhead.

For completeness we also recall some earlier related to the subject of zero-value attacks on SIKE. Coordinate randomisation in SIKE was recommended as a countermeasure in [KPHS18, ZYD+20, GLK21]. We also mentioned it as a direct way to protect against the attack exposed in Chapter 4. Coordinate randomisation was known not to be sufficient in certain ECC implementations, and zero-value attacks were already introduced by Goubin [Gou03], Akishita and Takagi [AT03], and

Izu and Takagi [IT03].

The possibility of applying a zero-value attack to SIKE was proposed by Koziel et al. [KAJ17]. Their attack was only theoretical, and already assumed a certain type of partial ciphertext validation. Finally they propose four attacks which work despite partial ciphertext validation. We will argue that those earlier attacks do not apply to the current SIKE standard.

## 5.1 Background

Elliptic curves and points on elliptic curves in SIKE are defined over a quadratic field $\mathbb{F}_{p^2}$ and represented with a pair of field elements $[X : Z]$. The pair $[X : Z]$ and $[\xi X : \xi Z]$ for $\xi \in \mathbb{F}_{p^2}^{\times}$ represent the same projective pair.

We want to single out the projective points which have one of the coordinates equal to zero and for that end we introduce new names for certain types of elliptic curves and elliptic curve points.

### 5.1.1 Elliptic curves

Depending on the subroutine of SIKE, an elliptic curve $E : y^2 = x^3 + ax^2 + x$ can be represented in projective coordinates in three different ways as shown in Section 2.5.2, page 25. We recall the representations here, referring to Section 2.5.2 for definitions of $A$ and $C$:

- $[A : C]$ where $[A : C] = [a : 1]$,
- $[A^+_{24} : C_{24}]$ where $[A^+_{24} : C_{24}] = [A + 2C : 4C]$,
- $[A^+_{24} : A^-_{24}]$ where $[A^+_{24} : A^-_{24}] = [A + 2C : A - 2C]$.

Considering all possibilities for an elliptic curve representations, invalid states included, an elliptic curve in SIKE may fall into one of the following categories:

**The *undefined* curve,** represented by pairs $[A : C] = [A^+_{24} : C_{24}] = [A^+_{24} : A^-_{24}] = [0 : 0]$. This does not represent any algebraic object.

**The *degenerate* curve,** represented by pairs $[A : C] = [A^+_{24} : C_{24}] = [1 : 0]$ or $[A^+_{24} : A^-_{24}] = [1 : 1]$. This is not, properly speaking, a curve.

**The *singular* curves,** represented by pairs $[A : C] = [\pm 2 : 1]$ or $[A^+_{24} : C_{24}] \in \{[0 : 1], [1 : 1]\}$ or $[A^+_{24} : A^-_{24}] \in \{[0 : 1], [1 : 0]\}$. These are not elliptic curves, because they exhibit a singularity in $(\mp 1, 0)$ and behave often as exceptional points in formulas.

**Elliptic curves,** not corresponding to any of the above, or alternatively curves represented by $[A : C] = [a : 1]$ with $a \neq \pm 2$.

### 5.1.2 Points on elliptic curves

Points on elliptic curve in SIKE are represented in Montgomery coordinates (see Section 2.5.2, page 26), i.e., only the $X$ and $Z$ coordinate are used.

Considering all possibilities for point representations, invalid states included, an elliptic curve point in SIKE may fall into one of the following categories:

**The *undefined* point** $[0:0]$, this does not correspond to any algebraic point.

**The *identity*** $\mathcal{O} := [1:0]$, the identity of the elliptic curve group law.

**The *distinguished* point** $T := [0:1]$, of order 2. This point is a member of all Montgomery curves. Assuming the curve is well defined, $[2]T = \mathcal{O}$.

**The *special* 4-torsion points** $[\pm 1:1]$, these points are members of all Montgomery curves. Assuming the curve is well defined $[2]P = T$ for any such point.

**An ordinary point** $[X:Z]$ not belonging to any of the above. Assuming the curve $[A:C]$ is well defined, such a point is on the curve $[A:C]$ if $X/Z + A/C + Z/X$ is a square in $\mathbb{F}_{p^2}$.

## 5.2 The three-point ladder attacks

In this section we provide two attacks based on forcing the target party to compute the elliptic curve point $\mathcal{O} = [1:0]$ or $T = [0:1]$ during an execution of the three-point ladder. The attack is performed in multiple steps. At each step we assume that we know the first $i-1$ bits of the secret key of the target party, and the attack targets the $i^{\text{th}}$ bit.

Our final goal is to see where and how such points can occur, and to force a computing party to compute such points based on a secret bit of their private key. The attack is done adaptively in an extend-and-prune manner, that is, we perform the attack in multiple steps; at each step we recover one bit of the private key by assuming that we know the previous parts. We write the secret key as $sk = sk_0 2^0 + sk_1 2^1 + \ldots$ with $sk_i \in \{0,1\}$ and we define $sk_{<k} := sk_0 2^0 + sk_1 2^1 + \cdots + sk_{k-1} 2^{k-1}$.

A careful analysis of the three-point ladder (as described in Section 2.5.4 and Algorithm 5, page 29) shows that an input $(Q, P, Q-P)$, the value of the updated triple $(R_0, R_2, R)$ after the $k^{\text{th}}$ loop iteration is

$$([2^{k+1}]Q, \quad -\boldsymbol{P} + [\boldsymbol{2^{k+1} - sk_{<k}}]\boldsymbol{Q}, \quad P + [sk_{<k}]Q) \quad \text{if } sk_k = 0, \tag{5.1}$$

$$([2^{k+1}]Q, \quad \boldsymbol{P} + [\boldsymbol{2^k + sk_{<k}}]\boldsymbol{Q}, \quad -P + [2^k - sk_{<k}]Q) \quad \text{if } sk_k = 1. \tag{5.2}$$

We show how to create a public key $(Q, P, Q-P)$ such that the second output point (in bold in Equations (5.1),(5.2)) is a zero-value point if and only if $sk_k = 0$ (or $sk_k = 1$).

### 5.2.1 Forcing $\mathcal{O}$

The second output point in Equation (5.1) is equal to $\mathcal{O}$ if $P = [2^{k+1} - sk_{<k}]Q$. For the second point to be equal to $\mathcal{O}$ in Equation (5.2) we must have $P = -[2^k + sk_{<k}]Q$. This simple equation allows us to define the following pair of malicious public key points:

Table 5.1: Malicious public key points which forces the computation of $\mathcal{O}$

| Public key | $Q$ | $P$ | $Q-P$ |
|---|---|---|---|
| $pk^{\mathcal{O}}_{k,0}$ | Point of order $3^{e_3}$ | $[2^{k+1} - sk_{<k}]Q$ | $-[2^{k+1} - sk_{<k} - 1]Q$ |
| $pk^{\mathcal{O}}_{k,1}$ | Point of order $3^{e_3}$ | $-[2^k + sk_{<k}]Q$ | $[2^k + sk_{<k} + 1]Q$ |

Note that $P, Q-P \notin \{\mathcal{O}, T\}$, so the public key passes the assumption in Algorithm 5, page 29. Plugging in $pk^{\mathcal{O}}_{k,0}$ (resp. $pk^{\mathcal{O}}_{k,1}$) in Equation (5.1) (resp. Equation (5.2)) shows that on correct guess, the target point becomes $\mathcal{O}$. Furthermore, if the guess is incorrect, then the point is $[2^{k+1} + 2^k]Q$ which has both coordinates non-zero.

### 5.2.2 Forcing $T$

The second output point in Equation (5.1) is equal to $T$ if $P = [2^{k+1} - sk_{<k}]Q + T$. For the second point to be equal to $T$ in Equation (5.2) we must have $P = -[2^k + sk_{<k}]Q - T$. This simple equation allows us to define the following pair of malicious public key points:

Table 5.2: Malicious public key points which forces the computation of $T$

| Public key | $Q$ | $P$ | $Q-P$ |
|---|---|---|---|
| $pk^{T}_{k,0}$ | Point of order $3^{e_3}$ | $[2^{k+1} - sk_{<k}]Q + T$ | $-[2^{k+1} - sk_{<k} - 1]Q - T$ |
| $pk^{T}_{k,1}$ | Point of order $3^{e_3}$ | $-[2^k + sk_{<k}]Q - T$ | $[2^k + sk_{<k} + 1]Q + T$ |

Note that $P, Q-P \notin \{\mathcal{O}, T\}$, so the public key passes the assumption in Algorithm 5, page 29. Plugging in $pk^{T}_{k,0}$ (resp. $pk^{T}_{k,1}$) in Equation (5.1) (resp. Equation (5.2)) shows that on correct guess, the target point becomes $T$. Furthermore, if the guess is incorrect, then the point is $[2^{k+1} + 2^k]Q$ which has both coordinates non-zero.

## 5.3 Isogeny attack

This section provides the attack on the isogeny computation (See Section 2.5.5, page 30 for the definition of isogeny computation). Our goal is to manipulate the isogeny kernel generating point $R$ of the target party so that it becomes incompatible with isogeny formulas, i.e., the order of the kernel point is different from the order of the isogeny. This leads either to computation of undefined points $[0:0]$ (which propagate indefinitely) or to completely (heuristically) random computations. The two cases can be easily distinguished by use of side channels.

In particular we propose that the distinguishing is executed during the computation of the $j$-invariant. The reason is that one of the subprocedures of the $j$-invariant is an $\mathbb{F}_p$ inversion which is computed as a sequence of circa $\log p$ hardcoded squarings and multiplications. Distinguishing if this sequence started from the element 0 or from a random element is an easy matter.

We assume that the target party computes an isogeny of degree $\ell_I^{e_I}$, and the other party is expected to have computed an isogeny of degree $\ell_J^{e_J}$ before sharing their public key. In practice $\ell_I^{e_I} = 3^{e_3}$ and $\ell_J^{e_J} = 2^{e_2}$, but the attack works also if we swap the parties, or more generally on any set of SIKE parameters. We drop the subscript $J$, and write $(\ell, e)$ for $(\ell_J, e_J)$ from now on.

We assume that the target's secret key is written in basis $\ell$, so $sk = sk_0\ell^0 + sk_1\ell^1 + \ldots$ with $sk_i \in \{0, 1, \ldots, \ell - 1\}$. Furthermore we define $sk_{<k} := sk_0\ell^0 + sk_1\ell^1 + \ldots + sk_{k-1}\ell^{k-1}$.

The isogeny algorithm uses a hard coded strategy, and attempts to compute an $\ell_I^{e_I}$-isogeny independently of the actual order of the kernel point $R$. We send the target party a malicious public key made of points in the $\ell^e$-torsion subgroup $E[\ell^e]$ (as opposed to $E[\ell_I^{e_I}]$ which is expected by the algorithm). Such public keys force the kernel generating point $R$ to be of degree a power of $\ell$.

**Definition 5.3.1.** We denote with $v_\ell$ the $\ell$-adic valuation, defined, for $a \in \mathbb{Z}^\times$, as

$$v_\ell(a) = \max\{x \in \mathbb{Z} \mid \ell^x \text{ divides } a\}. \tag{5.3}$$

*Remark.* By an abuse of notation we set $v_\ell(R) = v_\ell(\text{ord}(R))$ for any elliptic curve point $R$.

Furthermore we can control the exact order of the point $R$ depending on a single $\ell$-digit of the secret key, $sk_k$, and assuming we know $sk_{<k}$. For example, given some $0 < e' \leq e$ and $0 \leq \ell' < \ell$ we can compute a malicious public key such that

$$v_\ell(R) = \begin{cases} e' & \text{if } sk_k \neq \ell', \\ e' - 1 & \text{if } sk_k = \ell'. \end{cases} \tag{5.4}$$

In addition to that we will show that there is an exponent $o > 0$ which satisfies so-called *leakage properties* for any point $R \in E[\ell^e]$ which is provided as a kernel of an $\ell_I^{e_I}$-isogeny:

L1. If $v_\ell(R) < o$ then the isogeny eventually computes undefined points $[0:0]$.
L2. If $v_\ell(R) \geq o$ then the isogeny computes random values.

The exponent $o$ depends on the underlying prime $p$ (more precisely the $\ell_I^{e_I}$ and $\ell^e$) and the tree-traversal strategy. These are public parameters and $o$ can therefore be precomputed for any set of SIKE parameters.

In this section, we first show how an adversary can control the order of a point in such a way that it depends on the value of an $\ell$-digit of the private-key. Then we show that there exists an exponent $o$ which satisfies the leakage properties (L1., L2.).

### 5.3.1 Controlling the kernel point order

We show how to create a public key, which when used by the target party produces a kernel point $R$ of constrained order. In particular, given an exponent $0 \le e' \le e$, an $\ell$-digit $\ell'$, and $sk_{<k}$, we want to force the target party to compute a point whose order is determined by formula 5.4.

That is done by creating a public key $pk_k^j$ as shown in Algorithm 9.

**Algorithm 9:** MALICIOUS PUBLIC KEY GENERATION

**Input:** Index of bit being guessed $k$, known part of secret key $sk_{<k}$,
an exponent $e'$, $\ell$-digit $\ell'$
**Assumes:** $0 < e' \le e$, $0 \le \ell' < \ell$ and $0 \le k \le e - e'$
**Output:** Public key $pk_k^j = (Q, P, Q-P)$.
**Procedure** `pk_j` $(k, sk_{<k}, e', \ell')$

$E \leftarrow$ any supersingular elliptic curve
$P_\ell, Q_\ell \leftarrow$ generators of $E[\ell^e]$
Assume $[\ell^{e-1}]Q_\ell \neq T$ `// Swap` $P_\ell, Q_\ell$ `otherwise`
$S = [\ell^{e-(e'-1)}]P_\ell$ `//` $v_\ell(S) = e' - 1$
$Q = [\ell^{e-(e'+k)}]Q_\ell$ `//` $v_\ell(Q) = e' + k$
$P = S - [sk_{<k} + \ell'\,\ell^k]Q$ `//` $v_\ell(P) \in \{e'-1, \ldots, e'+k\}$
**return** $(Q, P, Q-P)$

The kernel generating point $R$ obtained from the public key shown in Algorithm 9 satisfies the order constraint as is proved in the following lemma.

**Lemma 5.3.1.** Let $0 < e' \le e$, let $\ell' \in \{0, 1, \ldots, \ell - 1\}$, and assume that $pk_k^j$ is the output of `pk_j`$(k, sk_{<k}, e', \ell')$. Then the kernel generator point $R = P + [sk]Q$ generated from the public key $pk_k^j$ of Algorithm 9 satisfies

$$v_\ell(R) = \begin{cases} e' & \text{if } sk_k \neq \ell', \\ e' - 1 & \text{if } sk_k = \ell'. \end{cases} \tag{5.5}$$

*Proof.* Following from Algorithm 9, we have $v_\ell(S) = e' - 1$ and $v_\ell(Q) = e' + k$. Denote with $Q' := [\ell^k]Q$, where we have $v_\ell(Q') = e'$. It follows that

$$\begin{aligned} P + [sk]Q &= S + [sk - sk_{<k} - \ell'\ell^k]Q \\ &= S + [sk_k - \ell']Q' + [sk_{k+1}\ell]Q' + \ldots \\ &= S + [sk_k - \ell']Q' + Q'', \end{aligned}$$

where $Q'' = [sk_{k+1}\ell]Q' + \ldots$. The point $Q''$ is independent from $S$ by construction and it satisfies $v_\ell(Q'') \le e' - 1$.

If $sk_k = \ell'$ then $R = S + Q''$. The point $R$ is the sum of two independent points which satisfy $v_\ell(S) = e' - 1$ and $v_\ell(Q'') \le e' - 1$. Therefore $v_\ell(R) = e' - 1$.

If $sk_k \neq \ell'$ then $R$ is a sum of two independent points of valuation $e'-1$ and $e'$ respectively. Therefore $v_\ell(R) = e'$. □

### 5.3.2 Evaluating formulas on bad points

Computing the isogeny with a kernel of incompatible order, i.e. order different from isogeny degree, leads to irregular behaviour. During the execution of the $\ell_I^{e_I}$-isogeny, geometric structure will be lost, and we will essentially work with random points on random elliptic curves. The goal of this section is to show when irregular behaviour starts, and what types of unexpected behaviour can happen.

The $\ell_I^{e_I}$-isogeny is computed by means of a hard-coded sequence of sub-algorithms which include point multiplication by $\ell_I$, degree $\ell_I$ isogeny computation, degree $\ell_I$ isogeny evaluation, and saving and loading a point. The order in which these steps are executed is encoded in a strategy as explained in Section 9 on page 31. Because saving or loading a point bears no algebraic structure, we will only analyse the remaining three operations.

The isogenies that are directly computed in SIKE are of degree 2, 3 or 4. They are computed by Vélu's formulas for Montgomery curves. As such they satisfy the following list of properties:

- P1 If the kernel point is of incorrect order, then the image point (resp. curve) is arbitrary and does not share any known geometric relation with the preimage point (resp. curve).
- P2 If the image of $[X:Z]$ is $[U:V]$, then the image of $[Z:X]$ is $[V:U]$.
- P3 If the input point is equal to the kernel point, then the output is $\mathcal{O}$.
- P4 If the kernel point is $[X:Z]$ and the input is $[Z:X]$, then the output point is $T$.
- P5 The image of $\mathcal{O}$ is $\mathcal{O}$.
- P6 The image of $T$ is $T$.

The first statement is not proven and is based on heuristics. Given our experiments (Section 5.4.4, page 84) and the current understanding of elliptic curve isogenies, there was no evidence to show the contrary. The other statements follow from the way the isogenies were constructed, see [Ren18].

*Remark.* The points $\mathcal{O}, T, [\pm 1:1]$ are contained in all supersingular Montgomery curves. The points $\mathcal{O}$ and $T$ are contained in all Montgomery curves. We call the points $\mathcal{O}, T, [\pm 1:1]$ *special*.

Furthermore the distinguished point $T$ induces an involution $P \mapsto P+T$ which is given by $[X:Z] \mapsto [Z:X]$. These properties stem from the definition of Montgomery curves.

**Point multiplication by $\ell_I$.** The only point multiplication formulas used in SIKE are doublings (see Equation 2.37 on page 27) and triplings (see Equation 2.39 on page 27). They are used during computations of $2^{e_2}$ and $3^{e_3}$ respectively.

The formulas for point multiplications can sometimes output undefined points $[0:0]$ when the

underlying curve is degenerate, or singular. We show when this can occur in Table 5.3.

Table 5.3: Doubling and tripling on degenerate and singular curves

| | Doubling | | | Tripling | | |
|---|---|---|---|---|---|---|
| Input point | Input curve $[A_{24}^+ : C_{24}]$ | | | Input curve $[A_{24}^+ : A_{24}^-]$ | | |
| | $[1:0]$ | $[1:1]$ | $[0:1]$ | $[1:1]$ | $[1:0]$ | $[0:1]$ |
| $\mathcal{O}$ | $[0:0]$ | $\mathcal{O}$ | $\mathcal{O}$ | $[0:0]$ | $\mathcal{O}$ | $\mathcal{O}$ |
| $T$ | $[0:0]$ | $\mathcal{O}$ | $\mathcal{O}$ | $[0:0]$ | $T$ | $T$ |
| $[1:1]$ | $T$ | $T$ | $[0:0]$ | $[1:1]$ | $[1:1]$ | $[0:0]$ |
| $[-1:1]$ | $T$ | $[0:0]$ | $T$ | $[-1:1]$ | $[0:0]$ | $[-1:1]$ |

**$\ell_I$-isogenies.** SIKE uses isogenies of degree 2, 3 and 4 as sub-algorithms for computing large degree isogenies. In particular, a $2^{e_2}$-isogeny is computed as a composition of ($e_2 \mod 2$) degree 2 isogenies and $\lfloor \frac{e_2}{2} \rfloor$ degree 4 isogenies. A $3^{e_3}$-isogeny is computed as a composition of $e_3$ degree 3 isogenies.

The formulas for 2, 3 and 4-isogenies can output undefined or degenerate values when the inputs are incompatible. The output curves can be degenerate when the kernel points are *special*. The output points can be undefined $[0:0]$ if both the kernel point and the input point are *special*. These occurrences are listed in Table 5.4 and Table 5.5.

Table 5.4: Isogeny computation with special kernels

| Isogeny degree | Curve representation | Kernel point | | | |
|---|---|---|---|---|---|
| | | $\mathcal{O}$ | $T$ | $[1:1]$ | $[-1:1]$ |
| 2 | $[A_{24}^+ : C_{24}]$ | $[1:0]$ | $[1:1]$ | $[0:1]$ | $[0:1]$ |
| 3 | $[A_{24}^+ : A_{24}^-]$ | $[1:1]$ | $[1:1]$ | $[1:0]$ | $[0:1]$ |
| 4 | $[A_{24}^+ : C_{24}]$ | $[1:0]$ | $[0:1]$ | $[1:1]$ | $[1:1]$ |

### 5.3.3 Traversal of the first isogeny branch

Almost all the analysis of the isogeny computation boils down to analysing the computations done on the public curve $E$ recovered from $pk_k^j$ (i.e., the first vertical branch in the tree traversal).

**Definition 5.3.2.** Given a strategy $S = (s_1, \ldots, s_{e_I-1})$ for computing an $\ell_I^{e_I}$-isogeny from kernel $\langle R \rangle$, we define $\mathrm{Ind}_S$ to be the set of indices of points $R_i = [\ell_I^i]R$ which are pushed by the first $\ell_I$-isogeny. More formally

$$\mathrm{Ind}_S = \{0, s_1, s_1+s_2, \ldots, s_1+\ldots+s_k\} \text{ where } k = \max\{i \mid s_1+\ldots+s_i < e_I - 1\} \tag{5.6}$$

For example, the set of indices of two strategies from Figure 2.6 on page 32 are $\{0\}$ and $\{0,3,5\}$.

Table 5.5: Isogeny evaluation on special points

| | Input point | Kernel point $\mathcal{O}$ | $T$ | $[1:1]$ | $[-1:1]$ |
|---|---|---|---|---|---|
| 2-isogeny | $\mathcal{O}$ | $\mathcal{O}$ | $[0:0]$ | $\mathcal{O}$ | $\mathcal{O}$ |
| | $T$ | $T$ | $[0:0]$ | $T$ | $T$ |
| | $[1:1]$ | $[-1:1]$ | $[-1:1]$ | $[0:0]$ | $[-1:1]$ |
| | $[-1:1]$ | $[-1:1]$ | $[-1:1]$ | $[-1:1]$ | $[0:0]$ |
| 3-isogeny | $\mathcal{O}$ | $\mathcal{O}$ | $[0:0]$ | $\mathcal{O}$ | $[0:0]$ |
| | $T$ | $T$ | $[0:0]$ | $T$ | $[0:0]$ |
| | $[1:1]$ | $[1:1]$ | $[1:1]$ | $[0:0]$ | $[1:1]$ |
| | $[-1:1]$ | $[-1:1]$ | $[-1:1]$ | $[-1:1]$ | $[0:0]$ |
| 4-isogeny | $\mathcal{O}$ | $\mathcal{O}$ | $[0:0]$ | $\mathcal{O}$ | $[0:0]$ |
| | $T$ | $T$ | $[0:0]$ | $T$ | $[0:0]$ |
| | $[1:1]$ | $[1:1]$ | $[1:1]$ | $[0:0]$ | $[1:1]$ |
| | $[-1:1]$ | $[1:1]$ | $[1:1]$ | $[1:1]$ | $[0:0]$ |

The process of computing the first step of an $\ell_I^{e_I}$-isogeny is shown in Algorithm 10.

**Algorithm 10:** FIRST VERTICAL BRANCH OF THE TREE-TRAVERSAL

**Input:** Kernel point $R$, starting curve $E$, set of indices Ind
**Assumes:** Ind stems from a strategy $S$, $R \in E$ of order $\ell_I^{e_I}$
**Output:** Image curve computed by the first $\ell_I$-isogeny $E'$,
Images of points evaluated by the first $\ell_I$-isogeny $\{\phi([\ell_I^i]R)\}_{i\in\text{Ind}}$
**Procedure** `first_iso(`$R, E,$ Ind`)`

1 **for** $i = 0$ **to** $e_I - 2$ **do**
2  **if** $i \in$ Ind **then**
3  $R_i = R$
4  $R = [\ell_I]R$
5 $\phi : E \to E/\langle R\rangle$  `// Degree` $\ell_I$ `isogeny with kernel` $\langle[\ell_I^{e_I-1}]R\rangle$
6 $E' = E/\langle R\rangle$
7 **for** $i \in$ Ind **do**
8  $R_i = \phi(R_i)$
 **return** $(E', \{R_i\}_{i\in\text{Ind}})$

### 5.3.4 Computing the isogeny with a bad kernel

In this subsection we will prove the existence of the exponent $o$ which satisfies the leakage properties (L1., L2.) from page 73.

Let $R$ be a point of order $\ell^{e'}$ provided as kernel to the isogeny of degree $\ell_I^{e_I}$. Let $E$ be the starting curve of the isogeny and let $\text{Ind}_S$, $E'$ and $R_i$ be as in Section 5.3.3.

Following from Section 5.3.2, the first image curve $E'$ is an arbitrary, generally non-supersingular curve. After the first isogeny is computed, all the following elliptic curves and points are arbitrary. The only deterministic "structure" that the points can carry is that some of them may have equal projective coordinates. Recall the properties on page 75. There are three ways in which the isogeny computation may proceed:

(i) **There is a pair of saved points with equal coordinates.** Assume that $R_\alpha$ and $R_\beta$ are projectively equivalent and $\alpha < \beta \in \mathrm{Ind}_S$. As the points are equal, their images through consecutive $\ell_I$-isogenies will stay equal until we compute the isogeny generated by some image of $R_\beta$. This isogeny will send the corresponding image of $R_\alpha$ to $\mathcal{O}$ (prop. P3). The point $\mathcal{O}$ is fixed by multiplications by $\ell_I$ and $\ell_I$-isogenies (prop. P5). Eventually an isogeny of kernel $\langle\mathcal{O}\rangle$ is computed, whose image curve is the degenerate curve (Table 5.4). The images of points $\mathcal{O}$ are also $\mathcal{O}$ (Table 5.5). The first multiplication of $\mathcal{O}$ by $\ell_I$ on the degenerate curve outputs the undefined point $[0:0]$ (Table 5.3). From this point onward all values will be 0, and the final $j$-invariant will be computed as $0/0$.

(ii) **There is a pair of saved points with *flipped* coordinates.** Assume $R_\alpha = [X:Z]$ and $R_\beta = [Z:X]$ and $\alpha < \beta \in \mathrm{Ind}_S$. This can only happen if $R_\alpha = R_\beta + T$ which implies that $R$ and $T$ are dependent. In particular the order of $R$ is even, so the isogeny being computed must be a $3^{e_3}$-isogeny. The property of $R_\alpha, R_\beta$ having *flipped* coordinates is preserved (prop. P2) until the image of $R_\beta$ is used to compute an isogeny. This isogeny will send the image of $R_\alpha$ to $T$ (prop. P4). The point $T$ is fixed by point tripling and 3-isogenies (prop. P6). Eventually an isogeny of kernel $\langle T\rangle$ is computed, whose image curve is the degenerate curve (Table 5.4). The first next image of $T$ under the isogeny generated by $T$ is the undefined point $[0:0]$ (Table 5.5). From that point onward all values will be 0, and the final $j$-invariant will be computed as $0/0$.

(iii) **There are no points with equal nor *flipped* coordinates.** The points and curves became arbitrary after computing the first $\ell_I$-isogeny (prop. P1). From this point onward we have different arbitrary values which propagate (prop. P1). The final curve and its $j$-invariant is random.

A visual example of isogeny computation in (i) is given in Appendix D.

Note that (i) is equivalent to the existence of $\alpha < \beta \in \mathrm{Ind}_S$ such that $[\ell_I^\alpha]R = \pm[\ell_I^\beta]R$, while (ii) is equivalent to $[\ell_I^\alpha]R = \pm[\ell_I^\beta]R + T$. In (iii), such $\alpha < \beta \in \mathrm{Ind}_S$ simply do not exist. These properties are characterized by the order of point $R$ as shown in the following lemma.

**Lemma 5.3.2.** For each set of SIKE parameters, let $S$ be a strategy for computing the $\ell_I^{e_I}$ isogeny and let $R$ be a point in the $\ell^e$ torsion. Furthermore assume that $R$ and $T$ are independent. Then there is an exponent $o > 0$ such that:

1. $v_\ell(R) \leq o - 1$ if and only if (i) applies,
2. $v_\ell(R) \geq o$ if and only if (iii) applies.

If $R$ and $T$ are dependent, then the following is true for the same exponent $o$:

1. $v_\ell(R) \leq o$ if and only if (i) or (ii) applies,
2. $v_\ell(R) \geq o+1$ if and only if (iii) applies.

*Proof.* The statement (i) can alternatively be expressed as: "there are $\alpha < \beta \in \mathrm{Ind}_S$ such that $R_\alpha = \pm R_\beta$, that is, $[\ell_I^\alpha]R = \pm[\ell_I^\beta]R$". Given that $R$ is a point of order $\ell^{v_\ell(R)}$, the previous statement reduces to a modular equivalence, more precisely $\ell_I^\alpha \mp \ell_I^\beta \equiv 0 \pmod{2^r}$, where we define $r := v_\ell(R)$.

This equation is certainly satisfied for $r = 0$, for all $\alpha, \beta \in \mathrm{Ind}_S$. Furthermore, an equality modulo $2^r$ for some $\alpha, \beta$ reduces to an equality modulo $2^{r'}$ for all $r' \leq r$. Therefore it is only left to prove that the equation is not satisfied for some $r \leq e$, for all $\alpha, \beta \in \mathrm{Ind}_S$. This is proven by observing SIKE parameters, in particular the strategies. We call the smallest such exponent $o$ the *break-point exponent*.

The statement (ii) can alternatively be expressed as: "there exist values $\alpha < \beta \in \mathrm{Ind}_S$ such that $R_\alpha = \pm R_\beta + T$". If $R$ and $T$ are independent, this cannot happen. Therefore, the first part of the lemma is proven.

If $R$ and $T$ are dependent, we must have $\ell = 2$ and $[2^{r-1}]R = T$. Furthermore $\ell_I = 3$, and (ii) reduces to $3^\alpha \mp 3^\beta - 2^{r-1} \equiv 0 \pmod{2^r}$. This equation is equivalent to the following two equations: $3^\alpha \mp 3^\beta \equiv 0 \pmod{2^{r-1}}$ and $3^\alpha \mp 3^\beta \not\equiv 0 \pmod{2^r}$. By the definition of the break-point exponent, $3^\alpha \mp 3^\beta \equiv 0 \pmod{2^{o-1}}$ and $3^\alpha \mp 3^\beta \not\equiv 0 \pmod{2^o}$, therefore $3^\alpha \mp 3^\beta - 2^o \equiv 0 \pmod{2^{o+1}}$ for some $\alpha, \beta$. On the other hand, $3^\alpha \mp 3^\beta \not\equiv 0 \pmod{2^o}$ for all $\alpha, \beta$, implies $3^\alpha \mp 3^\beta - 2^o \not\equiv 0 \pmod{2^{o+1}}$ for all $\alpha, \beta$. This finishes the proof. □

We report the values for $o$ for both parties and all parameter sets in Table 5.6.

| SIKE instances | p434 | p503 | p610 | p751 |
|---|---|---|---|---|
| $2^{e_2}$-isogeny | 3 | 4 | 2 | 5 |
| $3^{e_3}$-isogeny | 9 | 7 | 7 | 8 |

Table 5.6: Break-point exponents $o$ for all parameter sizes.

**Side-channel attack.** Using Lemma 5.3.2, two different outcomes of the isogeny computation can be forced depending on the value of a secret $\ell$-digit: the party either computes only zero values from a certain point in the tree traversal and onward, or completely random values. When zero values can be distinguished from random ones with a side channel, such a behaviour enables an adaptive bit-by-bit key recovery.

We propose to perform the zero-value distinction on the subroutine responsible for the subfield

inversion within the $j$-invariant computation. This is because the $j$-invariant computation occurs at one of the last steps of the key exchange, making it a conveniently identifiable target, and because the subfield inversion is usually computed as $x^{-1} = x^{p-2}$; a noticeable sequence of $\geq 400$ similar field operations. This scenario is illustrated in Algorithm 11.

**Algorithm 11:** ATTACK ON THE THE ISOGENY COMPUTATION

**Input:** Breaking point $o$.
**Output:** The secret key $sk$.
**Procedure** `attack(`$o$`)`
  **for** $k = 0$ **to** $e - o$ **do**
1  Assume we know $sk_{<k} = \sum_{i=0}^{k-1} sk_i \ell^i$
    **for** $\ell' = 0$ **to** $\ell - 1$ **do**
2  $pk_k^j \leftarrow$ `pk_j(`$k, sk_{<k}, o, \ell'$`)` `// Algorithm 9, page 74.`
3  Send $pk_k^j$ to the target
      *Side-channel analysis of exponentiation*
4  **if** computation of $0^{p-2}$ is detected **then**
5  $sk_k = \ell'$
6  Brute-force the remaining $\ell$-digits of the secret key
  **return** $sk$

Attacks on all parameters, which the reader can simulate by running a simple python script, are included in https://github.com/nKolja/SIKE-zero-value-attacks. A visualisation of the attack on all parameter sets is provided in Appendix E, where we explain how the isogeny computation evolves in terms of tree traversal graph, and explicitly show when the first undefined point is computed.

**Attack on the reference implementation.** For completeness we also mention how the attack works on the reference implementation of SIKE, which is a part of the official SIKE submission to the NIST post-quantum standardisation process. The reference implementation represents elliptic curve points in affine coordinates, which makes it very inefficient due to the excessive number of field inversions that are computed. When compared to the Cortex-M4 implementation, the reference implementation is around 180 times slower. For this reason this version is generally ignored in analyses of SIKE.

The reference implementation uses GMP [GMP91] for field arithmetic, and in particular computes field inversions by means of a built-in inversion function. This function returns an error whenever the field element 0 is provided as input. Therefore, our attack can be extended to a timing attack or even a long distance attack if the reference implementation is used by the target.

## 5.4 Experimental evaluations

The attacks on both the three-point ladder and the isogeny computation were verified in practice. The three-point ladder was attacked by means of electromagnetic analysis, and the isogeny attack was performed by means of power consumption analysis. The details on the hardware used in the attack can be found in [DEG+22].

### 5.4.1 Software

The attacked software calls the functions of the recommended implementation of SIKEp434 for 32-bit Cortex-M4 microcontrollers with input ciphertexts received from the computer.

Moreover, the scalar multiplication of the library is protected with coordinate randomisation. As the original library does not offer such a countermeasure, coordinates are randomised after computing the coefficient of the received curve, and before the Montgomery three-point ladder. A random representation of the points $(Q, P, Q-P)$ is generated from the received affine coordinates $(x_Q, x_P, x_{Q-P})$. This countermeasure consumes $6 \times \log_2(p)$ random bits to generate three random $Z$ coordinates and requires three $\mathbb{F}_{p^2}$ multiplications.

Finally, the code is further modified by adding a trigger. When toggled, the oscilloscope is notified to start the capture of the electromagnetic activity or power consumption. The purpose of such a modification is to make the collection of traces more convenient. Note, however, that the attack is still applicable without a trigger and that the synchronization of traces can be performed using, e.g., cross-correlation techniques [DPN+16].

### 5.4.2 Distinguishing zero values

There exist many ways to distinguish zero values in a power or EM trace. For instance, [Gou01, AT03] argue that a zero value can be observed in a single measurement by noticing a significant drop of power consumption or EM radiations. In practice, however, this method requires setting a manual threshold based on observing the measured samples. As a result, to remove the hassle of detecting zero values manually, we propose to make the distinction by comparing a trace (or a collection of traces) to another. We use two different types of tests for comparing traces – the $t$-test and collision power analysis.

**t-test.** Welch's $t$-test [SM15] is a statistical hypothesis testing method that examines whether two classes of traces were sampled from indistinguishable populations.

In our case, such a test is used to compare a collection of multiple traces of (known to be) non-zero value executions against a collection of multiple traces that may or may not exhibit zero values depending on a secret bit.

The test proceeds by computing $t$ statistics based on the observed means and variances of two power sample or electromagnetic sample populations. Significant $t$ values imply that the two collections are *not* indistinguishable.

When attacking the three-point ladder, some $t$ values are thus expected to be large when a zero is processed in the target trace collection, since zero values are not affected by coordinate randomisation. When non-zero values are processed in both classes, all the $t$ values are expected to be small, even when identical inputs were provided to the three-point ladder due to the randomisation of coordinates [KAJ17].

**Collision power analysis.** When the target operation can be forced to process both zero and random (i.e., non-zero) values regardless of the value for the private key, a more efficient approach can be employed based on collision power analysis [SWP03, MME10]. In this case, the values processed in a trace are detected by comparing the trace against two baselines (i.e., templates). The baselines correspond to power traces relating the same execution as the target trace but in which processed values are known to be zero and random. Such a scenario is applicable to the $j$-invariant computation, as the entire procedure processes zero values when a special input is provided.

In practice, a collision power analysis is typically mounted using Pearson's Correlation Coefficient (PCC). This technique correlates a target power trace $\mathrm{Tr} \in \mathbb{R}^m$ with a baseline $\mathrm{B} \in \mathbb{R}^m$ by computing $\rho(\mathrm{Tr}, \mathrm{B}) = \mathrm{Cov}(\mathrm{Tr}, \mathrm{B}) / (\sigma_{\mathrm{Tr}} \sigma_{\mathrm{B}})$ (where Cov is the covariance, and $\sigma$ the standard deviation). The greater the value of the coefficient, the more correlated the trace is to the baseline. As a result, a zero value is detected when the corresponding trace has a greater correlation coefficient with the zero-valued baseline than with the random-valued one.

### 5.4.3 Three-point ladder

The following sections describe a proof of concept for the attack on the three-point ladder. We experimentally verified the attack from Section 5.2.2 where the target is forced to compute the distinguished point $T$.

#### Experimental procedure

The victim board runs the decapsulation routine, and computes the three-point ladder to obtain $P + [sk]Q$. The computer sends the public key triplets with randomised coordinates to the board. To find a bit $sk_k$, we will compare the electromagnetic emissions of the board performing the ladder computations with three types of input:

- A random, correct triplet of points,
- A malicious triplet $pk^T_{k,0}$ and
- A malicious triplet $pk^T_{k,1}$.

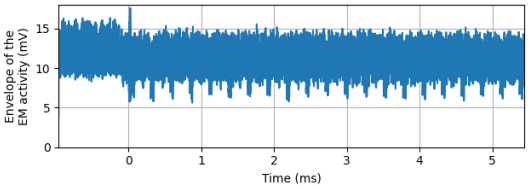



Figure 5.1: Trace for the wrong hypothesis.

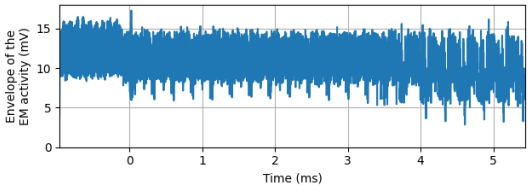



Figure 5.2: Trace for the correct hypothesis.

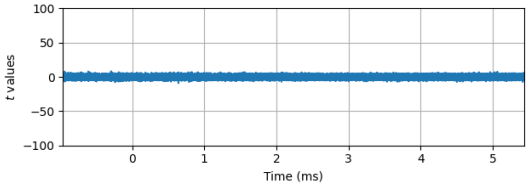



Figure 5.3: t-test for the wrong hypothesis.

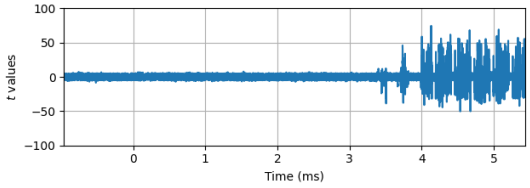



Figure 5.4: t-test for the correct hypothesis.

Figure 5.5: Experimental results for an attack on one bit.

For each of the cases, the attacker records multiple traces of the execution of the three-point ladder on the board. For the sake of diversity, $pk_{k,0}^T$ (respectively $pk_{k,1}^T$) inputs are generated multiple times by choosing different points and curves. Then, two $t$-tests (see Section 5.4.2) are computed:

T0 between the traces obtained with a random triplet and the ones obtained with $pk_{k,0}^T$,
T1 between the traces obtained with a random triplet and the ones obtained with $pk_{k,1}^T$.

The $t$-test graph exhibiting peaks corresponds to the correct hypothesis. Comparing $t$-tests T0 and T1 eliminates the need for a threshold indicating that the values are significant.

### Results

In Figure 5.5 we show the results of an attack on a single bit. In particular, we attack the $12^{\text{th}}$ bit of the secret key, and the time window in the figures is limited to the first 20 iterations of the three-point ladder loop.

The trace corresponding to random inputs (not shown) is similar to the one obtained for the wrong hypothesis (Figure 5.1). The difference between the trace corresponding to the correct bit hypothesis and the wrong one is visible to the naked eye.

The attack is automated by performing a statistical test, as explained in Section 5.4.3. We use 10 raw traces from each collection (30 traces per secret bit) to compute T0 and T1. Do note that the collection obtained with random inputs can be reused for both T0 and T1, and for all bits. The $t$ values are shown on Figures 5.3 and 5.4.

This proof of concept shows that a bit of the secret key can be extracted by use of side-channel, even if the implementation was protected by coordinate randomisation.

### 5.4.4 Isogeny computation

The following experiment describes a proof of concept for the attack on the isogeny computation as described in Section 5.3 using power analysis.

#### Target operation

In the experiment, only the power consumption of the first $\mathbb{F}_p$ multiplication from the modular inversion involved in the computation of the $j$-invariant is measured. The main reason this choice was made is because the trace of a single multiplication was enough to extract the full secret key.

If the leakage of one field multiplication alone is not enough to correctly detect the presence of zero values, a single trace including the whole modular inversion can be segmented into multiple sub-power traces to boost the accuracy of the comparison.

#### Experiment procedure

The experiment followed the approach with the two baselines as described in Section 5.4.2. The targeted parameter set was SIKEp434 with a 434-bit prime $p = 2^{216}3^{137} - 1$. The target party has a key of size $\lfloor \log_2(3^{137}) \rfloor = 217$ bits and computes a $3^{137}$-isogeny. The malicious points are contained in the $2^e$ torsion, with $e = 216$. The break-point exponent $o$ is equal to 9 (see Table 5.6, page 74). We can extract bits of index $k$ for $0 \leq k \leq e - o$ (see Algorithm 9, page 74). Therefore the attack allows us to extract $216 - 9 + 1 = 208$ bits of the secret key.

The attack procedure is given in Algorithm 12.

**Algorithm 12:** Experimental attack on isogeny computation.

**do** capture the baselines $B^{(0)}$, $B^{(*)}$ for the two categories—zero and random:
  Send $pk^j_{(0)} \leftarrow$ `pk_j`$(0,0,o-1,0)$ to capture $B^{(0)}$.
  Send $pk^j_{(*)} \leftarrow$ `pk_j`$(0,0,o+1,0)$ to capture $B^{(*)}$.
**for** $k = 0$ **to** $e - o$ (starting with $sk_{<0} = 0$) **do**
  Send $pk^j_k \leftarrow$ `pk_j`$(k, sk_{<k}, o, 0)$ to capture $\mathrm{Tr}_i$.
  **if** $\rho(\mathrm{Tr}_i, B^{(*)}) > \rho(\mathrm{Tr}_i, B^{(0)})$ **then** $sk_{<k+1} = sk_{<k} + 2^k$. `// Section 5.4.2, page 82`
**return** $sk_{<e-o+1}$

**Results**

Across $N = 1,000$ experiments, the first 209 private-key bits were always successfully extracted through collision power analysis with baselines. Figure 5.8 shows power traces of the two baselines. Table 5.7 shows the average correlation coefficients when a target trace is compared against the two baselines.

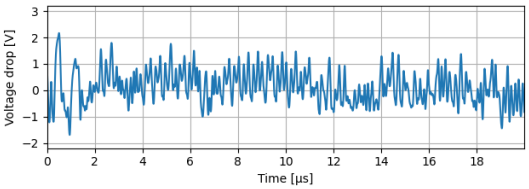



Figure 5.6: Zero-valued baseline $\mathrm{B}^{(0)}$.

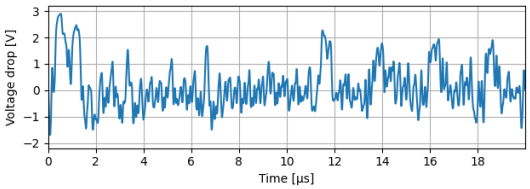



Figure 5.7: Random-valued baseline $\mathrm{B}^{(*)}$.

Figure 5.8: Examples of baseline traces corresponding to a single $\mathbb{F}_{p^2}$ multiplication processing the zero value in one case, and a random value in the other.

| Baseline | Target $j = 0$ | Target $j \neq 0$ |
|---|---|---|
| $j = 0$ | **0.9975** | 0.3915 |
| $j \neq 0$ | 0.3916 | **0.9909** |

Table 5.7: Average PCCs between baselines and target traces ($N = 1,000$).

Given the significant correlations for a single field multiplication, the results give strong evidence that the power analysis of one multiplication is enough to detect zeros. This proof of concept shows that it is possible to extract the (almost) full secret key, which was furthermore successfully confirmed across 1000 experiments.

## 5.5 Countermeasures

Our attacks rely on ciphertexts containing maliciously generated point triplets $(Q, P, Q-P)$ which are not the legitimate images of the public $\ell_I^{e_I}$-torsion basis under an isogeny of degree $\ell^e$. Validating SIKE ciphertexts is a problem that is believed to be as hard as breaking SIKE itself, thus we cannot hope to completely rule out side-channel attacks using malicious ciphertexts. Nevertheless our malicious ciphertexts deviate from the legitimate format in a detectable way, letting us design an effective countermeasure.

To counter the attack it is enough to check that $P$ and $Q$ are both of order $\ell_I^{e_I}$, and that they generate the $\ell_I^{e_I}$-torsion, i.e., that they are independent. The independence can be checked by showing that $[\ell_I^{e_I-1}]P \neq [\ell_I^{e_I-1}]Q$. Furthermore, if $\ell_I = 2$, we also check that $[\ell_I^{e_I-1}]P \neq [0:1]$. These checks can be done at the cost of two exponentiations. This test is called the *CLN test*, after the names of its first proponents [CLN16].

We added the countermeasure to Microsoft's PQCrypto-SIDH library [HPR07] and tested it on a laptop equipped with an Intel Coffee Lake i9-8950HK CPU 2.90GHz with Turbo Boost turned off. We benchmarked both the generic C version and the version with x64 assembly optimizations. The results are reported in Table 5.8 and show a performance hit of around 10%.

| | SIKEp434 | SIKEp503 | SIKEp610 | SIKEp751 |
|---|---|---|---|---|
| Decapsulation unprotected | 10,562,332 | 14,743,837 | 28,449,749 | 44,881,873 |
| Decapsulation with CLN | 11,801,827 | 16,296,830 | 32,218,683 | 50,351,641 |
| ratio | 1.117 | 1.105 | 1.132 | 1.121 |
| Decapsulation compressed | 11,306,190 | 15,569,411 | 29,537,047 | 46,913,973 |

Table 5.8: Performance in cycles of unprotected SIKE decapsulation vs SIKE decapsulation with CLN countermeasure.

### 5.5.1 Compressed SIKE

Compressed SIKE (see Section 2.5.2, page 26) offers protection from our attack by construction. In compressed SIKE, the target party reads only the coordinates of the public key points in $\mathbb{Z}/(\ell_I^{e_I})\oplus\mathbb{Z}/(\ell_I^{e_I})$, but they compute a basis of the $\ell_I^{e_I}$ torsion themself, so the points obtained must be of correct order. Furthermore the overhead in the decapsulation procedure is slightly smaller than the overhead of the CLN protected uncompressed SIKE (see Table 5.8). Key generation and encapsulation are still slower in compressed SIKE than in the CLN protected version.

### 5.5.2 Relation to previous work

In a work predating the first round of NIST's competition, Koziel, Azarderaksh, and Jao explored zero-value attacks on SIDH/SIKE protected with coordinate randomisation and the CLN test [KAJ17]. They presented four attacks and claimed that these manage to bypass the countermeasure. We shall now explain why none of them applies to SIKE.

The first attack ([KAJ17, §5]) targets the three-point ladder. It is based on the fact that, in the main loop of the ladder, one of the inputs to the differential addition is always either $P$ or $Q-P$, according to a secret bit. First, we note that the attack assumes the coordinates of $P$ and $Q-P$ are not randomised before entering the ladder, but in this case a simpler DPA would work as well. Second, since the introduction of a more efficient ladder in [FLOJRH18], all implementations of SIDH, including SIKE, have moved away from using a fixed pair of points in the loop, as can be seen in Algorithm 5.

The second attack ([KAJ17, §6]) also targets the three-point ladder, and asks to find "[Montgomery] curves with points $P_0 = (\pm 1, y)$ with a large order". However, such points always have order 4 ([CS17]), the attack can thus not be mounted against any version of SIDH implemented using Montgomery curves, such as SIKE.

The third attack ([KAJ17, §7.2]) proposes to make a guess on the secret, and then generate a ciphertext such that, if the guess is correct, the private isogeny computed during decapsulation will pass through the (supersingular) curve with $A = 0$. This is clearly feasible, and although the ciphertext generated would not pass the Fujisaki–Okamoto test, such an attack is hard to detect at an early phase. The only obstacle, already realised by [KAJ17], is that in all efficient implementations of SIDH/SIKE the curve $E_A$ is internally represented as $[A^+_{24} : C_{24}]$ or as $[A^+_{24} : A^-_{24}]$. Neither of these encodings can produce a zero if it represents an elliptic curve, thus it seems that this idea cannot be used in a realistic scenario.

The last attack ([KAJ17, §7.3]) tries to force zeros in points pushed through the isogeny computation and, in a sense, foreshadowed our attack presented in Section 5.3. The description in [KAJ17] is however incomplete and, by the authors' own admission, does not apply to Montgomery curves.

### 5.5.3 Blocking all zero-value attacks

The CLN countermeasure does not only block our attack, it actually blocks all zero-value attacks on SIKE. To be precise – the field element 0 is never computed during the execution of SIKE if the input public key has passed the CLN test. To prove this, we provide tables of all field computations that are executed during SIKE. We argue that the outputs of those computations are 0 only if the input elliptic curve points are of a certain order, and we show that that cannot happen if the input points pass the CLN test. All the tables are provided in Appendix F.

## 5.6 Conclusion

We described two types of zero-value attacks against SIKE: one on the three-point ladder, the other on the isogeny computation. Both attacks are based on special-point inputs that enable an adaptive bit-by-bit key recovery. The attacks were analysed in theory, and also verified experimentally on the recommended SIKE implementation for Cortex-M4 with both electromagnetic and power analysis using different techniques. At last, we argued that the Costello–Longa–Naehrig test which verifies the order of the points is sufficient to stop the attacks. Furthermore we showed that the CLN test blocks all zero-value attacks, i.e., that the field element 0 is never computed during the execution of SIKE when the public key passes the CLN test.

An interesting open question to consider is whether it is worth dropping uncompressed SIKE altogether and replacing it with compressed SIKE. This of course depends on the side-channel and performance trade-off that one is willing to make. A semi-compressed version of SIKE with uncompressed public keys and compressed ciphertexts would also be an interesting research direction to pursue.

# 6 Single-trace clustering power analysis of Cortex-M4 SIKE

In this chapter we present a side-channel clustering power analysis against the recommended implementation of SIKE for the ARM Cortex-M4 microcontroller. The attack targets the three-point ladder procedure, in particular the conditional swap sub-procedure which is directly dependent on a bit of the secret key. The attack is effective even if the implementation is protected by coordinate randomisation or the CLN test (see Section 5.5).

The attack is applicable in a fully ephemeral setting. In other words, a single execution of the key exchange is sufficient, and only samples from the power domain of the microcontroller are necessary. Neither electromagnetic sampling, nor profiling, nor previous knowledge of the target device is required and the key recovery is completely non-supervised, i.e., no template is ever built.

A clustering method based on thresholding the distribution of sorted power samples is shown to be sufficient to recover the full key. Finally, an effective countermeasure based on splitting the masking value into multiple random shares during the swapping procedure is shown to protect against the attack described in the paper, with a performance overhead which is less than 1% of the overall runtime of the algorithm.

## 6.1 Side-channel analysis

In this section we briefly recall the parts of SIKE which are targeted, we explain techniques used in the attack (clustering), and describe the attack which works by classifying the power samples of a single execution of the three-point ladder in SIKE.

### 6.1.1 Point of attack.

SIKE makes use of the three-point ladder to compute a secret point that is then used to generate the kernel of a party's secret isogeny. More specifically, the three-point ladder is an optimised computation of the elliptic curve operation of $P + [sk]Q$ for a scalar $sk$ of $n$ bits, by using only

the $X$ and $Z$ coordinates of the points $Q$, $P$, and $Q-P$ on a Montgomery curve. The three-point ladder was introduced in 5.

One of the main sub-procedures of the three-point ladder is a conditional swap function `cswap` executed once per each realisation of a loop step.

#### Conditional swap

To perform the conditional swapping of points, a function `cswap` (also shown in Appendix G, Listing G.1) takes `R` and `R2` as parameters, as well as a mask that expands the value of the private bit difference on a whole word. Given two consecutive private-key bits $sk_{i-1}, sk_i$ (for $0 \leq i < n$ with $sk_{-1} = sk_n = 0$), with 32-bit words, such a mask corresponds to:

$$\texttt{mask} = \begin{cases} \texttt{0x00000000} & \text{if } sk_{i-1} \oplus sk_i = 0, \\ \texttt{0xFFFFFFFF} & \text{if } sk_{i-1} \oplus sk_i = 1. \end{cases}$$

Then, each word of the two elliptic curve points (resp. $a$ and $b$) is processed according to this formula:

$$\begin{aligned} \texttt{tmp} &= \texttt{mask} \,\&\, (a \oplus b), \\ a &= \texttt{tmp} \oplus a, \\ b &= \texttt{tmp} \oplus b, \end{aligned}$$

where & corresponds to the bit-wise "and" operator. Such a procedure is known to be constant-time (see [CS18]).

### 6.1.2 Clustering

#### k-means.

The *k-means algorithm* [M+67] is an unsupervised clustering algorithm that partitions a population of $n$ samples into $k$ sets solely based on the values of the samples.

Informally, the algorithm starts with $k$ groups of means $\mu_j$ $(0 \leq j < k)$, and reassigns the samples to the group with the closest mean. As doing so may change the means of the groups, the process is repeated until convergence. The procedure is shown in pseudocode in Algorithm 13.

### 6.1.3 Attack procedure

The attack assumes a passive adversary able to monitor the power consumption of the target device. The power consumption of the entire three-point ladder is measured with a fixed sampling rate. The power samples are then segmented into multiple power traces synchronised at the beginning of each step of the three-point ladder. Moreover, only the segments corresponding to the execution of the `cswap` functions are considered, each of them ultimately consisting of $M$

**Algorithm 13:** THE $k$-MEANS ALGORITHM.

**Input:** Set $S = \{s_i \in \mathbb{R}\}$ of $n$ real values, $k$ number of clusters.
**Output:** A partition of $S$ into $k$ clusters.
**Procedure** `k-means(`$S, k$`)`

```
1  Assign each s_i to a cluster j at random for all i.
2  repeat
3      Compute μ_j as the mean of each cluster for all j.
4      Assign s_i to the cluster j = argmin|s_i − μ_j| for all i.
   until No μ_j changes value.
   return The final cluster assignment
```

samples (typically, a few thousands).

The `cswap` procedure uses a mask of value either `0x00000000` or `0xFFFFFFFF` depending on the difference between two consecutive secret bits. Due to the high Hamming distance between the possible mask values, the power consumption is directly correlated with the value of the mask, and thus the value of the corresponding secret bit.

The attack attempts to distinguish whether the swap occurred or not for each iteration of the loop by gathering the samples at a same location in all iterations and clustering them with the $k$-means algorithm.

Since a difference of bit s can only be zero (identical) or one (different), only two clusters are considered (i.e., $k = 2$). The private key can be entirely reconstructed from the labels at the end of the clustering.

**Attack procedure.** Given the $n$ segmented power traces $T_i$ of $M$ power samples each, the procedure consists of three steps:

1. Select a sample location $0 \leq t < M$ in the power traces.
2. Cluster with $k$-means the $n$ power samples at location $t$ throughout the traces $T_i$ (for $0 \leq i < n$), and reconstruct the key from the labels.
3. Verify the key obtained.

Algorithm 14 shows the second step of the attack in more details. The samples $s_i$ must all correspond to the samples at a same time throughout the $n$ segmented power traces. Since Algorithm 14 considers samples from a single timing location in the power traces, the procedure can be repeated with all different positions until a returned key (or its bitwise inverse) successfully decrypts a ciphertext. As a result, the overall attack has a complexity of precisely $M$ executions of $k$-means and key try-outs.

**Algorithm 14:** CLUSTERING POWER ANALYSIS.

**Input:** Set $S = \{s_i \in \mathbb{R}\}$ of $n$ power samples at time $t$.
**Assumes:** The power samples are aligned,
$s_i$ corresponds to $i$'th step of the three-point ladder loop.
**Output:** Secret key $sk$.
**Procedure** `cpa(`$S$`)`

```
1  S_0, S_1 = k-means(S, 2)
2  sk_{-1} = 0
3  for i = 0 to n − 1 do
4      if s_i ∈ S_0 then
5          swap = 0
6      if s_i ∈ S_1 then
7          swap = 1
8      sk_i = swap ⊕ sk_{i−1}
9  sk = Σ_{i≥0} sk_i 2^i
   return sk
```

## 6.2 Attack Enhancements

In a full attack as described in Section 6.1.3, the adversary needs to pass through all sample locations in the traces and use $k$-means to recover a key candidate that eventually needs to be verified. Adopting a better strategy for any of these steps can lead to both faster and more successful results.

This section lists enhancements to speed up the eventual recovery of the key. These can sometimes be combined to improve the overall attack.

### 6.2.1 Thresholding

Since the overall attack needs to run a clustering algorithm several times, an algorithm that clusters the power samples more efficiently leads to faster results.

While Algorithm 13 already involves low-complexity computations, the clustering algorithm does not needs to be generic and can therefore be tailored to a one-dimensional two-population problem by splitting the distributions with an appropriate middle point.

Many solutions exist to find a suitable middle point, such as computing the overall mean of all the samples, or finding the biggest gap between two neighboring power sample. Algorithm 15 proposes a clustering which calculates the literal middle point of the distribution by finding the maximum and minimum. Such a solution runs in $\mathcal{O}(n)$, but can be tweaked to present other advantages that are described in the next subsection.

**Algorithm 15:** THRESHOLDING CLUSTERING ALGORITHM.

**Input:** Set $S = \{s_i \in \mathbb{R}\}$ of $n$ power samples at time $t$.
**Assumes:** The power samples are aligned,
$s_i$ corresponds to $i$'th step of the three-point ladder loop.
**Output:** Secret key $sk$.
**Procedure** `tca(`$S$`)`

```
d = (min(S) + max(S))/2
sk_{-1} = 0
for i = 0 to n − 1 do
    if s_i ≤ d then
        swap = 0
    if s_i > d then
        swap = 1
    sk_i = swap ⊕ sk_{i−1}
sk = Σ_{i≥0} sk_i 2^i
return sk
```

### 6.2.2 Enhancing key verification

The attack achieves a better performance by reducing the number of key candidates to verify and by correcting plausible clustering mistakes.

#### Majority rule.

As noted by [PITM14], a same labelling re-occurring throughout many different locations is likely to be correct. Two majority rules are therefore proposed:

1. A *vertical* majority in which a candidate key occurring multiple times through the timing locations is verified in priority.
2. A *horizontal* majority in which individual key bits are labelled given their majority throughout the clusterings at all timing locations.

In a horizontal majority rule, a threshold can be selected to filter all the bits for which the clusterings give the same results, while the remaining bits can simply be guessed.

#### Educated thresholding.

In the clustering power analysis against the three-point ladder of SIKE, two observations can be made:

1. A clustering is successful only when the two sub-distributions are separated.
2. The number of swaps must always be even.

The first observation stems from the fact that two samples of identical value should always be assigned to the same cluster. Hence, a successful clustering can only be found by splitting the overall distribution in two in between two sample values.

The second observation is due to the fact that the three-point ladder requires the points to always be "un-swapped" at the end of the procedure. This means that the sizes of the two clusters are always even which can therefore be used to validate the key found.

As a result, one can design a thresholding algorithm similar to Algorithm 15 that first sorts the power samples and then separates the distribution in two, each call at a different threshold starting from a middle point. The threshold can move depending on the distance between the current threshold and the two cluster centres (similarly as in $k$-means). By iteratively calling such an algorithm, the labels that are more likely to be erroneous can be marked and subsequently flipped in a way that makes sure that the total number of swaps is always even.

## 6.3 Experimental results

This section reports a proof of concept for the clustering power analysis described in Section 6.1.3, in addition to an evaluation of the efficiency of the enhancements proposed in Section 6.2.

Details on the hardware setup, the equipment, and the specifications can be found in [GK22].

### 6.3.1 Target implementation

The software considered is the SIKE implementation for Cortex-M4 [SAJA20]. The implementation was modified in order to introduce the coordinate randomisation countermeasure, and to ease the power trace collection.

In particular, only the three-point ladder loop is executed, and the microchip is provided with a secret key, a valid public key triple $Q, P, Q-P$, and a randomness seed. The seed is used at each iteration of the loop to randomise the coordinates of the triple. The randomisation is carried out at the beginning of each loop iteration by generating three random non-zero $\mathbb{F}_{p^2}$ elements $r_{R_0}, r_P, r_{Q-P}$ and multiplying the $X$ and $Z$ coordinates of the points $R_0, R$ and $R_2$ correspondingly. In total, 6 multiplications in $F_{p^2}$ are performed:

$$[X_{R_0} : Z_{R_0}], [X_R : Z_R], [X_{R_2} : Z_{R_2}] \mapsto [X_{R_0} r_{R_0} : Z_{R_0} r_{R_0}], [X_R r_R : Z_R r_R], [X_{R_2} r_{R_2} : Z_{R_2} r_{R_2}]. \quad (6.1)$$

We decided to use ChaCha8 [Ber08] for pseudo-random number generation due to its efficiency and ease of implementation.

Though these modifications create an unrealistic attack scenario, the experiment is *still* practical on unmodified software but requires additional effort of marginal complexity. A reader interested in processing equivalent power traces from the acquisition of raw power samples from the

execution of unmodified software is advised to read [DPN+16].

The final software on which power traces were acquired can be found here: https://github.com/AymericGenet/SIKE-clusterswap-2021.

### 6.3.2 Traces collection

Power traces of the `cswap` executions were collected in the following manner:

(1) Send the Cortex-M4 a random key and randomness seed.
(2) Generate and send three valid public key points $Q$, $P$, and $Q - P$.
(3) Repeat the following $n$ times:
    (a) Make the Cortex-M4 execute the next loop iteration.
    (b) Save the power trace from the oscilloscope.

The above was repeated 1,000 times, each time with a different key and seed, for SIKEp434 (hence $n = 218$). An example of a power trace along with its frequency components for SIKEp434 is shown in Fig 6.1. Note that most of the frequency components are zero due to the limiting analog bandwidth of the oscilloscope (20 MHz).

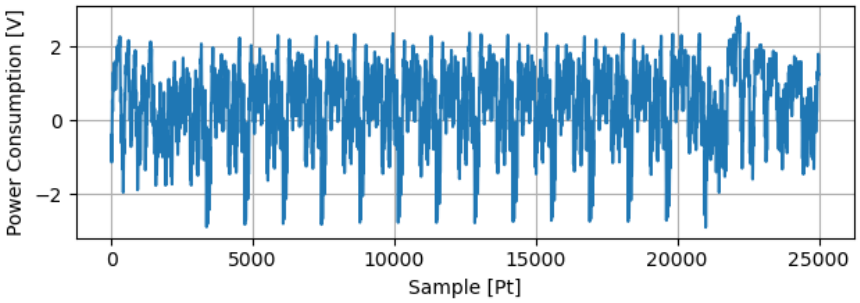



Figure 6.1: Example of one of the $n$ power traces corresponding to `cswap` in a single iteration of the loop.

### 6.3.3 Clustering power analysis

In the next step of the experiment, the $n$ collected traces of each experiment are exploited to attempt a key recovery as explained in Section 6.1.3 (cf. Algorithm 14).

(1) Process (for $0 \leq i < n$):
    (a) Let $T_i$ be a power trace of length $M$,
(2) Run the attack on $T_i$:
    (a) Go to the next sample location $0 \leq t < M$.
    (b) Run clustering on $S = \{T_i[t] \mid 0 \leq i < n\}$.
    (c) Record the $sk_t$ returned for time $t$.

The success rate is calculated through all the timing positions and frequencies over the 1,000 sets of measured traces by comparing the recovered key with the correct key.

### 6.3.4 Results

Out of the 1,000 experiments, the correct key is *always* found in the set of recovered keys $\{sk_t\}$. Table 6.2 reports various metrics about how often the correct key appears in the set of recovered keys. The independent success rates of each timing position is reported in Fig. 6.3. Finally, examples of sample distribution successfully clustered is shown in Fig. 6.4.

Figure 6.2: Statistics on the number of timing locations which yield the correct key across the $N$ = 1,000 experiments.

| $k$-means | | | | Thresholding | | | |
|---|---|---|---|---|---|---|---|
| min. | max. | $\mathbb{E}(\#t)$ | $\sigma(\#t)$ | min. | max. | $\mathbb{E}(\#t)$ | $\sigma(\#t)$ |
| 154 | 341 | 251.704 | 30.056 | 115 | 289 | 196.668 | 29.366 |

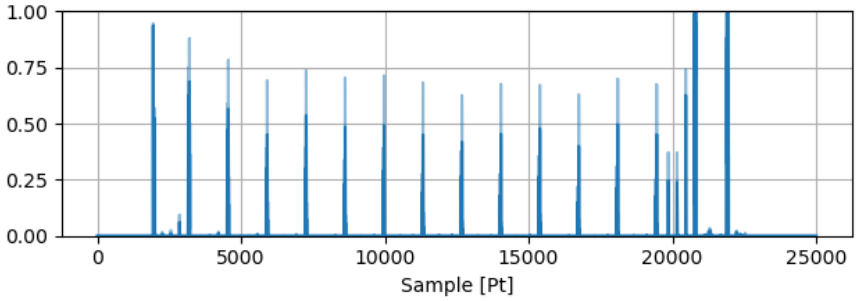



Figure 6.3: Success rate of the clustering power analysis (thresholding in opaque vs. $k$-means in transparent) at each timing locations.

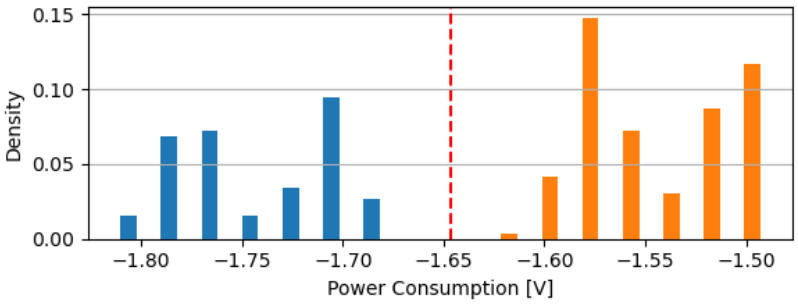



Figure 6.4: Example of a power sample distributions (at $t$ = 1943). The threshold (in red) was found by Algorithm 15.

### 6.3.5 Discussion

The above experiment proves that the recommended Cortex-M4 implementation of SIKE from [SAJA20] is vulnerable to low-effort power analyses, even in the case when the implementation is protected with coordinate randomisation. As a result, the main objective of the experiment is achieved.

The rest of the discussion focuses on the efficiency of the improvements.

**Thresholding efficiency.**

In addition to demonstrating the efficiency of the pre-processing phase, the experiments show that the thresholding proposed in Algorithm 15 is almost as successful as $k$-means. While $k$-means discovered the secret key in more timing locations than thresholding (cf. Fig. 6.3), our experiment with $k$-means took 29 hours to be performed, while the same analysis with thresholding took only 6.5 hours.

**Majority rule efficiency.**

The extremely high occurrence of the correct key in Table 6.2 confirms that the vertical majority rule explained in Section 6.2 helps validating the key. In all experiments, the most recorded candidate was always observed to be the correct key. As a result, the correct key is expected to be recovered within the first try-outs as the other candidates were all observed to be either random or close to the correct key.

### 6.3.6 Other SIKE instances

Note that the success of the clustering is closely connected to the relatively big number of samples available. As more samples are obtained, the distinction between the two clusters becomes easier. However, depending on the noise, additional samples may undermine the success of the overall clustering.

Still, similar results (if not better) have been obtained by running the same experiment with the bigger instances of SIKE. The experiments were executed with fewer runs, a fixed wavelet level, and only using the thresholding algorithm. The results are reported in Table 6.5 and prove that the attack is not limited to SIKEp434.

## 6.4 Countermeasure

Protecting the point-swapping procedure against clustering power analysis is not obvious, as the attack defeats classical countermeasures of [Cor99] which include coordinate randomisation, exponent randomisation, point blinding, and even the CLN countermeasure explained in Chapter

Figure 6.5: Statistics on the total number of timing locations which yield the correct key across the $N = 10$ experiments with the other instances of SIKE.

| Thresholding | | | | | |
|---|---|---|---|---|---|
| $p$ | $n$ | min. | max. | $\mathbb{E}(\#t)$ | $\sigma(\#t)$ |
| 434 | 217 | 115 | 289 | 196.7 | 29.4 |
| 503 | 252 | 154 | 232 | 184.2 | 22.9 |
| 610 | 304 | 335 | 419 | 396.4 | 23.6 |
| 751 | 378 | 271 | 321 | 297.5 | 15.8 |

5.5. Moreover, due to the recent study which relies on deep learning [PCBP21], even the tiniest bias in the power consumption may lead to a full recovery.

To make the task even more challenging, the target CPU of Cortex-M4 is known to be hard to protect (see [BCH+21]). As the Cortex-M4 appears to leak in the Hamming distance of the pipeline registers (see [CGD17]), the countermeasures need not only to consider the Hamming weight of the processed values, but also the Hamming distance between the values used by two consecutive instructions.

In this section, a countermeasure based on splitting the swapping mask is suggested.

### 6.4.1 Description

The proposed countermeasure revises the original swapping procedure from Section 6.1.1, page 90, in the following sense; instead of computing the value `mask` all at once, the idea is to split this quantity into two shares and add each share separately in a two-stage process to both $a$ and $b$. Such a procedure avoids computing values of extreme Hamming distances.

To this end, the swapping mask is replaced by two 32-bit masks: `m1` and `m2` such that their bitwise "xor" is equal to `mask`. In other words, given two consecutive private-key bits $sk_{i-1}$, and $sk_i$ (for $0 \le i < n$ with $sk_{-1} = sk_n = 0$):

$$\texttt{m1} \oplus \texttt{m2} = \begin{cases} \texttt{0x00000000} & \text{if } sk_{i-1} \oplus sk_i = 0, \\ \texttt{0xFFFFFFFF} & \text{if } sk_{i-1} \oplus sk_i = 1. \end{cases}$$

Given the two masks `m1` and `m2`, the new procedure works as follows:

$$\begin{aligned} \texttt{tmp1} &= \texttt{m1} \,\&\, (a \oplus b), \\ \texttt{tmp2} &= \texttt{m2} \,\&\, (a \oplus b), \\ a &= (\texttt{tmp1} \oplus a) \oplus \texttt{tmp2}, \\ b &= (\texttt{tmp1} \oplus b) \oplus \texttt{tmp2}. \end{aligned}$$

Because of the way we define $m1$ and $m2$, and because "&" distributes over "$\oplus$" as follows $(x \,\&\, m1) \oplus (x \,\&\, m2) = x \,\&\, (m1 \oplus m2)$, the above procedure swaps $a$ and $b$ in the same way as

the method in Section 6.1.1.

### 6.4.2 Implementation

The results from Section 6.3 provide insight on the critical points of the procedure that require particular care. Mainly three leaking points were identified:

1. The generation of the masks.
2. The instructions used to perform the swapping operation.
3. Exiting the function.

The third point can be avoided by incorporating the procedure to the code without calling a function, so only the first two points are addressed.

#### Masks generation

Let `swap` refer to the secret difference of private-key bits (i.e., $\texttt{swap} = sk_{i-1} \oplus sk_i$). The suggested countermeasure involves generating two random masks `m1` and `m2` that are either equal or bitwise complement depending on `swap`. To achieve this, given a random `m1`, the second mask `m2` is derived with the following formula: $\texttt{m2} = (1 - 2 \cdot \texttt{swap})(\texttt{m1} + \texttt{swap})$. This makes `m2` become the bitwise complement of `m1` through the representation of negative numbers in the CPU with the two's complement, i.e., $\neg\, \texttt{m1} = -(\texttt{m1} + 1)$.

**Performance.** Safely generating these two quantities requires sampling additional randomness. In particular, the multiplication of $(\texttt{m1} + \texttt{swap})$ by $(1 - 2 \cdot \texttt{swap})$ is computed as $u_1(\texttt{m1} + \texttt{swap}) - u_2(\texttt{m1} + \texttt{swap})$ where $u_1 - u_2 = 1 - 2 \cdot \texttt{swap}$. In total the mask generation requires at least 4 bytes of entropy (15 bits of which are effective) for generating `m1` and 4 bytes of entropy (14 bits of which are effective) for protecting the computation of `m2`. Such an implementation introduces an overhead of at least 12 additional instructions when compared to the original mask computation. The code is given in Listing G.3.

#### The swapping operation

Because of the Cortex-M4 leakage model, the order of the operations and the order of the operands play a critical role in the countermeasure. Particular care has to be taken with store and load instructions, as the power consumption of these procedures leaks sensitive values. As a result, given the two masks `m1` and `m2` generated as before, the implementation of the countermeasure must follow a special order given in Listing G.4.

**Performance.** As opposed to the original pattern of 8 instructions, such a solution requires 14 instructions per iteration which introduces further delays.

**Benchmarks**

We compare the run-time of our countermeasure against the run-time of the unprotected version of SIKE. About 62% of the overhead stems from acquiring randomness for mask generation. We generate a new pair of masks for each swap. In total there are $14 \times 4 = 56$ swaps, each of them requiring 8 bytes of entropy in order to generate the masks safely. For that reason we decided to use the Tiny Mersenne Twister pseudorandom number generator [SM11] seeded with a 64-bit value obtained from a hardware-based source of true randomness.

The protected `swap_points` is about 5.7 times slower than the unprotected swap. Within the three-point ladder function, the overhead adds up to about 70,000 additional cycles, which take up 5% of the total three-point ladder computing time. When considered as a part of a full execution of SIKE, the overhead due to protecting the swap boils down from about 1% in the key generation and decapsulation to 0.7% in the encapsulation procedure.

We generated the randomness needed for the masks in an ad-hoc manner during the execution of the three-point ladder. We believe that the overhead induced by the randomness generation can be further lowered by generating it during a pre-computation stage, or by using different algorithms.

Table 6.1: Run-times (in cycles) of the SIKEp434 implementation on an Intel Coffee Lake i9-8950HK CPU @ 2.90GHz with Turbo boost turned off.

| Operation | unprotected | protected |
|---|---:|---:|
| Mask generation | 1 | 251 |
| Swapping operation | 71 | 148 |
| Three-point ladder | 1,172,432 | 1,241,721 |
| Key generation | 6,083,645 | 6,153,241 |
| Encapsulation | 9,893,673 | 9,962,113 |
| Decapsulation | 10,625,881 | 10,747,176 |

### 6.4.3 Experimental validation

The proposed countermeasure was validated by conducting Welch's $t$-test [SM16]. Such a test gives a degree of confidence that two classes of power samples are statistically indistinguishable. In the present case, the two classes respectively correspond to whether the points were swapped during the collection of the power traces, or not.

The $t$ values are computed with the following formula:

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\sigma_0^2/n_0 + \sigma_1^2/n_1}}$$

where $\mu_0$, $\mu_1$ correspond to the means of the two classes, $\sigma_0^2$, $\sigma_1^2$ to their variances, and $n_0$, $n_1$ to

their cardinality (here, $n_0, n_1 \approx 1000$). A threshold of 4.5 for the $t$ values is set to reject the null hypothesis (see [DZD+17]). In other words, a $t$ value greater than the threshold gives evidence that the two distributions are not indistinguishable.

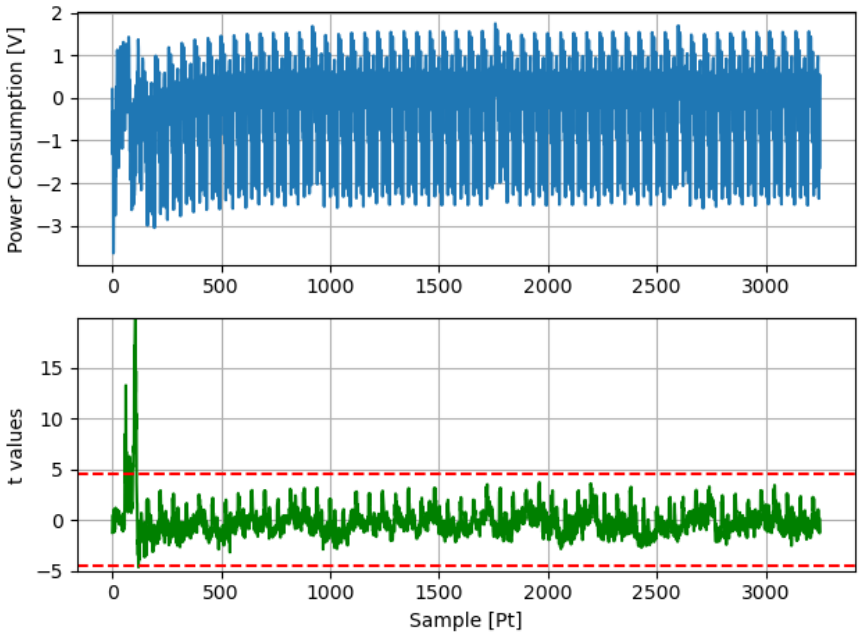



Figure 6.6: $t$-test of the countermeasure both in timing and frequency. The horizontal lines in red show the threshold above which the null hypothesis is rejected.

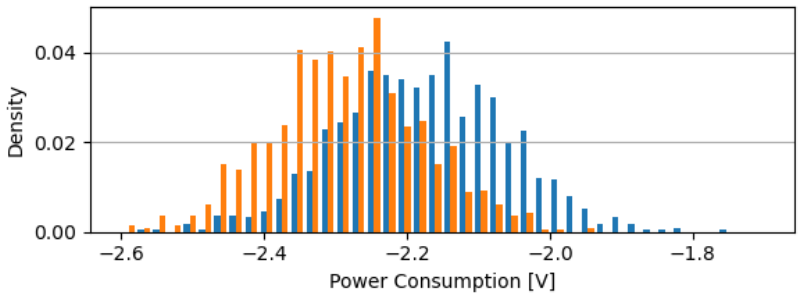



Figure 6.7: Power sample distributions at the locations which produced the highest value in both $t$-tests.

The results are shown in Fig. 6.6. Even though significantly large $t$ values appear in the plots, the attack is still unsuccessful when re-run against the countermeasure as the histograms corresponding to the power samples from the two classes overlap with each other at all points in time and frequency. Fig. 6.7 illustrates this by showing the histograms at the highest peaks of the $t$-test plots from Fig. 6.6. As one can notice, both histograms exhibit a significant variance, which prevents the attack to fully recover the private key. The histograms at all other points showed a similar overlapping.

While such a discrepancy of distributions prevents a successful clustering with the techniques described in this paper, more sophisticated attacks (such as [PITM14, PCBP21]) may still prevail. These attacks may therefore require additional efforts to withstand.

## 6.5 Conclusion

This chapter described a plain clustering power analysis able to recover the entire private key in a single execution of the three-point ladder in the implementation of SIKE for Cortex-M4.

While the attack has been experimentally shown to be always successful, the reader must keep in mind that the experiment was performed on a chip that is particularly vulnerable to power analysis. However, the countermeasure described completely thwarts the attack even on such a vulnerable chip. If the countermeasure is safe under such defenceless circumstances, then the implementation can be assumed to be safe in a more realistic scenario.

**Future work.** The experiment could be repeated with electromagnetic radiations of the microcontroller. Also, the countermeasure requires to be evaluated against other side-channel attacks and improved both in performance and security. Speeding up the countermeasure is also a matter of interest. Generating randomness during a pre-computation stage for both coordinate randomisation, and swap mask generating is a good direction to look into, and can bring a notable improvement in reducing the overhead from the countermeasure. Finally, an evaluation of other sensitive operations in SIKE—such as the isogeny computation—can be conducted, as there are still many other points that have not been evaluated yet that may also leak secret information through power consumption.

# Appendix Part IV

# A The Legendre pseudorandom function

## A.1 Computing the stabiliser $\mathrm{Stab}(f)$ of $f$

Let $m \in \mathrm{Stab}(f)$ be a matrix of order $r'$. Following the same argumentation from Section (3.2.2) there exists a change of coordinate matrix $P$ such that $D = P^{-1}mP$ is a diagonal matrix. We give a set of representatives for matrices $D$ and $P$ such that for each $m$ there is a single pair $D, P$ in that set satisfying

$$m = P\,D\,P^{-1}.$$

This property can be used to argue that we need only to find one $m_r$ of order $p_r$ for any prime divisor $p_r \mid r'$. Given $m_r$, an element $m_i$ of order $p_r^i$ is simply $PD^{1/p_r^{i-1}}P^{-1}$, and an element $m_q$ of order $q_r$ for some other divisor $q_r \mid r'$ is $PD_qP^{-1}$ for the corresponding matrix $D_q$ of order $q_r$. Furthermore, an element of order $p_r q_r$ can be found by computing $m_r^u m_q^v$ with $up_r + vq_r = 1$. Therefore in order to find the full stabiliser group we need only to find one element of prime order. This is done by searching for elements of order $q$ in the stabiliser, for each prime $q \mid r$, so we may start by assuming that we know $r'$.

The search for $m$ is done by going through the conjugacy class of a matrix $D$ of order $r'$, until we find a matrix that stabilises $f$. The conjugacy class has size $\Theta(p^2)$ so we expect to find $m$ in $\Theta(p^2)$ steps, but we have to be careful and go through the whole class without repetitions.

The process is explained separately for rational and irrational matrices.

### A.1.1 Rational matrices

If $m$ is of rational then for some $P \in \mathrm{GL}_2(\mathbb{F}_p)$

$$m = P \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} P^{-1}$$

where $a, b \in \mathbb{F}_p$ non-zero such that $\xi := a/b$ has order $r'$. Since $m$ is defined up to scalar multiplication in $\mathbb{F}_p^*$, we may suppose that $a = \xi$ and $b = 1$, so $D = \left(\begin{smallmatrix} \xi & 0 \\ 0 & 1 \end{smallmatrix}\right)$ for some $\xi$ primitive $r'$'th root of unity in $\mathbb{F}_p$. There are in total $\varphi(r')$ different $\xi$ values to consider, however each one will give rise to a different generator of the stabiliser of $f$, so the choice of $\xi$ does not matter.

The search for $m$ is done by enumerating $PDP^{-1}$, where matrices $P$ are chosen from $\mathrm{GL}_2(\mathbb{F}_p)$ up to right multiplication by an element of $Z(D) = \{\left(\begin{smallmatrix} a & 0 \\ 0 & b \end{smallmatrix}\right) | \ ab \neq 0\}$, the centraliser of $D$. In total there are $p^2 + p$ elements in $\mathrm{GL}_2(\mathbb{F}_p)/Z(D)$. One set of representatives can be chosen to be

$$\{\left(\begin{smallmatrix} 0 & 1 \\ 1 & d \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 0 \\ c & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 1 \\ c & d \end{smallmatrix}\right) | \ c, d \in \mathbb{F}_p \text{ such that the determinants are non-zero}\}.$$

When $r' = 2$, so $D = \left(\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, the set of representatives is halved because $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right) \in Z(D)$ after projecting on $\mathrm{PGL}_2(\mathbb{F}_p)$. In that case we give the following $(p^2 + p)/2$ representatives for the matrices $P$ :

$$\{\left(\begin{smallmatrix} 0 & 1 \\ 1 & d \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 1 \\ c & d \end{smallmatrix}\right) | \ c < d \in \mathbb{F}_p\}$$

where the ordering of elements of $\mathbb{F}_p$ is induced from the lift to $\{0, 1, \ldots, p-1\}$.

### A.1.2 Irrational matrices

If $m$ is of irrational then for some $P \in \mathrm{GL}_2(\mathbb{F}_{p^2})$

$$m = P \begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix} P^{-1}$$

where $\lambda, \mu \in \mathbb{F}_{p^2}$ are conjugate roots of an irreducible second degree polynomial such that $\xi := \lambda/\mu$ is a primitive $r'$'th root of unity.

**Lemma A.1.1.** The diagonal matrix $D$ defined above is unique in $\mathrm{GL}_2(\mathbb{F}_{p^2})/\mathbb{F}_p^*$.

*Proof.* Since $\xi = \overline{\mu}/\mu = \mu^{p-1}$ we have $\xi^{p+1} = 1$. Due to the primitivity of $\xi$ it follows that $r' \mid p + 1$.

If $\xi \in \mathbb{F}_p$ then $\xi^2 = 1$ so $\xi = -1$ and $r' = 2$. In that case $\lambda = -\mu$, so the minimal polynomial of $\lambda$ is $x^2 - c$ for some non-square $c$. Up to multiplying $D$ by a constant in $\mathbb{F}_p^*$, we may suppose $\lambda = \sqrt{u}$ for a fixed non-square $u$, and therefore there is only one such matrix.

If $\xi$ is not rational, then $\overline{\xi} = \xi^p = 1/\xi$, so $\xi\overline{\xi} = 1$. From $\lambda = \xi\mu$ we have $D = \left(\begin{smallmatrix} \xi\mu & 0 \\ 0 & \mu \end{smallmatrix}\right)$. The determinant and the trace of $D$ are the same as those of $m$, so in particular they are rational. This means that

$$\mu(\xi + 1) \in \mathbb{F}_p$$
$$\xi\mu^2 \in \mathbb{F}_p$$

from which it follows that $\mu = \frac{a}{\xi+1}$ and $\lambda = \frac{\xi a}{\xi+1}$ for some $a \in \mathbb{F}_p$. For any choice of $a$, the second condition follows from $\xi\overline{\xi} = 1$. Multiplying $\lambda$ and $\mu$ by any non-zero rational constant does not

change the property of $D$ being conjugate to $m \in \mathrm{PGL}_2(\mathbb{F}_p)$, to them being irrational conjugates of each other or to their quotient being equal to $\xi$. Therefore we may suppose $\lambda = \frac{\xi}{\xi+1}$ and $\mu = \frac{1}{\xi+1}$. □

We start by computing a primitive root of unity $\xi$ of order $r'$, and set $D$ as above. As before, the choice of $\xi$ does not matter.

The search for $m$ follows by going through $P\,D\,P^{-1}$ where the matrices $P$ are chosen such that $PDP^{-1}$ is rational and up to right multiplication by $Z(D)$, the centraliser of $D$.

**Rational** $PDP^{-1}$

If $PDP^{-1}$ is rational we have $PDP^{-1} = \overline{P}\,\overline{D}\,\overline{P}^{-1}$, so

$$(P^{-1}\overline{P})\left(\begin{smallmatrix}\mu & 0\\ 0 & \lambda\end{smallmatrix}\right) = \left(\begin{smallmatrix}\lambda & 0\\ 0 & \mu\end{smallmatrix}\right)(P^{-1}\overline{P}).$$

Define $A_P := P^{-1}\overline{P}$. The matrix $A_P$ satisfies $A_P^{-1} = \overline{A_P}$, so it has to satisfy

$$A_P = \begin{pmatrix} 0 & \alpha \\ 1/\overline{\alpha} & 0 \end{pmatrix}$$

for some non-zero $\alpha$ in $\mathbb{F}_{p^2}$. From $\overline{P} = PA_P$ we have some constraints on $P$,

$$P \in \left\{ \begin{pmatrix} q & \overline{q\alpha} \\ r & \overline{r\alpha} \end{pmatrix} \,\middle|\, q, r \in \mathbb{F}_{p^2},\ qr \neq 0,\ q^{p-1} \neq r^{p-1} \right\}.$$

**The centraliser** $Z(D)$

The matrix $D$ is diagonal with different eigenvalues, so

$$Z(D) = \left\{ \begin{pmatrix} x & 0 \\ 0 & \overline{y} \end{pmatrix} \,\middle|\, x, y \in \mathbb{F}_{p^2},\ xy \neq 0 \right\}.$$

Multiplying a $P$ on the right by an element of the centraliser gives

$$\begin{pmatrix} q & \overline{q}\,\overline{\alpha} \\ r & \overline{r}\,\overline{\alpha} \end{pmatrix}\begin{pmatrix} x & 0 \\ 0 & \overline{y} \end{pmatrix} = \begin{pmatrix} qx & \overline{q}\,\overline{\alpha}\,\overline{y} \\ rx & \overline{r}\,\overline{\alpha}\,\overline{y} \end{pmatrix} = \begin{pmatrix} qx & \overline{qx}\,\overline{(\frac{\alpha y}{x})} \\ rx & \overline{rx}\,\overline{(\frac{\alpha y}{x})} \end{pmatrix},$$

which sends $(q, r)$ to $(qx, rx)$ and $\alpha$ to $\alpha\frac{y}{x}$, so we may assume that $q = \alpha = 1$. A set of $p^2 - p$ representatives for matrices $P$ is

$$\left\{ \begin{pmatrix} 1 & 1 \\ r & \overline{r} \end{pmatrix} \,\middle|\, r \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p \right\}.$$

When $r' = 2$, so $D = \left(\begin{smallmatrix} \sqrt{u} & 0 \\ 0 & -\sqrt{u} \end{smallmatrix}\right)$ for some rational non-square $u$, the set of representatives is halved because $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right) \in Z(D)$ after projecting on $\mathrm{GL}_2(\mathbb{F}_p^2)/\mathbb{F}_p^*$. In that case we give the following $(p^2-p)/2$ representatives for matrices $P$:

$$\left\{ \begin{pmatrix} 1 & 1 \\ r & \overline{r} \end{pmatrix} \middle| \, r = a\sqrt{u} + b\,, \quad 1 \le a \le \tfrac{p-1}{2},\ 0 \le b < p \right\}.$$

# B SIKE

Figure B.1: The SIKE protocol.

| Alice | | Bob |
|---|---|---|
| | | $sk_B \leftarrow_{\$} [0, \ell_B^{e_B})$ |
| | | $s \leftarrow_{\$} \{0,1\}^\tau$ |
| | | $R_B \leftarrow P_B + [sk_B]Q_B$ |
| | | Let $\phi_B : E_0 \to E_B$ be s.t. $\ker(\phi_B) = \langle R_B \rangle$ |
| | $\xleftarrow{pk_B}$ | $pk_B = (E_B, \phi_B(P_A), \phi_B(Q_A))$ |
| $m \leftarrow_{\$} \{0,1\}^\tau$ | | |
| $sk_A \leftarrow G(m \Vert pk_B) \bmod \ell_A^{e_A}$ | | |
| $R_A \leftarrow P_A + [sk_A]Q_A$ | | |
| Let $\phi_A : E_0 \to E_A$ be s.t. $\ker(\phi_A) = \langle R'_A \rangle$ | | |
| $R'_A \leftarrow \phi_B(P_A) + [sk_A]\phi_B(Q_A)$ | | |
| Let $\phi'_A : E_B \to E_{AB}$ be s.t. $\ker(\phi'_A) = \langle R_A \rangle$ | | |
| $c_0 = (E_A, \phi_A(P_B), \phi_A(Q_B))$ | | |
| $c_1 = F(j(E_{AB})) \oplus m$ | | |
| $K = H(m \Vert c_0 \Vert c_1)$ | | |
| | $\xrightarrow{(c_0 \Vert c_1)}$ | $R'_B \leftarrow \phi_A(P_B) + [sk_B]\phi_A(Q_B)$ |
| | | Let $\phi_B : E_B \to E'_{AB}$ be s.t. $\ker(\phi_B) = \langle R'_B \rangle$ |
| | | $m' \leftarrow F(j(E'_{AB})) \oplus c_1$ |
| | | $sk'_A \leftarrow G(m' \Vert pk_B) \bmod \ell_A^{e_A}$ |
| | | $R' \leftarrow P_A + [sk'_A]Q_A$ |
| | | Let $\phi' : E_0 \to E'_A$ be s.t. $\ker(\phi') = \langle R' \rangle$ |
| | | **if** $(E'_A, \phi'(P_B), \phi'(Q_B)) = c_0$ |
| | | $K = H(m' \Vert c_0 \Vert c_1)$ |
| | | **else** |
| | | $K = H(s \Vert c_0 \Vert c_1)$ |

# C SIKE code

We include the code of the `xDBLADD`, `fp2mul_mont`, `fp2sqr_mont` and `mp_addfast` functions from [SAJA20]. Minor changes, such as variable naming, have been made to the code in order to adapt it to the names used in this paper. The lines of code `3,6,7,8,9,10,11,15,16,17,18,19` and the `mp_addfast` (highlighted in red) correspond to the targeted instructions.

```
void xDBLADD(point_proj_t Q, point_proj_t P, point_proj_t QP, const
   f2elm_t A24)
{  // Simultaneous doubling and differential addition.
   // Input: projective Montgomery points Q=(Q->X:Q->Z), P=(P->X:P
  ->Z),   Q-P=(QP->X:QP->Z), and Montgomery curve constant A24=(A+2)
  /4.
   // Output: projective Montgomery points Q <- 2*Q, and P <- Q+P.

   f2elm_t t0, t1, t2;

1    fp2add(Q->X, Q->Z, t0);
2    fp2sub(Q->X, Q->Z, t1);
3    fp2sqr_mont(t0, Q->X);
4    fp2sub(P->X, P->Z, t2);
4.5  fp2correction(t2);
5    fp2add(P->X, P->Z, P->X);
6    fp2mul_mont(t0, t2, t0);
7    fp2sqr_mont(t1, Q->Z);
8    fp2mul_mont(t1, P->X, t1);
9    fp2sub(Q->X, Q->Z, t2);
10   fp2mul_mont(Q->X, Q->Z, Q->X);
11   fp2mul_mont(t2, A24, P->X);
12   fp2sub(t0, t1, P->Z);
13   fp2add(P->X, Q->Z, Q->Z);
14   fp2add(t0, t1, P->X);
15   fp2mul_mont(Q->Z, t2, Q->Z);
16   fp2sqr_mont(P->Z, P->Z);
17   fp2sqr_mont(P->X, P->X);
18   fp2mul_mont(P->Z, QP->X, P->Z);
19   fp2mul_mont(P->X, QP->Z, P->X);
   //In practice 19 is called outside of xDBLADD
}
```

Figure C.1: xDBLADD from [SAJA20].

```
void fp2mul_mont(const f2elm_t a, const f2elm_t b, f2elm_t c)
{   // GF(p^2) multiplication.
    // Inputs: a = a0+a1*i and b = b0+b1*i.
    // Output: c = c0+c1*i.
    felm_t t1, t2;
    dfelm_t tt1, tt2, tt3;
    digit_t mask;
    unsigned int i;

1   mp_addfast(a[0], a[1], t1);
2   mp_addfast(b[0], b[1], t2);

3   fpmul_mont(a[0], b[0], c[0]);
4   fpmul_mont(a[1], b[1], tt2);
5   fpmul_mont(t1, t2, c[1]);

6   fpsub(c[1],c[0],c[1]);
7   fpsub(c[1],tt2,c[1]);

8   fpsub(c[0],tt2,c[0]);
}
```

Figure C.2: `fp2mul_mont` from [SAJA20].

```
void fp2sqr_mont(const f2elm_t a, f2elm_t c)
{   // GF(p^2) squaring.
    // Inputs: a = a0+a1*i.
    // Output: c = c0+c1*i.
    felm_t t1, t2, t3;

1   mp_addfast(a[0], a[1], t1);
2   fpsub(a[0], a[1], t2);
3   mp_addfast(a[0], a[0], t3);
4   fpmul_mont(t1, t2, c[0]);
5   fpmul_mont(t3, a[1], c[1]);
}
```

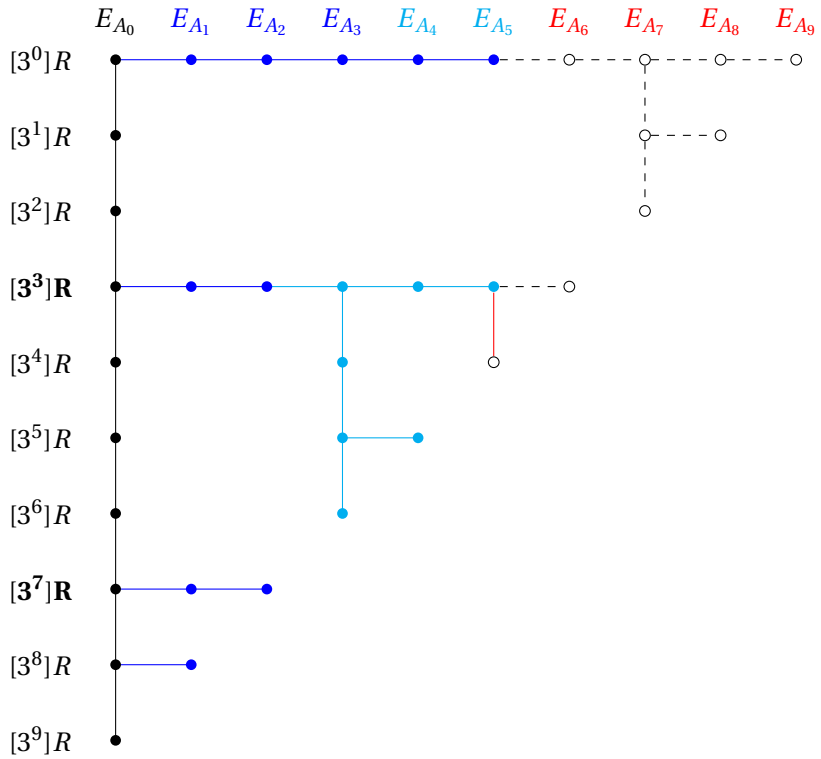
Figure C.3: `fp2sqr_mont` from [SAJA20].

```
void __attribute__ ((noinline, naked))  mp_addfast(const digit_t* a,
   const digit_t* b, digit_t* c)
{   // Multiprecision addition, c = a+b.
    asm(
1         "push  {r4-r9,lr}             \n\t"
2         "mov r14, r2                  \n\t"

3         "ldmia r0!, {r2-r5}           \n\t"
4         "ldmia r1!, {r6-r9}           \n\t"
5         "adds r2, r2, r6              \n\t"
6         "adcs r3, r3, r7              \n\t"
7         "adcs r4, r4, r8              \n\t"
8         "adcs r5, r5, r9              \n\t"
9         "stmia r14!, {r2-r5}          \n\t"

10        "ldmia r0!, {r2-r5}           \n\t"
11        "ldmia r1!, {r6-r9}           \n\t"
12        "adcs r2, r2, r6              \n\t"
13        "adcs r3, r3, r7              \n\t"
14        "adcs r4, r4, r8              \n\t"
15        "adcs r5, r5, r9              \n\t"
16        "stmia r14!, {r2-r5}          \n\t"

17        "ldmia r0!, {r2-r5}           \n\t"
18        "ldmia r1!, {r6-r9}           \n\t"
19        "adcs r2, r2, r6              \n\t"
20        "adcs r3, r3, r7              \n\t"
21        "adcs r4, r4, r8              \n\t"
22        "adcs r5, r5, r9              \n\t"
23        "stmia r14!, {r2-r5}          \n\t"

20        "ldmia r0!, {r2-r3}           \n\t"
21        "ldmia r1!, {r6-r7}           \n\t"
22        "adcs r2, r2, r6              \n\t"
23        "adcs r3, r3, r7              \n\t"
24        "stmia r14!, {r2-r3}          \n\t"

26        "pop  {r4-r9,pc}              \n\t"
    :
    :
    :
    );
}
```

Figure C.4: `mp_addfast` from [SAJA20].

# D A visual explanation of the isogeny attack

Figure D.1: An example of an isogeny computation with a kernel of wrong order




In Figure D.1, we can see a $3^{10}$-isogeny computation with a kernel of incompatible order.

- With black we denote regular points and supersingular elliptic curves.
- With blue we denote arbitrary points, isogenies with bad kernel, and arbitrary (non-supersingular) elliptic curves.
- With cyan we denote the point $\mathcal{O}$, isogenies with kernel $\langle\mathcal{O}\rangle$, triplings of $\mathcal{O}$ and degenerate

elliptic curves.

- With red we denote the isogeny (tripling) which first creates the undefined point $[0:0]$, and undefined elliptic curves.
- With circles we denote undefined points $[0:0]$.
- With dashed lines we denote isogenies which send points to the undefined point and undefined elliptic curves.

Assume that the points $[3^3]R$ and $[3^7]R$ are equal. On the first curve $E_{A_0}$ the point $R$ is tripled 9 times, and the points $[3^0]R, [3^3]R, [3^7]R$ and $[3^8]R$ are saved. A 3-isogeny is computed from $[3^9]R$, and the saved points are pushed. The images of $[3^3]R$ and $[3^7]R$ are still equal. Another 3-isogeny is computed from the image of $[3^8]$ and the saved points are pushed. The images of $[3^3]R$ and $[3^7]R$ are still equal. The third 3-isogeny sends the image of $[3^3]R$ to $\mathcal{O}$. Triplings of $\mathcal{O}$ are equal to $\mathcal{O}$. The next isogeny, generated by $\mathcal{O}$ sends the saved $\mathcal{O}$ points to $\mathcal{O}$, and the next curve is the degenerate curve. The next isogeny is also generated by $\langle\mathcal{O}\rangle$, sends $\mathcal{O}$ to $\mathcal{O}$, and the next curve is the degenerate curve. The first next tripling is the tripling of $\mathcal{O}$ on the degenerate curve which outputs the undefined point $[0:0]$. From this point onward all the outputs are $[0:0]$.

# E Practical isogeny computation with kernel points of bad order

In this appendix we provide visual representations of tree traversals associated to isogeny computations with kernel generating points of bad order. The tree traversal graphs are provided for all SIKE parameter sets. These are all the tree traversals that can occur during an execution of the attack from Chapter 5.

We use the same colour palette as in Appendix D, for both degree $2^{e_2}$ and $3^{e_3}$ isogenies.

- With black we denote regular points and supersingular elliptic curves.
- With blue we denote arbitrary points, isogenies with bad kernel, and arbitrary (non-supersingular) elliptic curves.
- With cyan we denote the point $\mathcal{O}$ (resp. $T$), isogenies with kernel $\langle\mathcal{O}\rangle$ (resp. $\langle T\rangle$), triplings of $\mathcal{O}$ (resp. $T$) and degenerate elliptic curves.
- With red we denote the isogeny or tripling which first creates the undefined point $[0:0]$, and undefined elliptic curves.
- With circles we denote undefined points $[0:0]$.
- With dashed lines we denote isogenies which send points to the undefined point and undefined elliptic curves.
- On the starting curve, we draw a symbol on the left of saved points which are equal, or that differ by $T$ (i.e. the points have *flipped* coordinates); equal points have the same symbol; points which differ by $T$ have the same symbol with a tilde above one of the symbols; points whose difference is not in $\{\mathcal{O}, T\}$ either have different symbols, or one of them doesn't have a symbol next to it.

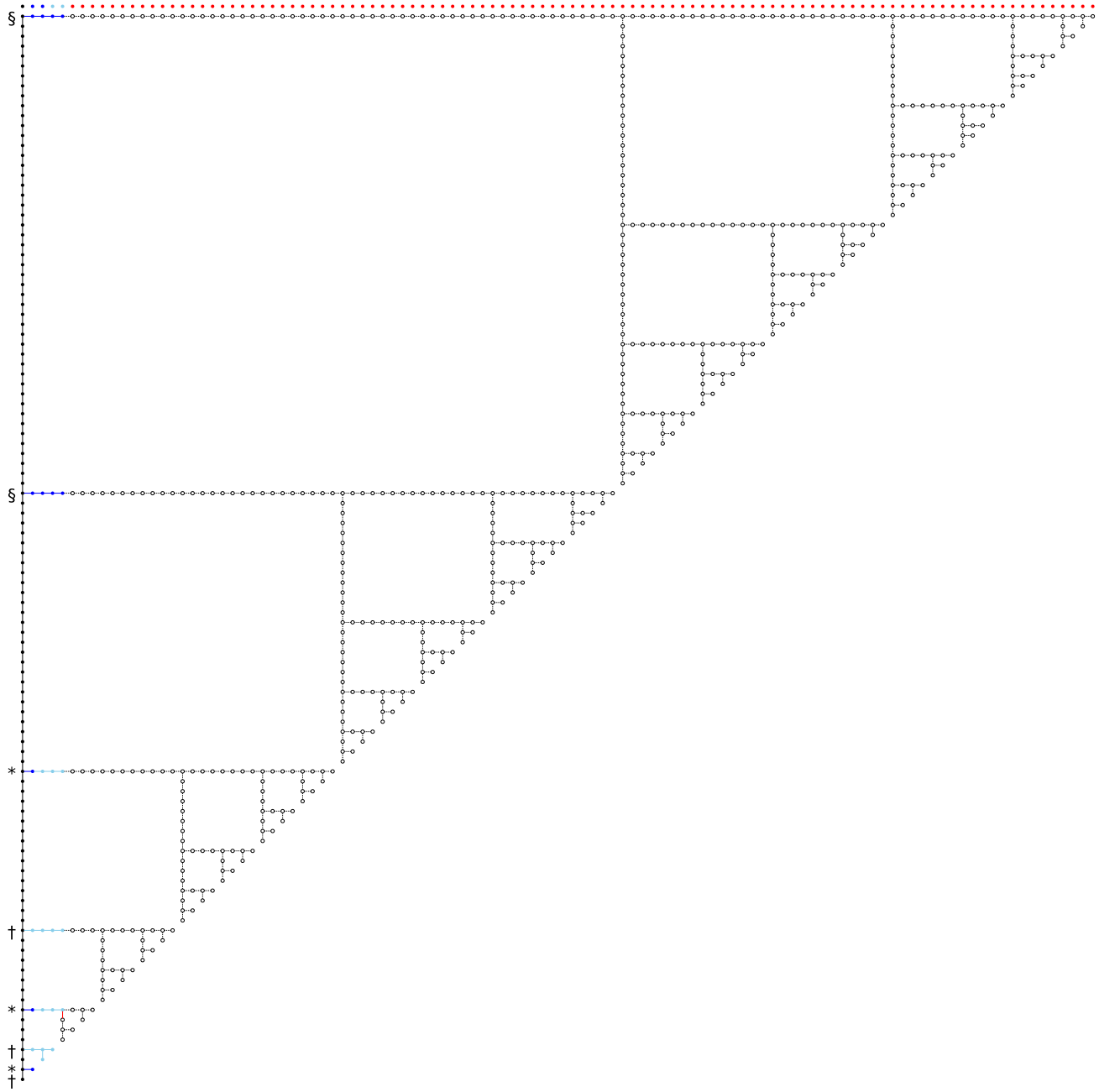



Figure E.1: Even degree isogeny with kernel of bad order; parameter set $p434$
Isogeny degree: $2^{216}$
Breaking point: $o = 3$
Kernel point degree: $3^{o-1}$

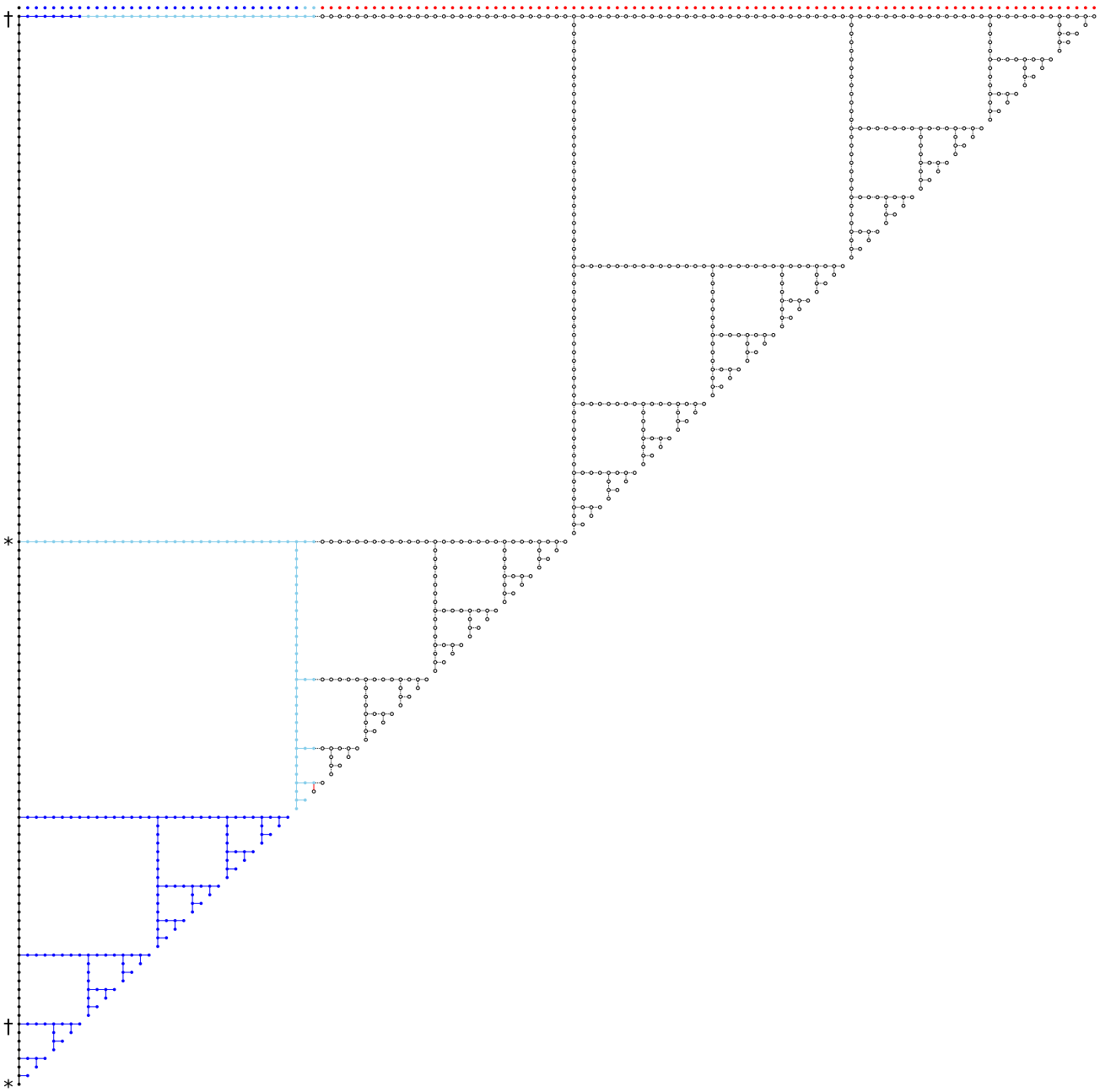


Figure E.2: Even degree isogeny with kernel of bad order; parameter set $p503$
Isogeny degree: $2^{250}$
Breaking point: $o = 4$
Kernel point degree: $3^{o-1}$

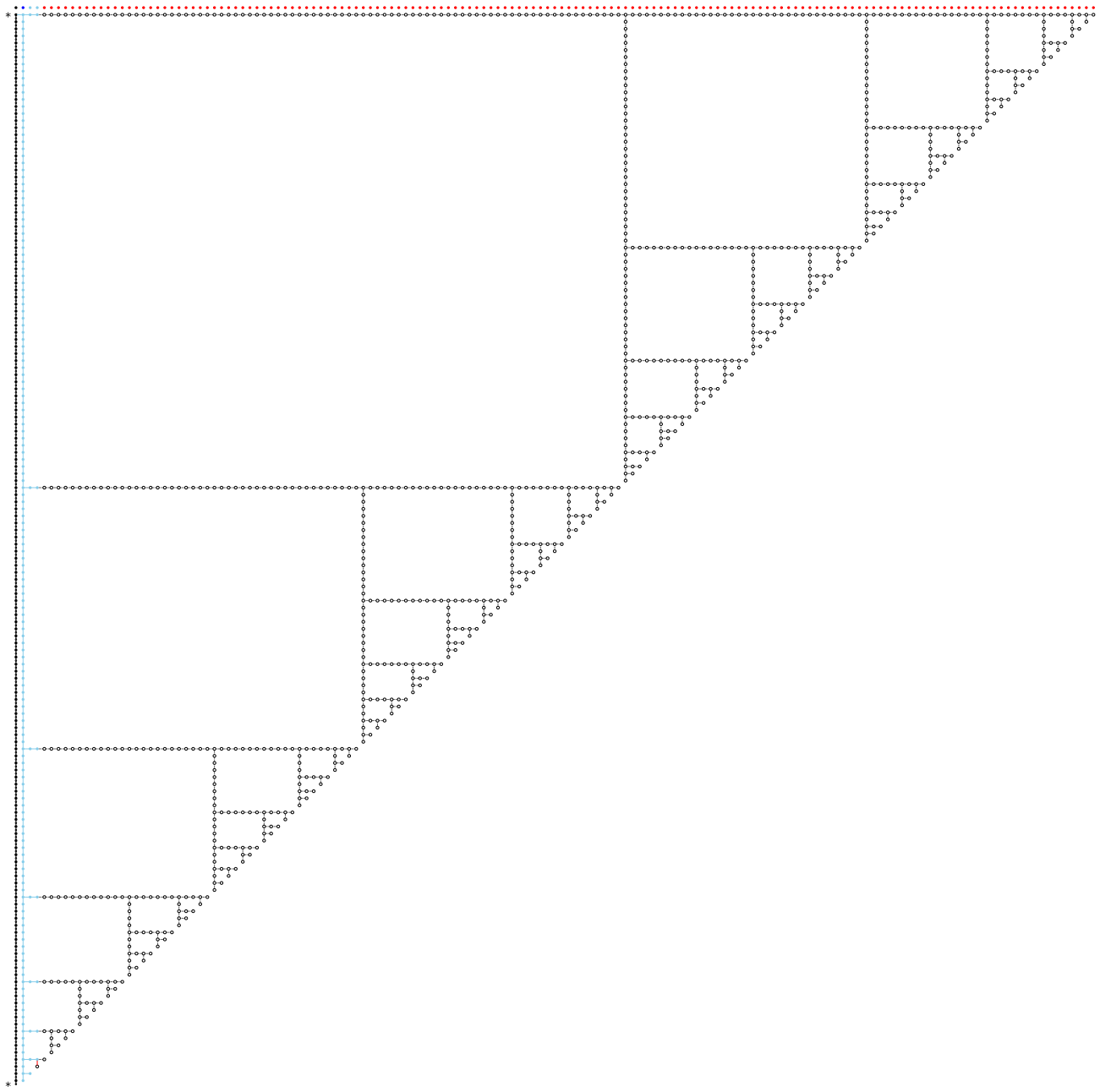


Figure E.3: Even degree isogeny with kernel of bad order; parameter set $p610$
Isogeny degree: $2^{305}$
Breaking point: $o = 2$
Kernel point degree: $3^{o-1}$
Note: On the first curve points are connected by doublings, and the first two curves by a 2-isogeny. Other isogenies are of degree 4.

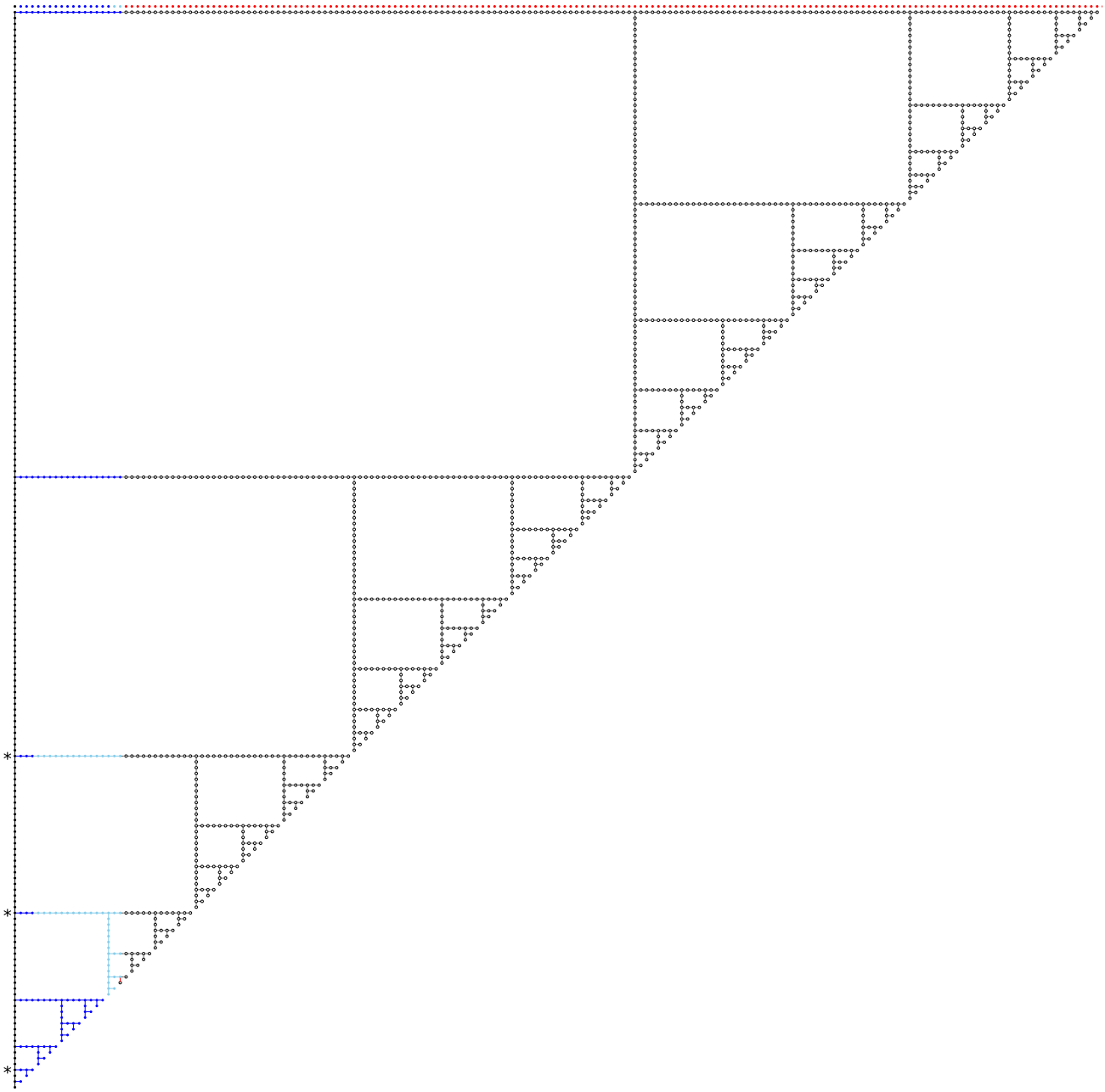


Figure E.4: Even degree isogeny with kernel of bad order; parameter set $p751$
Isogeny degree: $2^{372}$
Breaking point: $o = 5$
Kernel point degree: $3^{o-1}$

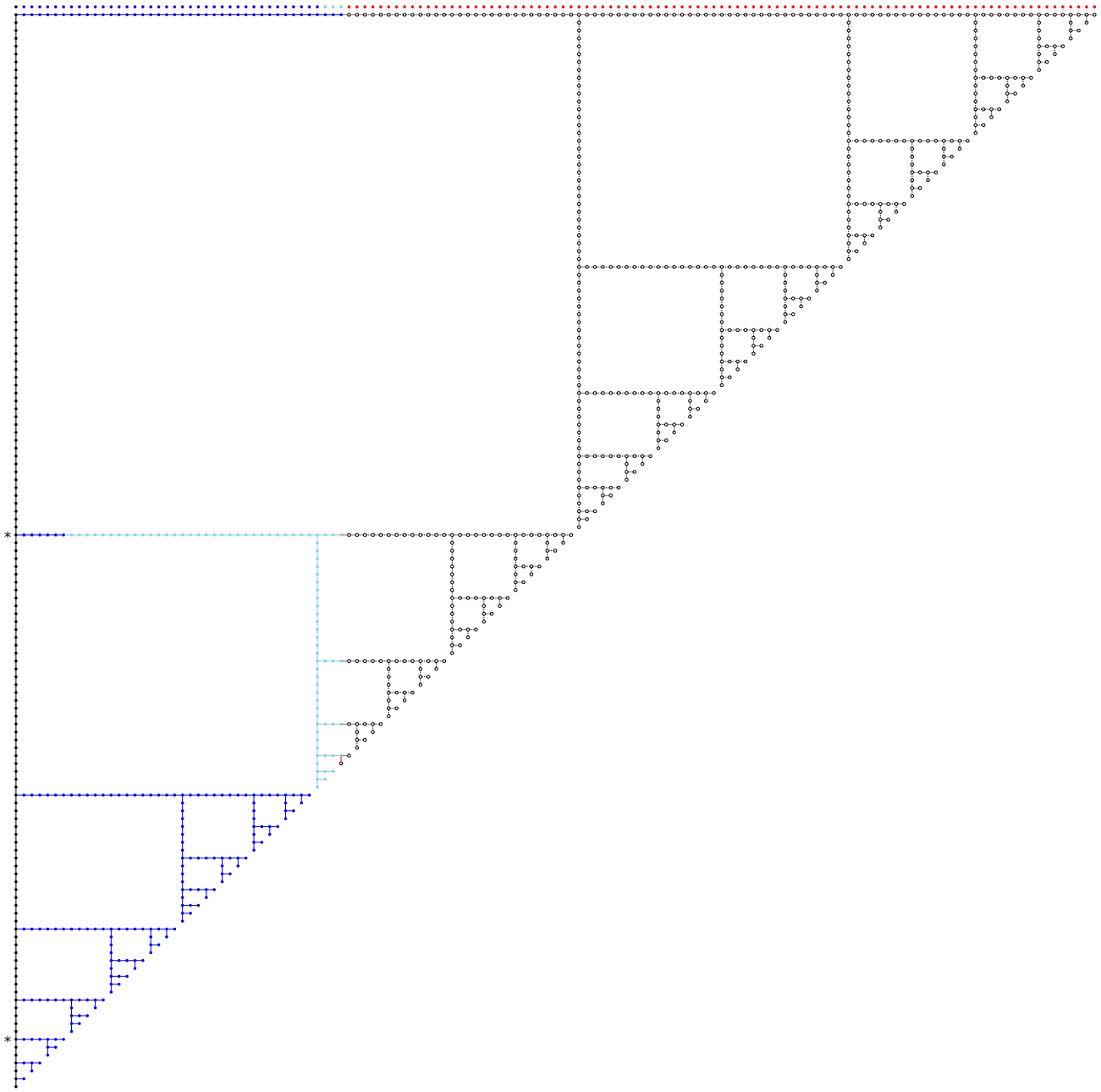


Figure E.5: Odd degree isogeny with kernel of bad order; parameter set $p434$.
Isogeny degree: $3^{137}$.
Breaking point: $o = 9$.
Kernel point degree: $2^{o-1}$.
Kernel point independent of $T$.

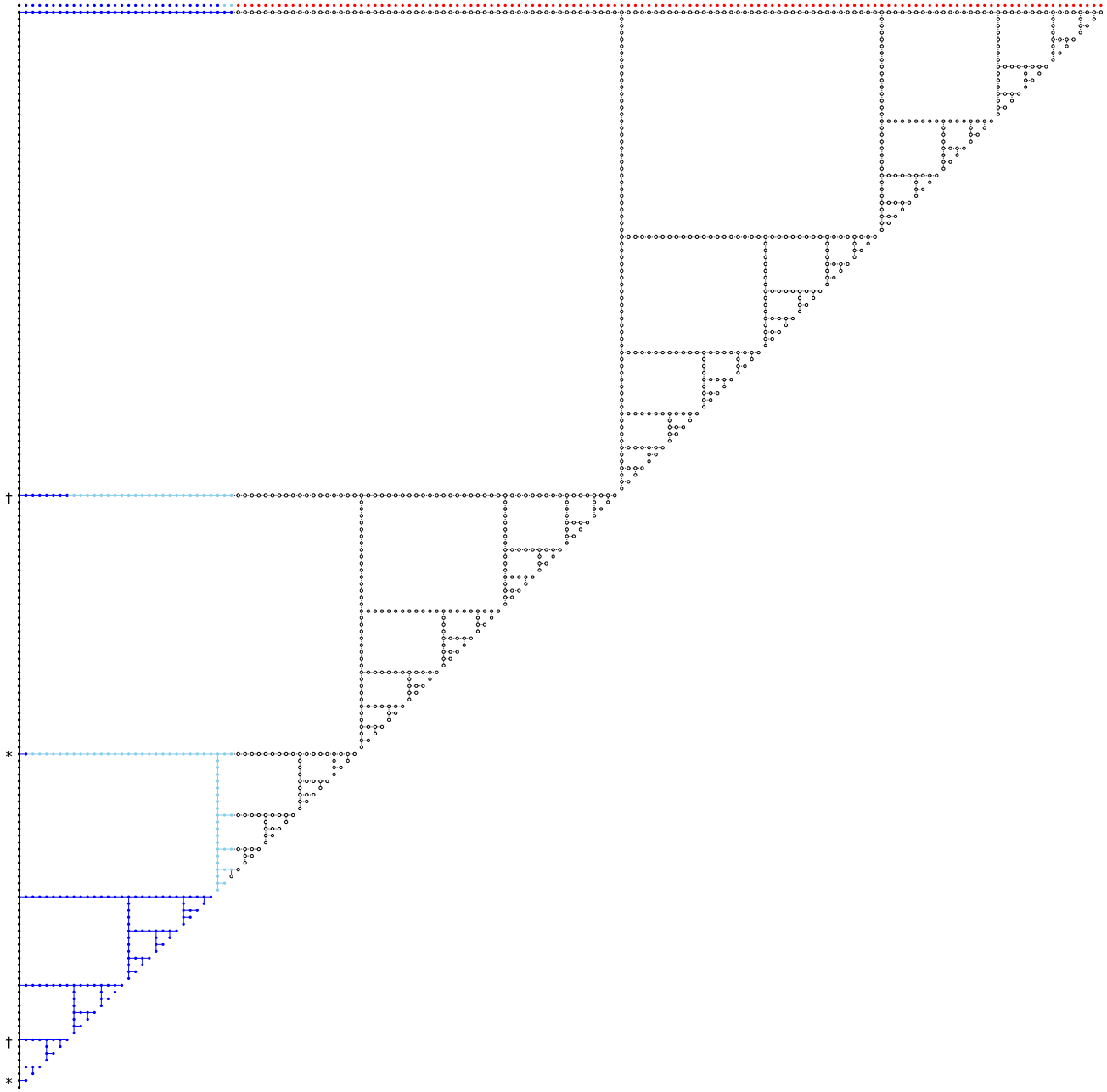



Figure E.6: Odd degree isogeny with kernel of bad order; parameter set $p503$.
Isogeny degree: $3^{159}$.
Breaking point: $o = 7$.
Kernel point degree: $2^{o-1}$.
Kernel point independent of $T$.

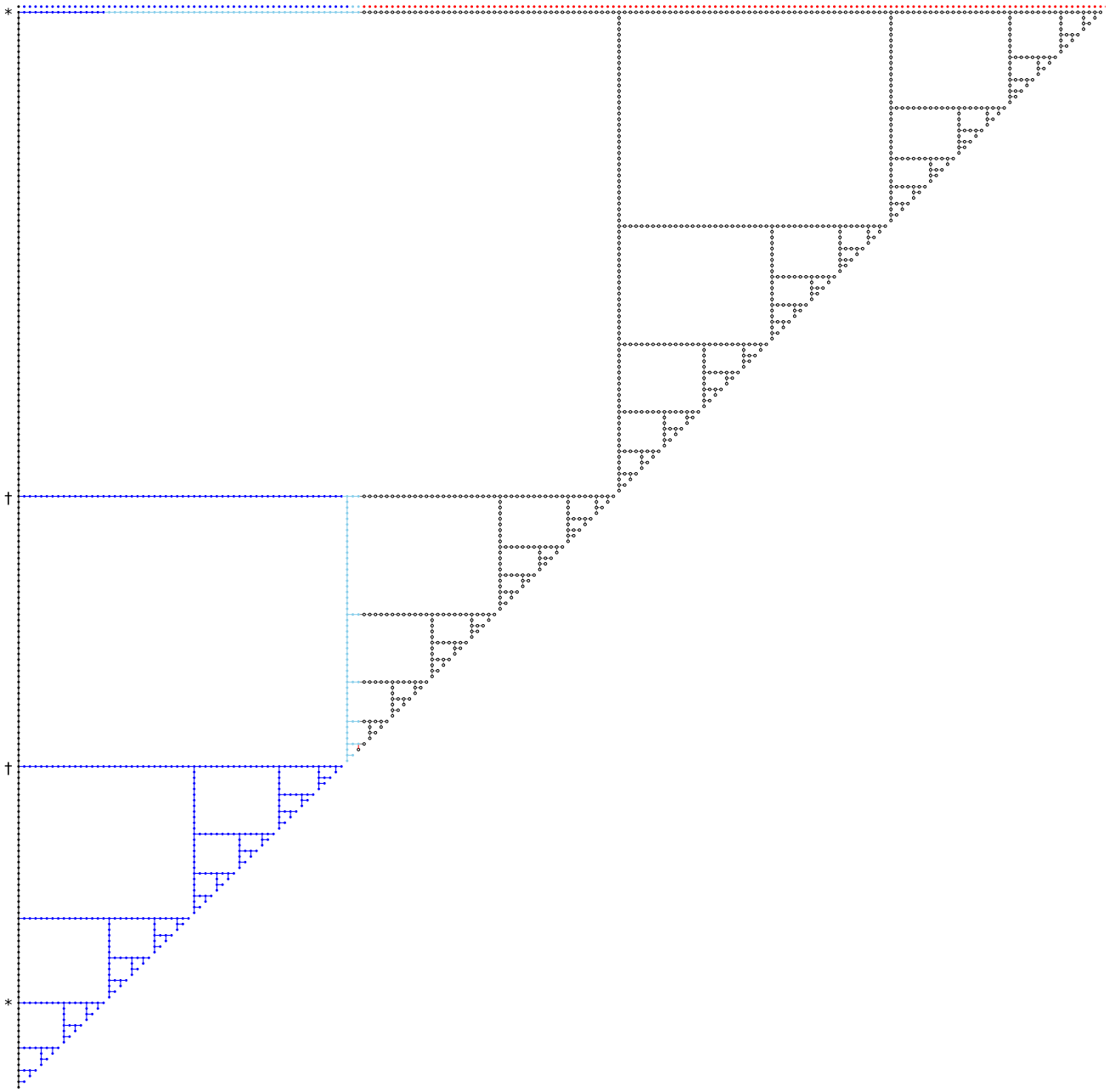



Figure E.7: Odd degree isogeny with kernel of bad order; parameter set $p610$.
Isogeny degree: $3^{192}$.
Breaking point: $o = 7$.
Kernel point degree: $2^{o-1}$.
Kernel point independent of $T$.

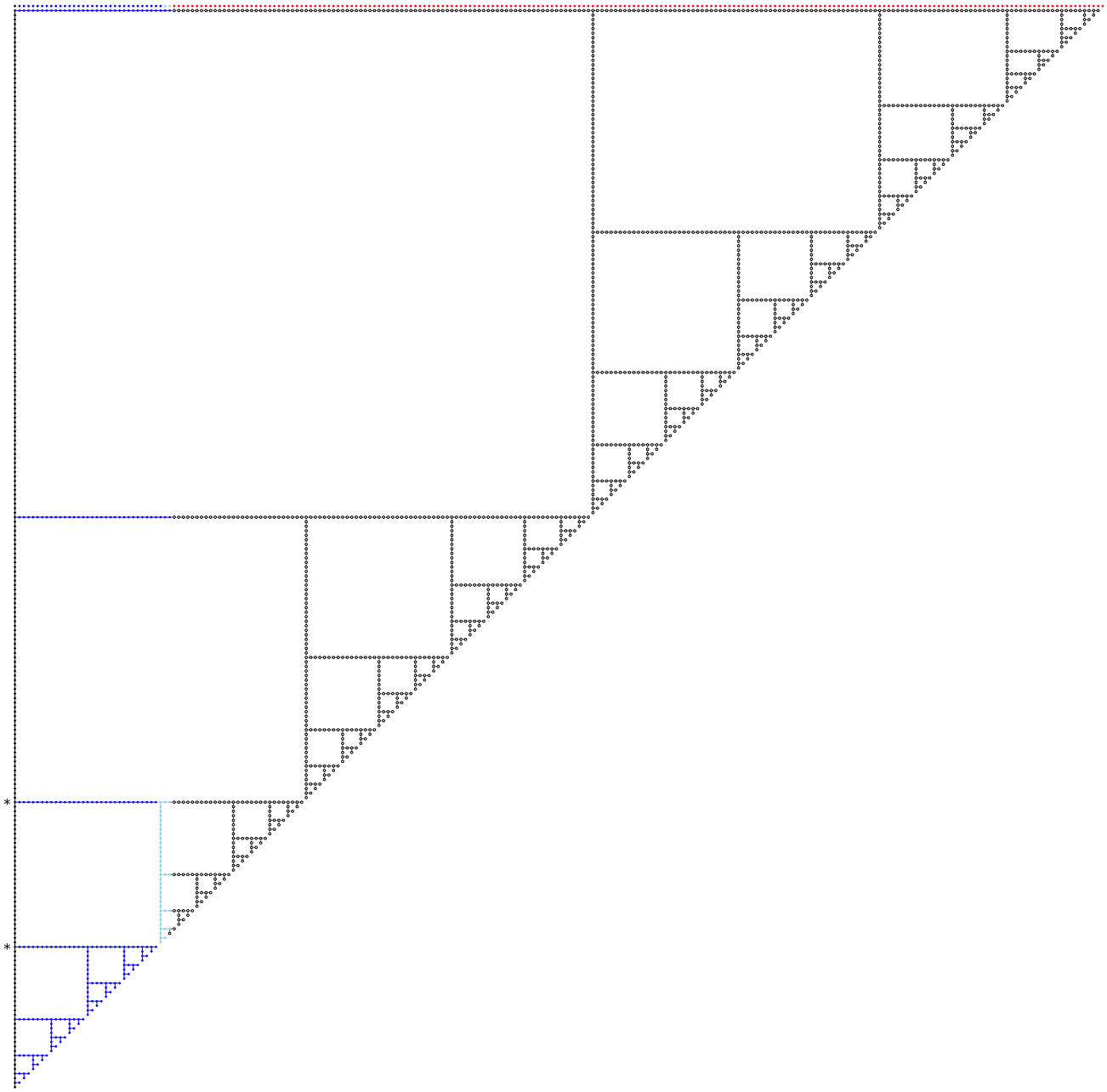


Figure E.8: Odd degree isogeny with kernel of bad order; parameter set $p751$.
Isogeny degree: $3^{239}$.
Breaking point: $o = 8$.
Kernel point degree: $2^{o-1}$.
Kernel point independent of $T$.

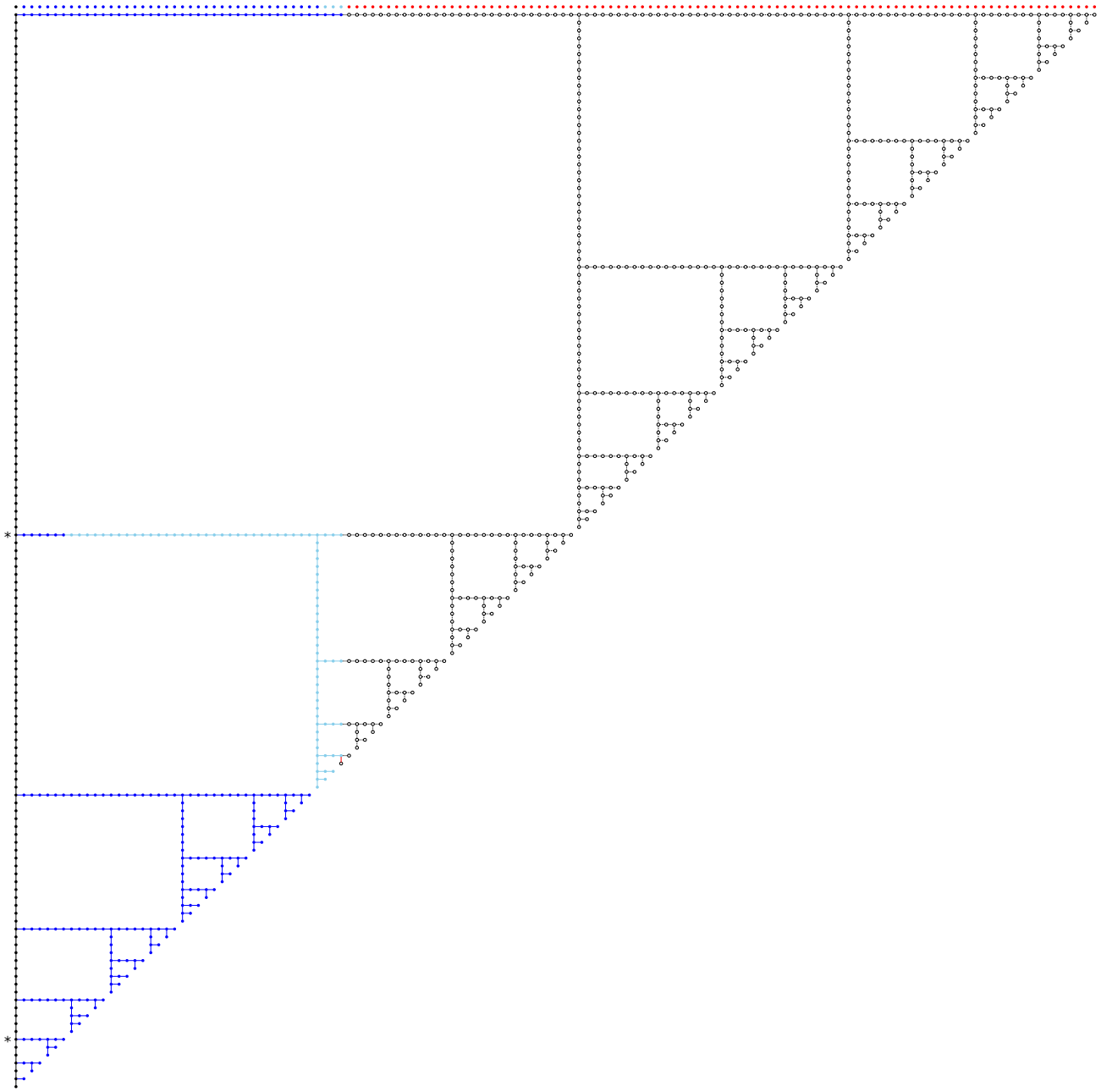


Figure E.9: Odd degree isogeny with kernel of bad order; parameter set $p434$.
Isogeny degree: $3^{137}$.
Breaking point: $o = 9$.
Kernel point degree: $2^{o-1}$.
Kernel point and $T$ are dependent.

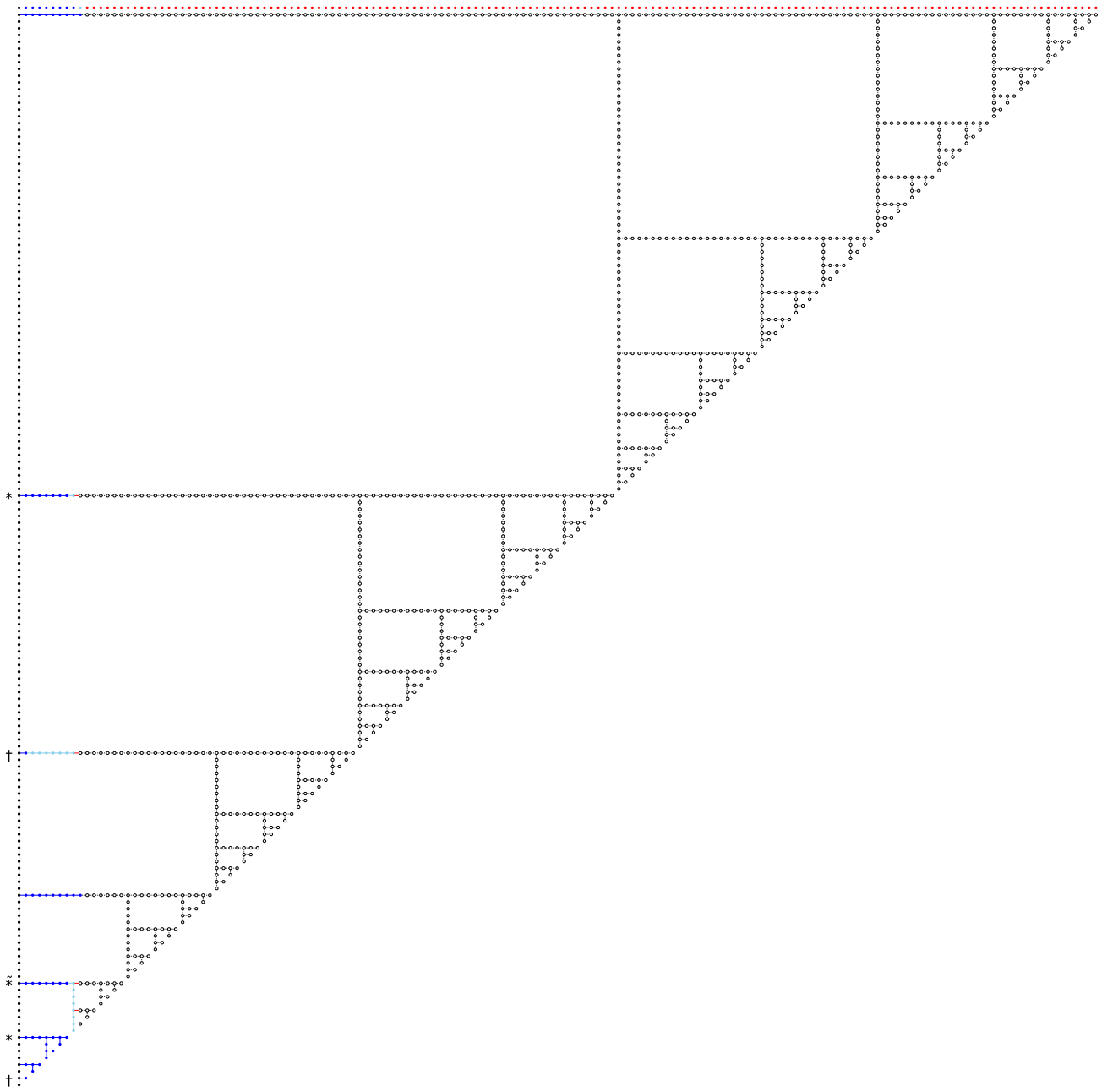


Figure E.10: Odd degree isogeny with kernel of bad order; parameter set $p503$.
Isogeny degree: $3^{159}$.
Breaking point: $o = 7$.
Kernel point degree: $2^{o-1}$.
Kernel point and $T$ are dependent.

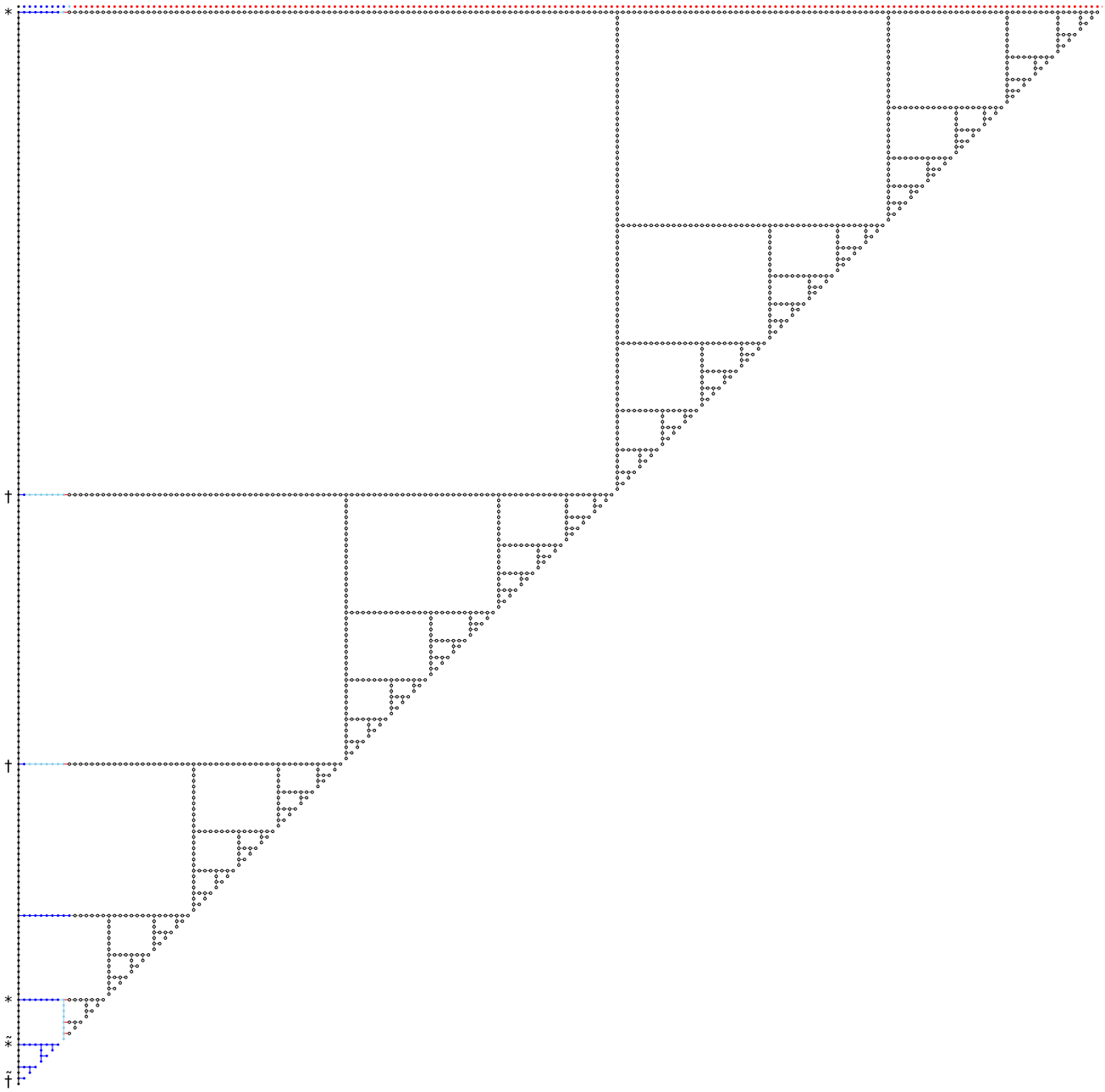



Figure E.11: Odd degree isogeny with kernel of bad order; parameter set $p610$.
Isogeny degree: $3^{192}$.
Breaking point: $o = 7$.
Kernel point degree: $2^{o-1}$.
Kernel point and $T$ are dependent.

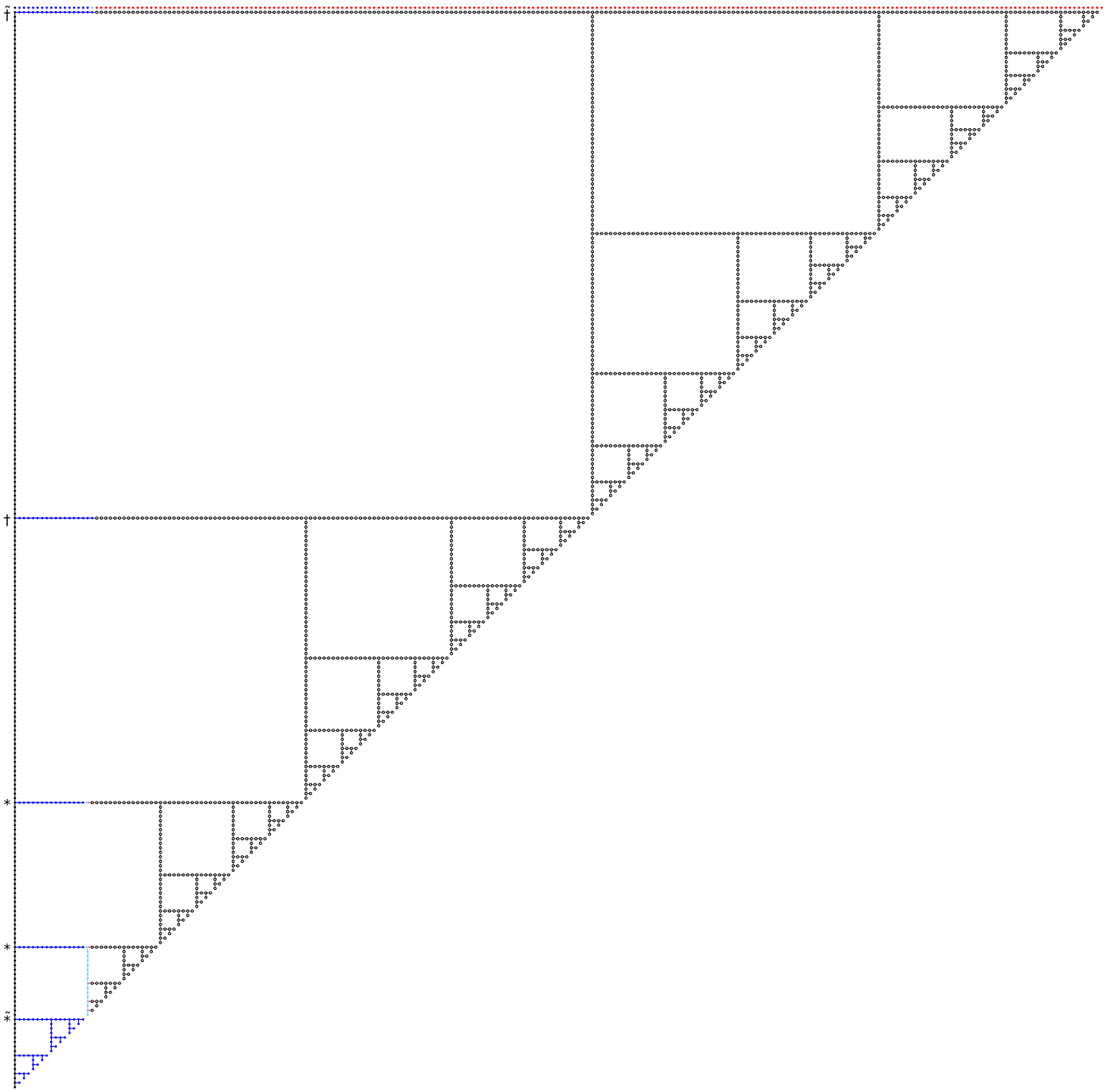


Figure E.12: Odd degree isogeny with kernel of bad order; parameter set $p751$.
Isogeny degree: $3^{239}$.
Breaking point: $o = 8$.
Kernel point degree: $2^{o-1}$.
Kernel point and $T$ are dependent.

# F Blocking zero-value attacks

We list all functions in SIKE that compute arithmetic operations in the field $\mathbb{F}_{p^2}$. These functions take as input elliptic curve points, or elliptic curve coefficients, and output (or update) elliptic curve points or elliptic curve coefficients. Additionally, there is the $j$-invariant, which takes an elliptic curve coefficient and outputs an affine $j$-invariant. We analyse all the arithmetic subfunctions which compute field additions/subtractions/multiplications/inversions and show that they never output the field element 0. Whether or not the element 0 is computed directly depends on the input of the higher level functions. All of the higher level functions are provided in separate tables, exactly as they are implemented in SIKE [JAC+17]. The assumptions that are made on inputs stem directly from the CLN test. In the tables different curve representations (see Section 2.5.2, page 25) are used analogously.

Table F.1: xDBL – point doubling, see formula (2.37) on page 27.
**Input:** Point $[X:Z]$, curve $[A_{24}^{+}:C_{24}]$.
**Assumption:** Order of $[X:Z]$ is $2^k$ for some $k \geq 2$ and $[2^{k-1}][X:Z] \neq [0:1]$, $A \neq \pm 2C$, $C \neq 0$.
**Output:** Point $[x:z] = [2][X:Z]$.

| Step | Variable | Formula | Factorisation | Roots |
|---|---|---|---|---|
| 1 | $t_0$ | $X-Z$ | $X-Z$ | $[1:1]$ |
| 2 | $t_1$ | $X+Z$ | $X+Z$ | $[-1:1]$ |
| 3 | $t_0$ | $t_0^2$ | $(X-Z)^2$ | $[1:1]$ |
| 4 | $t_1$ | $t_1^2$ | $(X+Z)^2$ | $[-1:1]$ |
| 5 | $z$ | $C_{24} \times t_0$ | $4C(X-Z)^2$ | $[1:1]$ |
| 6 | $x$ | $z \times t_1$ | $4C(X-Z)^2(X+Z)^2$ | $[\pm 1:1]$ |
| 7 | $t_1$ | $t_1 - t_0$ | $4XZ$ | $[0:1],[1:0]$ |
| 8 | $t_0$ | $A_{24}^{+} \times t_1$ | $4(A+2C)XZ$ | $[0:1],[1:0]$ |
| 9 | $z$ | $z + t_0$ | $4C(X^2 + aXZ + Z^2)$ | Points of order 2 |
| 10 | $z$ | $z \times t_1$ | $4C\ 4XZ(X^2 + aXZ + Z^2)$ | 2-torsion |

By assumption the input $[X:Z]$ in Table F.1 is independent from $[0:1]$ and therefore from $[\pm 1:1]$. Moreover $[X:Z] \neq [1:0]$ due to the order constraint, so the point $[X:Z]$ is different from the listed roots. Furthermore the input curve is not degenerate/singular/undefined, so it can never lead to computing an intermediate 0 value. This is not explicitly stated in the table but it stems from the factorisation formulas.

Table F.2: xTPL – point tripling, see formula (2.39) on page 27.
**Input:** Point $[X:Z]$, curve $[A_{24}^- : A_{24}^+]$.
**Assumption:** Order of $[X:Z]$ is $3^k$ with $k \geq 2$, $A \neq \pm 2C$, $C \neq 0$.
**Output:** Point $[x:z] = [3][X:Z]$.

| Step | Variable | Formula | Factorisation | Roots |
|---|---|---|---|---|
| 1 | $t_0$ | $X - Z$ | $X - Z$ | $[1:1]$ |
| 2 | $t_2$ | $t_0^2$ | $(X-Z)^2$ | $[1:1]$ |
| 3 | $t_1$ | $X + Z$ | $X + Z$ | $[-1:1]$ |
| 4 | $t_3$ | $t_1^2$ | $(X+Z)^2$ | $[-1:1]$ |
| 5 | $t_4$ | $t_0 + t_1$ | $2X$ | $[0:1]$ |
| 6 | $t_0$ | $t_1 - t_0$ | $2Z$ | $[1:0]$ |
| 7 | $t_1$ | $t_4^2$ | $4X^2$ | $[0:1]$ |
| 8 | $t_1$ | $t_1 - t_3$ | $(3X+Z)(X-Z)$ | $[1:1], [-1:3]$(!) |
| 9 | $t_1$ | $t_1 - t_2$ | $2(X-Z)(X+Z)$ | $[\pm 1:1]$ |
| 10 | $t_5$ | $t_3 \times A_{24}^+$ | $(A+2C)(X+Z)^2$ | $[-1:1]$ |
| 11 | $t_3$ | $t_3 \times t_5$ | $(A+2C)(X+Z)^4$ | $[-1:1]$ |
| 12 | $t_6$ | $A_{24}^- \times t_2$ | $(A-2C)(X-Z)^2$ | $[1:1]$ |
| 13 | $t_2$ | $t_2 \times t_6$ | $(A-2C)(X-Z)^4$ | $[1:1]$ |
| 14 | $t_3$ | $t_2 - t_3$ | $-4C(X^4 + 2a + 6X^2Z^2 + 2aXZ^3 + Z^4)$ | Pts of ord 4 |
| 15 | $t_2$ | $t_5 - t_6$ | $4C(X^2 + aXZ + Z^2)$ | Pts of ord 2 |
| 16 | $t_1$ | $t_1 \times t_2$ | $8C(X+Z)(X-Z)(X^2 + aXZ + Z^2)$ | $[\pm 1:1]$, Pts of ord 2 |
| 17 | $t_2$ | $t_3 \times t_1$ | $-4C(X^4 - 6X^2Z^2 - 4aXZ^3 - 3Z^4)$ | Pts of ord 3 + $T$ |
| 18 | $t_2$ | $t_2^2$ | $16C^2(X^4 - 6X^2Z^2 - 4aXZ^3 - 3Z^4)^2$ | Pts of ord 3 + $T$ |
| 19 | $x$ | $t_2 \times t_4$ | $32XC^2(X^4 - 6X^2Z^2 - 4aXZ^3 - 3Z^4)^2$ | 3-torsion + $T$ |
| 20 | $t_1$ | $t_3 - t_1$ | $-4C(3X^4 + 4aX^3Z + 6X^2Z^2 - Z^4)$ | Pts of ord 3 |
| 21 | $t_1$ | $t_1^2$ | $16C^2(3X^4 + 4aX^3Z + 6X^2Z^2 - Z^4)^2$ | Pts of ord 3 |
| 22 | $z$ | $t_0 \times t_1$ | $32ZC^2(3X^4 + 4aX^3Z + 6X^2Z^2 - Z^4)^2$ | 3-torsion |

All the roots in Table F.2, with the exception of $[-1:3]$ are points of order $1, 2, 3, 4$ or $6$. The order of the input point $[X:Z]$ is $\geq 9$ by assumption, so it must be different from the aforementioned roots. The only possible threat comes from the point $[-1:3]$ in step 8. It seems out of reach to prove that the point $[-1:3]$ cannot be of order $3^k$ for $k \geq 2$ on supersingular elliptic curves. One could cook up a heuristic argument showing that such property happens with negligible probability relative to the prime $p$. To remediate this problem we provide an alternative solution – an algorithm for computing point tripling without using intermediate functions which have $[-1:3]$ for a root. This algorithm is provided in Table F.3.

Furthermore, the the curve coefficients never lead to computing 0 as an intermediate values. This stems from the fact that the input curve is not degenerate/singular/undefined. This is not explicitly stated in the table, but can be incurred from the factorisation formulas.

Table F.3: `xTPL_alt` – point tripling alternative algorithm, see formula (2.39) on page 27.
**Input:** Point $[X:Z]$, curve $[A_{24}^- : A_{24}^+]$.
**Assumption:** Order of $[X:Z]$ is $3^k$ with $k \geq 2$, $A \neq \pm 2C$, $C \neq 0$.
**Output:** Point $[x:z] = [3][X:Z]$.

| Step | Variable | Formula | Factorisation | Roots |
|---|---|---|---|---|
| 1 | $t_2$ | $X - Z$ | $X - Z$ | $[1:1]$ |
| 2 | $t_3$ | $X + Z$ | $X + Z$ | $[-1:1]$ |
| 3 | $t_1$ | $t_2 \times t_3$ | $(X - Z)(X + Z)$ | $[\pm 1:1]$ |
| 4 | $t_0$ | $t_3 - t_2$ | $2Z$ | $[1:0]$ |
| 5 | $t_4$ | $t_2 + t_3$ | $2X$ | $[0:1]$ |
| 6 | $t_2$ | $t_2^2$ | $(X - Z)^2$ | $[1:1]$ |
| 7 | $t_3$ | $t_3^2$ | $(X + Z)^2$ | $[-1:1]$ |
| 8 | $t_1$ | $t_1 + t_1$ | $2(X - Z)(X + Z)$ | $[\pm 1:1]$ |
| 9 | $t_5$ | $t_3 \times A_{24}^+$ | $(A + 2C)(X + Z)^2$ | $[-1:1]$ |
| 10 | $t_3$ | $t_3 \times t_5$ | $(A + 2C)(X + Z)^4$ | $[-1:1]$ |
| 11 | $t_6$ | $A_{24}^- \times t_2$ | $(A - 2C)(X - Z)^2$ | $[1:1]$ |
| 12 | $t_2$ | $t_2 \times t_6$ | $(A - 2C)(X - Z)^4$ | $[1:1]$ |
| 13 | $t_3$ | $t_2 - t_3$ | $-4C(X^4 + 2aX^3Z + 6X^2Z^2 + 2aXZ^3 + Z^4)$ | Pts of ord 4 |
| 14 | $t_2$ | $t_5 - t_6$ | $4C(X^2 + aXZ + Z^2)$ | Pts of ord 2 |
| 15 | $t_1$ | $t_1 \times t_2$ | $8C(X + Z)(X - Z)(X^2 + aXZ + Z^2)$ | $[\pm 1:1]$, Pts of ord 2 |
| 16 | $t_2$ | $t_3 \times t_1$ | $-4C(X^4 - 6X^2Z^2 - 4aXZ^3 - 3Z^4)$ | Pts of ord 3 + $T$ |
| 17 | $t_2$ | $t_2^2$ | $16C^2(X^4 - 6X^2Z^2 - 4aXZ^3 - 3Z^4)^2$ | Pts of ord 3 + $T$ |
| 18 | $x$ | $t_2 \times t_4$ | $32XC^2(X^4 - 6X^2Z^2 - 4aXZ^3 - 3Z^4)^2$ | 3-torsion + $T$ |
| 19 | $t_1$ | $t_3 - t_1$ | $-4C(3X^4 + 4aX^3Z + 6X^2Z^2 - Z^4)$ | Pts of ord 3 |
| 20 | $t_1$ | $t_1^2$ | $16C^2(3X^4 + 4aX^3Z + 6X^2Z^2 - Z^4)^2$ | Pts of ord 3 |
| 21 | $z$ | $t_0 \times t_1$ | $32ZC^2(3X^4 + 4aX^3Z + 6X^2Z^2 - Z^4)^2$ | 3-torsion |

The alternate point tripling algorithm (Table F.3) has only points of order $1, 2, 3, 4$ and $6$ for roots, while the input point is of order $3^k$ with $k \geq 2$. Therefore it can never compute the value 0 during any of the steps.

When compared to the algorithm in Table F.2, the alternative algorithm computes one extra multiplication, but one squaring and one addition less. This leads to about 1% - 1.5% performance overhead during a tripling computation.

Table F.4: xDBLADD – point double-and-add, see Algorithm (4) on page 29.
**Input:** Points $Q = [X_Q : Z_Q]$, $P = [X_P : Z_P]$, and $QP = [X_{QP} : Z_{QP}]$ and curve constant $a$ as $a_{24} = (a+2)/4$.
**Assumption:** The points $Q, P, QP$ are on the curve of coefficient $a$,
The point $Q$ is independent of $P$ and $QP = Q - P$,
Either all points are of order $3^{e_3}$, or
$P$ and $QP$ are of order $2^{e_2}$, $Q$ is of order $2^k$ with $k \geq 1$ and $[2^{k-1}]Q = T$
**Output:** $Q$ and $P$ are updated to $[2]Q$ and $Q + P$.

| Step | Variable | Formula | Factorisation | Roots |
|---|---|---|---|---|
| 1 | $t_0$ | $X_Q + Z_Q$ | $X_Q + Z_Q$ | $[-1:1]$ |
| 2 | $t_1$ | $X_Q - Z_Q$ | $X_Q - Z_Q$ | $[1:1]$ |
| 3 | $X_Q$ | $t_0^2$ | $(X_Q + Z_Q)^2$ | $[-1:1]$ |
| 4 | $t_2$ | $X_P - Z_P$ | $X_P - Z_P$ | $[1:1]$ |
| 5 | $X_P$ | $X_P + Z_P$ | $X_P + Z_P$ | $[-1:1]$ |
| 6 | $t_0$ | $t_0 \times t_2$ | $(X_Q + Z_Q)(X_P - Z_P)$ | $[\pm 1:1]$ |
| 7 | $Z_Q$ | $t_1^2$ | $(X_Q - Z_Q)^2$ | $[1:1]$ |
| 8 | $t_1$ | $t_1 \times X_P$ | $(X_Q - Z_Q)(X_P + Z_P)$ | $[\pm 1:1]$ |
| 9 | $t_2$ | $X_Q - Z_Q$ | $4X_Q Z_Q$ | $[1:0], [1:0]$ |
| 10 | $X_Q$ | $X_Q \times Z_Q$ | $(X_Q - Z_Q)^2 (X_Q + Z_Q)^2$ | $[\pm 1:1]$ |
| 11 | $X_P$ | $a_{24} \times t_2$ | $a_{24} 4X_Q Z_Q$ | $[1:0], [0:1]$ |
| 12 | $Z_P$ | $t_0 - t_1$ | $-2(X_Q Z_P - X_P Z_Q)$ | $[X_P : Z_P] = [X_Q : Z_Q]$ |
| 13 | $Z_Q$ | $X_P + Z_Q$ | $X_Q^2 + aX_Q Z_Q + Z_Q^2$ | Pts of order 2 |
| 14 | $X_P$ | $t_0 + t_1$ | $2(X_Q X_P - Z_Q Z_P)$ | $[X_P : Z_P] = [X_Q : Z_Q] + T$ |
| 15 | $Z_Q$ | $Z_Q \times t_2$ | $4X_Q Z_Q (X_Q^2 + aX_Q Z_Q + Z_Q^2)$ | 2-torsion |
| 16 | $Z_P$ | $Z_P^2$ | $4(X_Q Z_P - X_P Z_Q)^2$ | $[X_P : Z_P] = [X_Q : Z_Q]$ |
| 17 | $X_P$ | $X_P^2$ | $4(X_Q X_P - Z_Q Z_P)^2$ | $[X_P : Z_P] = [X_Q : Z_Q] + T$ |
| 18 | $Z_P$ | $Z_P \times X_{QP}$ | $4X_{QP}(X_Q Z_P - X_P Z_Q)^2$ | $[0:1], [X_P : Z_P] = [X_Q : Z_Q]$ |
| 19 | $X_P$ | $X_P \times Z_{QP}$ | $4Z_{QP}(X_Q X_P - Z_Q Z_P)^2$ | $[1:0], [X_P : Z_P] = [X_Q : Z_Q] + T$ |

The xDBLADD procedure from Table F.4 is called multiple times during the execution of the three-point ladder. The root equations in steps $12-14$ and $16-19$ are never satisfied because of the independence assumption. If the input of the three-point ladder is a public key consisting of points of order $3^{e_3}$, then the order assumption implies that none of the points are equal to the roots in steps $1-11$, 13, 15, 18 and 19. If the input of the three-point ladder is a public key consisting of points of order $2^{e_2}$, and $Q$ is of order $2^k$, then the steps $4, 5, 13, 18, 19$ can be safely ignored due to the order constraint of points $P$ and $QP$. Furthermore in steps 6 and 8 the value 0 cannot stem from the point $P$ because of the same constraint. For the remaining steps, the only way that 0 can show up is if it stems from the point $Q$. However, the point $Q$ is always known during the three-point ladder computation, and it does not depend on the secret key. It will always lead to computing 0 in either steps $1, 3, 6, 10$ (if $Q = [-1:1]$) or $2, 7, 8, 10$ (if $Q = [1:1]$) in the penultimate loop iteration of the three-point ladder, and in steps $9, 11, 15$ (when $Q = [0:1]$) in the last loop iteration of the three-point ladder. This is correct behaviour of the algorithm, it does not depend on the secret key, and cannot be exploited by an attacker.

Table F.5: `get_2_isog` – 2-isogeny computation, see formula (2.41) on page 27.
**Input:** Kernel point $[X_2 : Z_2]$.
**Assumption:** Kernel point is of order 2, different from $T = [0 : 1]$.
**Output:** Image curve $[a_{24}^+ : c_{24}]$ of isogeny with kernel $\langle [X_2 : Z_2] \rangle$.

| Step | Variable | Formula | Factorisation | Roots |
|---|---|---|---|---|
| 1 | $a_{24}^+$ | $X_2^2$ | $X_2^2$ | $[0 : 1]$ |
| 2 | $c_{24}$ | $Z_2^2$ | $Z_2^2$ | $[1 : 0]$ |
| 3 | $a_{24}^+$ | $c_{24} - a_{24}^+$ | $-(X_2 - Z_2)(X_2 + Z_2)$ | $[\pm 1 : 1]$ |

In Table F.5, the 2-isogeny kernel point is always a point of order 2 which is different from $T = [0 : 1]$. Therefore the kernel point is never equal to any of the roots, which means that the value 0 is never computed during the execution of the algorithm. Furthermore, the output curve is never a degenerate, singular or undefined curve.

Table F.6: `eval_2_isog` – 2-isogeny evaluation, see formula (2.42) on page 28.
**Input:** Kernel point $[X_2 : Z_2]$, point to be evaluated $[X : Z]$.
**Assumption:** Kernel point is of order 2, different from $T = [0 : 1]$,
Point $[X : Z]$ satisfies $[2^k][X : Z] = [X_2 : Z_2]$ with $k \geq 1$.
**Output:** Image $[x : z]$ of point $[X : Z]$ through the isogeny with kernel $\langle [X_2 : Z_2] \rangle$.

| Step | Variable | Formula | Factorisation | Roots |
|---|---|---|---|---|
| 1 | $t0$ | $X_2 + Z_2$ | $X_2 + Z_2$ | $[-1 : 1]$ |
| 2 | $t1$ | $X_2 - Z_2$ | $X_2 - Z_2$ | $[1 : 1]$ |
| 3 | $t2$ | $X + Z$ | $X + Z$ | $[-1 : 1]$ |
| 4 | $t3$ | $X - Z$ | $X - Z$ | $[1 : 1]$ |
| 5 | $t0$ | $t0 \times t3$ | $(X_2 + Z_2)(X - Z)$ | $[\pm 1 : 1]$ |
| 6 | $t1$ | $t1 \times t2$ | $(X_2 - Z_2)(X + Z)$ | $[\pm 1 : 1]$ |
| 7 | $t2$ | $t0 + t1$ | $2(XX_2 - ZZ_2)$ | $[X : Z] = [X_2 : Z_2] + T$ |
| 8 | $t3$ | $t0 - t1$ | $2(XZ_2 - ZX_2)$ | $[X : Z] = [X_2 : Z_2]$ |
| 9 | $x$ | $X \times t2$ | $2X(XX_2 - ZZ_2)$ | $[X : Z] \in \langle [X_2 : Z_2] \rangle + T$ |
| 10 | $z$ | $Z \times t3$ | $2Z(XZ_2 - ZX_2)$ | $[X : Z] \in \langle [X_2 : Z_2] \rangle$ |

In Table F.6, the input point $[X : Z]$ lies above the kernel point $[X_2 : Z_2]$ (i.e., $[2^k][X : Z] = [X_2 : Z_2]$ with $k \geq 1$). Therefore the input point is independent of $[\pm 1 : 1]$ and $[1 : 0]$, which implies that the input point and the kernel point can never be equal to the roots in steps $1 - 6$. Since $[X : Z]$ is of a strictly larger order than $[X_2 : Z_2]$, we cannot have any of the constraints in lines $7 - 10$. Therefore 0 is never computed during 2-isogeny evaluation.

Table F.7: `get_3_isog` – 3-isogeny computation, see formula (2.43) on page 28.
**Input:** Kernel point $[X_3 : Z_3]$.
**Assumption:** Kernel point is of order 3.
**Output:** Image curve $[a_{24}^+ : a_{24}^-]$ of isogeny with kernel $\langle [X_3 : Z_3] \rangle$, and field elements $k_1, k_2, k_3$.

| Step | Variable | Formula | Factorisation | Roots |
|---|---|---|---|---|
| 1 | $k_1$ | $X_3 - Z_3$ | $X_3 - Z_3$ | $[1:1]$ |
| 2 | $t_0$ | $k_1^2$ | $(X_3 - Z_3)^2$ | $[1:1]$ |
| 3 | $k_2$ | $X_3 + Z_3$ | $X_3 + Z_3$ | $[-1:1]$ |
| 4 | $t_1$ | $k_2^2$ | $(X_3 + Z_3)^2$ | $[-1:1]$ |
| 5 | $t_3$ | $X_3 + X_3$ | $2X_3$ | $[0:1]$ |
| 6 | $t_3$ | $t_3^2$ | $4X_3^4$ | $[0:1]$ |
| 7 | $t_2$ | $t_3 - t_0$ | $(3X_3 - Z_3)(X_3 + Z_3)$ | $[-1:1], [1:3]$ |
| 8 | $t_3$ | $t_3 - t_1$ | $(3X_3 + Z_3)(X_3 - Z_3)$ | $[1:1], [-1:3]$ |
| 9 | $t_4$ | $t_3 + t_0$ | $4X_3(X_3 - Z_3)$ | $[0:1], [1:1]$ |
| 10 | $t_4$ | $t_4 + t_4$ | $8X_3(X_3 - Z_3)$ | $[0:1], [1:1]$ |
| 11 | $t_4$ | $t_1 + t_4$ | $(3X_3 - Z_3)^2$ | $[1:3]$ |
| 12 | $a_{24}^-$ | $t_2 \times t_4$ | $(3X_3 - Z_3)^3(X_3 + Z_3)$ | $[-1:1], [1:3]$ |
| 13 | $t_4$ | $t_1 + t_2$ | $4X_3(X_3 + Z_3)$ | $[0:1], [1:1]$ |
| 14 | $t_4$ | $t_4 + t_4$ | $8X_3(X_3 + Z_3)$ | $[0:1], [1:1]$ |
| 15 | $t_4$ | $t_0 + t_4$ | $(3X_3 + Z_3)^2$ | $[0:1], [1:1]$ |
| 16 | $a_{24}^+$ | $t_3 \times t_4$ | $(3X_3 + Z_3)^3(X_3 - Z_3)$ | $[1:1], [-1:3]$ |

The three isogeny computation, shown in Table F.7 takes for input a kernel point of order 3. The roots of the subfunctions are points in the 2-torsion, and the points $[\pm 1 : 3]$. The kernel point can never be equal to one of the 2-torsion points due to the order constraint. The same is actually true for the points $[\pm 1 : 3]$ when the underlying curve is not singular or undefined. We leave as an exercise to the reader to plug in the point $[\pm 1 : 3]$ into the tripling formula in equation 2.39 or in Table F.2 or F.3, and conclude that $[3][\pm 1 : 3] = [1 : 0]$ if and only if the underlying curve is singular. Therefore, the value 0 is never computed during the 3-isogeny computation.

Table F.8: `eval_3_isog` – 3-isogeny evaluation, see formula (2.44) on page 28.
**Input:** Kernel point $[X_3 : Z_3]$, point to be evaluated $[X : Z]$, field elements $k_1, k_2, k_3$ (Table F.7).
**Assumption:** Kernel point is of order 3,
Point $[X : Z]$ satisfies $[3^k][X : Z] = [X_3 : Z_3]$ with $k \geq 1$.
**Output:** Image $[x : z]$ of point $[X : Z]$ through the isogeny with kernel $\langle [X_3 : Z_3] \rangle$.

| Step | Variable | Formula | Factorisation | Roots |
|---|---|---|---|---|
| 1 | $t_0$ | $X + Z$ | $X + Z$ | $[-1 : 1]$ |
| 2 | $t_1$ | $X - Z$ | $X - Z$ | $[1 : 1]$ |
| 3 | $t_0$ | $t_0 \times k_1$ | $(X_3 - Z_3)(X + Z)$ | $[\pm 1 : 1]$ |
| 4 | $t_1$ | $t_1 \times k_2$ | $(X_3 + Z_3)(X - Z)$ | $[\pm 1 : 1]$ |
| 5 | $t_2$ | $t_0 + t_1$ | $2(XX_3 - ZZ_3)$ | $[X : Z] = [X_3 : Z_3] + T$ |
| 6 | $t_0$ | $t_1 - t_0$ | $-2(XZ_3 - ZX_3)$ | $[X : Z] = [X_3 : Z_3]$ |
| 7 | $t_2$ | $t_2^2$ | $4(XX_3 - ZZ_3)^2$ | $[X : Z] = [X_4 : Z_4] + T$ |
| 8 | $t_0$ | $t_0^2$ | $4(XZ_3 - ZX_3)^2$ | $[X : Z] = [X_4 : Z_4]$ |
| 9 | $x$ | $X \times t_2$ | $4X(XX_3 - ZZ_3)^2$ | $[X : Z] \in \langle [X_3 : Z_3] \rangle + T$ |
| 10 | $z$ | $Z \times t_0$ | $4Z(XZ_3 - ZX_3)^2$ | $[X : Z] \in \langle [X_3 : Z_3] \rangle$ |

In Table F.8, the input point $[X : Z]$ and the kernel point $[X_3 : Z_3]$ are of order some power of 3, with $[X : Z]$ of strictly greater order than $[X_3 : Z_3]$. Therefore they are different from all the roots Table F.8, meaning that the value 0 is never computed during 3-isogeny evaluation.

Table F.9: `get_4_isog` – 4-isogeny computation, see formula (2.45) on page 28.
**Input:** Kernel point $[X_4 : Z_4]$.
**Assumption:** Kernel point is of order 4, different from $[\pm 1 : 1]$.
**Output:** Image curve $[a_{24}^+ : c_{24}]$ of isogeny with kernel $\langle [X_4 : Z_4] \rangle$, and field elements $k_1, k_2, k_3$.

| Step | Variable | Formula | Factorisation | Roots |
|---|---|---|---|---|
| 1 | $k_2$ | $X_4 - Z_4$ | $X_4 - Z_4$ | $[1 : 1]$ |
| 2 | $k_3$ | $X_4 + Z_4$ | $X_4 + Z_4$ | $[-1 : 1]$ |
| 3 | $k_1$ | $Z_4^2$ | $Z_4^2$ | $[1 : 0]$ |
| 4 | $k_1$ | $k_1 + k_1$ | $2Z_4^2$ | $[1 : 0]$ |
| 5 | $c_{24}$ | $k_1^2$ | $4Z_4^4$ | $[1 : 0]$ |
| 6 | $k_1$ | $k_1 + k_1$ | $4Z_4^2$ | $[1 : 0]$ |
| 7 | $a_{24}^+$ | $X_4^2$ | $X_4^2$ | $[0 : 1]$ |
| 8 | $a_{24}^+$ | $a_{24}^+ + a_{24}^+$ | $2X_4^2$ | $[0 : 1]$ |
| 9 | $a_{24}^+$ | $\left(a_{24}^+\right)^2$ | $4X_4^2$ | $[0 : 1]$ |

In Table F.9 the kernel point $[X_4 : Z_4]$ is of order 4 and different from $[\pm 1 : 1]$ by assumption. Therefore it is different from all the roots in the table, and thus the 0 value is never computed during 4-isogeny computation.

Table F.10: `eval_4_isog` – 4-isogeny evaluation, see formula (2.46) on page 28.
**Input:** Kernel point $[X_4 : Z_4]$, point to be evaluated $[X : Z]$, field elements $k_1, k_2, k_3$ from Table F.9.
**Assumption:** Kernel point is of order 4, different from $[\pm 1 : 1]$,
Point $[X : Z]$ satisfies $[4^k][X : Z] = [X_4 : Z_4]$ with $k \geq 1$.
**Output:** Image $[x : z]$ of point $[X : Z]$ through the isogeny with kernel $\langle [X_4 : Z_4] \rangle$.

| Step | Variable | Formula | Factorisation | Roots |
|---|---|---|---|---|
| 1 | $t_0$ | $X+Z$ | $X+Z$ | $[-1:1]$ |
| 2 | $t_1$ | $X-Z$ | $X-Z$ | $[1:1]$ |
| 3 | $x$ | $t_0 \times k_2$ | $(X_4-Z_4)(X+Z)$ | $[\pm 1:1]$ |
| 4 | $z$ | $t_1 \times k_3$ | $(X_4+Z_4)(X-Z)$ | $[\pm 1:1]$ |
| 5 | $t_0$ | $t_0 \times t_1$ | $(X-Z)(X+Z)$ | $[\pm 1:1]$ |
| 6 | $t_0$ | $t_0 \times k_1$ | $4Z_4^2(X-Z)(X+Z)$ | $[\pm 1:1], [1:0]$ |
| 7 | $t_1$ | $x+z$ | $2(XX_4 - ZZ_4)$ | $[X:Z] = [X_4:Z_4] + T$ |
| 8 | $z$ | $x-z$ | $-2(XZ_4 - ZX_4)$ | $[X:Z] = [X_4:Z_4]$ |
| 9 | $t_1$ | $t_1^2$ | $4(XX_4 - ZZ_4)^2$ | $[X:Z] = [X_4:Z_4] + T$ |
| 10 | $z$ | $z^2$ | $4(XZ_4 - ZX_4)^2$ | $[X:Z] = [X_4:Z_4]$ |
| 11 | $x$ | $t_0 + t_1$ | $4X(XX_4^2 + XZ_4^2 - 2ZX_4Z_4)$ | $[X:Z] = [X_4^2 + Z_4^2 : 2X_4Z_4] + T$ |
| 12 | $t_0$ | $z - t_0$ | $4Z(ZX_4^2 + ZZ_4^2 - 2XX_4Z_4)$ | $[X:Z] = [X_4^2 + Z_4^2 : 2X_4Z_4]$ |
| 13 | $x$ | $x \times t_1$ | $16X(XX_4 - ZZ_4)^2(XX_4^2 + XZ_4^2 - 2ZX_4Z_4)$ | $[X:Z] \in \langle [X_4:Z_4] \rangle + T$ |
| 14 | $z$ | $z \times t_0$ | $16Z(XZ_4 - ZX_4)^2(ZX_4^2 + ZZ_4^2 - 2XX_4Z_4)$ | $[X:Z] \in \langle [X_4:Z_4] \rangle$ |

In Table F.10 the points $[X : Z]$ and $[X_4 : Z_4]$ cannot be equal to any of the listed roots due to the assumptions. This is not obvious only in steps 11 and 12. The point $[X_4^2 + Z_4^2 : 2X_4Z_4]$ is equal to $[2][X_4 : Z_4]$. This formula is correct when $[X_4 : Z_4]$ is a point of order 4 different from $[\pm 1 : 1]$, which is true by assumption. The interested reader can prove the formula for $[2][X_4 : Z_4]$ by computing the doubling and comparing the output to the 4-division polynomial (polynomial whose roots are exactly the points of order 4), whose factor can be found in Table F.2, line 14.

Table F.11: `j_inv` – $j$-invariant computation, see formula (2.48) on page 28.
**Input:** Curve $[A:C]$.
**Assumption:** Input curve $[A:C]$ is not degenerate/singular/undefined, i.e., $A \neq \pm 2C$, $C \neq 0$.
**Output:** The $j$-invariant $j$ of the curve $[A:C]$.

| Step | Variable | Formula | Factorisation | Roots in $[A:C]$ |
|---|---|---|---|---|
| 1 | $j$ | $A^2$ | $A^2$ | $[0:1]$ |
| 2 | $t_1$ | $C^2$ | $C^2$ | $[1:0]$ |
| 3 | $t_0$ | $t_1 + t_1$ | $2C^2$ | $[1:0]$ |
| 4 | $t_0$ | $j - t_0$ | $A^2 - 2C^2$ | $[\pm\sqrt{2}:1]$ |
| 5 | $t_0$ | $t_0 - t_1$ | $(A^2 - 3C^2)$ | $[\pm\sqrt{3}:1]$ |
| 6 | $j$ | $t_0 - t_1$ | $(A^2 - 4C^2)$ | $[\pm 2:1]$ |
| 7 | $t_1$ | $t_1^2$ | $(A^2 - 4C^2)^2$ | $[\pm 2:1]$ |
| 8 | $j$ | $j \times t_1$ | $C^2(A^2 - 4C^2)^2$ | $[\pm 2:1]$ |
| 9 | $t_0$ | $t_0 + t_0$ | $2(A^2 - 3C^2)$ | $[\pm\sqrt{3}:1]$ |
| 10 | $t_0$ | $t_0 + t_0$ | $4(A^2 - 3C^2)$ | $[\pm\sqrt{3}:1]$ |
| 11 | $t_1$ | $t_0^2$ | $16(A^2 - 3C^2)^2$ | $[\pm\sqrt{3}:1]$ |
| 12 | $t_0$ | $t_0 \times t_1$ | $64(A^2 - 3C^2)^3$ | $[\pm\sqrt{3}:1]$ |
| 13 | $t_0$ | $t_0 + t_0$ | $128(A^2 - 3C^2)^3$ | $[\pm\sqrt{3}:1]$ |
| 14 | $t_0$ | $t_0 + t_0$ | $256(A^2 - 3C^2)^3$ | $[\pm\sqrt{3}:1]$ |
| 15 | $j$ | $1/j$ | $C^{-2}(A^2 - 4C^2)^{-2}$ | / / |
| 16 | $j$ | $z \times t_0$ | $256C^{-2}(A^2 - 3C^2)(A^2 - 4C^2)^{-2}$ | $[\pm\sqrt{3}:1]$ |

During the computation of the $j$-invariant (Table F.11), the value 0 can be computed in steps 4, 5, $9-14$ and 16 if the input curve coefficients are $[\pm\sqrt{2}:1]$ or $[\pm\sqrt{3}:1]$. Steps $1-3$ and $6-8$ can be disregarded because of the input assumption. The information that the attacker can obtain from observing the computation of a 0 is that the public curve (assumed to be provided by the attacker) and the curve of coefficient $\pm\sqrt{2}$ or $\pm\sqrt{3}$ are connected by the secret isogeny of the target party.

Finding an elliptic curve together with a well-chosen set of torsion generators (i.e. finding a public key) such that the target's secret isogeny goes from said curve to a known curve is a hard problem. In fact, we already know of such a case – the SIKE public parameter starting curve (as known curve) and the targets public curve (as public key, minus the points) are two curves connected with the target parties isogeny. However, finding a good set of torsion generators is equivalent to finding the secret isogeny itself, therefore assumed to be hard.

A small modification can be made in order to avoid computing a subfunction with the curve $[\pm\sqrt{2}:1]$ as a root. By computing $t_0 + t_1$ in step 4 and $j - t_0$ in step 5 we avoid the occurrence of the curve $[\pm\sqrt{2}:1]$ in the set of roots, and obtain another $[1:0]$ instead.

# G Conditional swap

```
static void swap_points(point_proj_t R, point_proj_t R2, const
  digit_t option)
{  // Swap points.
   // If option = 0 then R <- R and R2 <- R2, else if option = 0xFF
  ...FF then R <- R2 and R2 <- R
   digit_t temp;
   unsigned int i;

1     for(i = 0; i < NWORDS_FIELD; i++) {
2        temp = option & (R->X[0][i] ^ R2->X[0][i]);
3        R->X[0][i] = temp ^ R->X[0][i];
4        R2->X[0][i] = temp ^ R2->X[0][i];
5        temp = option & (R->X[1][i] ^ R2->X[1][i]);
6        R->X[1][i] = temp ^ R->X[1][i];
7        R2->X[1][i] = temp ^ R2->X[1][i];
8        temp = option & (R->Z[0][i] ^ R2->Z[0][i]);
9        R->Z[0][i] = temp ^ R->Z[0][i];
10       R2->Z[0][i] = temp ^ R2->Z[0][i];
11       temp = option & (R->Z[1][i] ^ R2->Z[1][i]);
12       R->Z[1][i] = temp ^ R->Z[1][i];
13       R2->Z[1][i] = temp ^ R2->Z[1][i];
   }
}
```

Figure G.1: Attacked source code of `swap_points` in C (simplified).

```
rsb   r8, r6, #0    /* mask = (0 - swap) */
add.w r2, r4, #92   /* R  */
add.w r3, r4, #540  /* R2 */
mov.w ip, #0        /* i = 0 */
<loop>:
ldr   r7, [r2, #0]  /* a = R->X[0][i] */
ldr   r1, [r3, #0]  /* b = R2->X[0][i] */
eor.w r0, r7, r1    /* a ^ b */
and.w r0, r0, r8    /* temp = mask & (a ^ b)*/
eors  r7, r0        /* a = temp ^ a */
eors  r1, r0        /* b = temp ^ b */
str.w r7, [r2], #4  /* R->X[0][i] = a */
str.w r1, [r3], #4  /* R2->X[0][i] = b */

...                 /* repeat with updated offset */

add.w ip, ip, #1    /* i++ */
cmp.w ip, #14       /* i < NWORDS_FIELD */
bne.n <loop>
```

Figure G.2: Compiled `swap_points` function with annotations (SIKEp434).

```
and.w %[u1], %[u1], #0xFFFFFFFD  /* u1 = random(4) & 0xFFFFFFFD */
and.w %[m1], %[u2], #0xFFFFFFFE  /* m1 = random(4) & 0xFFFFFFFE */
add.w %[u2], %[u1], %[swap]      /* u2 = u1 + swap */
add.w %[m2], %[m1], %[swap]      /* r  = m1 + swap */
add.w %[u1], %[u1], #1           /* u1 = u1 + 1 */
mul.w %[u1], %[u1], %[m2]        /* u1 = u1*r */
add.w %[u2], %[u2], %[swap]      /* u2 = u2 + swap */
mul.w %[u2], %[u2], %[m2]        /* u2 = u2*r */
sub.w %[m2], %[u1], %[u2]        /* m2 = u1 - u2 */
```

Figure G.3: Source code of the secure masks generation in assembly.

```
1   ldr.w %[a], [%[R]]            /* a = R[i]  */
2   ldr.w %[b], [%[R2]]           /* b = R2[i] */
3   eor.w %[tmp1], %[a], %[b]     /* tmp1 = a ^ b */
4   and.w %[tmp1], %[m1]          /* tmp1 = tmp1 & m1 */
5   eor.w %[b], %[b], %[tmp1]     /* a = a ^ tmp1 */
6   eor.w %[a], %[a], %[tmp1]     /* b = b ^ tmp1 */
7   eor.w %[tmp2], %[a], %[b]     /* tmp2 = a ^ b = R[i] ^ R2[i]*/
8   str.w %[b], [%[R2]]           /* R2[i] = b */
9   and.w %[tmp2], %[m2]          /* tmp2 = tmp2 & m2 */
10  str.w %[a], [%[R]]            /* R[i]  = a */
11  eor.w %[b], %[b], %[tmp2]     /* b = b ^ tmp2 */
12  eor.w %[a], %[a], %[tmp2]     /* a = a ^ tmp2 */
13  str.w %[a], [%[R]], #4        /* R[i]  = a */
14  str.w %[b], [%[R2]], #4       /* R2[i] = b */

...                               /* repeat with updated offset */
```

Figure G.4: Source code of the secure swapping operation in assembly.

# Bibliography


[AJK+16] Reza Azarderakhsh, David Jao, Kassem Kalach, Brian Koziel, and Christopher Leonardi. Key compression for isogeny-based cryptosystems. Cryptology ePrint Archive, Report 2016/229, 2016. https://eprint.iacr.org/2016/229. (page 4)

[Apo20] Daniel Apon. Passing the final checkpoint! NIST PQC 3rd round begins, 2020. https://meetings.ams.org/math/fall2020se/meetingapp.cgi/Paper/1656, https://www.scribd.com/document/474476570/PQC-Overview-Aug-2020-NIST. (page 4)

[APS19] Melissa Azouaoui, Romain Poussier, and François-Xavier Standaert. Fast side-channel security evaluation of ECC implementations - shortcut formulas for horizontal side-channel attacks against ECSM with the Montgomery ladder. In Ilia Polian and Marc Stöttinger, editors, *COSADE 2019*, volume 11421 of *LNCS*, pages 25–42. Springer, Heidelberg, April 2019. (page 55)

[AT03] Toru Akishita and Tsuyoshi Takagi. Zero-value point attacks on elliptic curve cryptosystem. In Colin Boyd and Wenbo Mao, editors, *ISC 2003*, volume 2851 of *LNCS*, pages 218–233. Springer, Heidelberg, October 2003. (page 69, 81)

[ATT+18] Aydin Aysu, Youssef Tobah, Mohit Tiwari, Andreas Gerstlauer, and Michael Orshansky. Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 81–88, 2018. (page 56)

[BBUV20] Ward Beullens, Tim Beyne, Aleksei Udovenko, and Giuseppe Vitto. Cryptanalysis of the Legendre PRF and generalizations. *IACR Trans. Symm. Cryptol.*, 2020(1):313–330, 2020. (page 37, 48, 50)

[BCH+21] Lejla Batina, Łukasz Chmielewski, Björn Haase, Niels Samwel, and Peter Schwabe. SCA-secure ECC in software – mission impossible? Cryptology ePrint Archive, Report 2021/1003, 2021. https://eprint.iacr.org/2021/1003. (page 98)

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, Heidelberg, August 2004. (page 56)

[BCP+14] Lejla Batina, Lukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. Online template attacks. In Willi Meier and Debdeep Mukhopadhyay, editors, *INDOCRYPT 2014*, volume 8885 of *LNCS*, pages 21–36. Springer, Heidelberg, December 2014. (page 56)

[Ber08] Daniel J. Bernstein. The ChaCha family of stream ciphers, 2008. (page 94)

[Ber15] Elwyn R. Berlekamp. *Algebraic Coding Theory - Revised Edition*. World Scientific Publishing Co., Inc., USA, 2015. (page 12)

[BF18a] Joppe W. Bos and Simon Friedberger. Arithmetic considerations for isogeny based cryptography. Cryptology ePrint Archive, Report 2018/376, 2018. https://eprint.iacr.org/2018/376. (page 4)

[BF18b] Joppe W. Bos and Simon J. Friedberger. Faster modular arithmetic for isogeny based crypto on embedded devices. Cryptology ePrint Archive, Report 2018/792, 2018. https://eprint.iacr.org/2018/792. (page 4)

[BFM+19] Joppe W. Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam. Assessing the feasibility of single trace power analysis of Frodo. In Carlos Cid and Michael J. Jacobson Jr:, editors, *SAC 2018*, volume 11349 of *LNCS*, pages 216–234. Springer, Heidelberg, August 2019. (page 56, 63)

[BV97] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997. (page 1)

[BZ10] Richard P. Brent and Paul Zimmermann. An o(m(n) log n) algorithm for the jacobi symbol, 2010. (page 52)

[Cas91] J. W. S. Cassels. *LMSST: 24 Lectures on Elliptic Curves*. London Mathematical Society Student Texts. Cambridge University Press, 1991. (page 13)

[CCJ04] Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity. *IEEE Transactions on Computers*, 53(6):760–768, 2004. (page 68)

[CFG+10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In Miguel Soriano, Sihan Qing, and Javier López, editors, *ICICS 10*, volume 6476 of *LNCS*, pages 46–61. Springer, Heidelberg, December 2010. (page 55)

[CGD17] Yann Le Corre, Johann Großschädl, and Daniel Dinu. Micro-architectural power simulator for leakage assessment of cryptographic software on ARM cortex-M3 processors. Cryptology ePrint Archive, Report 2017/1253, 2017. https://eprint.iacr.org/2017/1253. (page 98)

[CGD18] Yann Le Corre, Johann Großschädl, and Daniel Dinu. Micro-architectural power simulator for leakage assessment of cryptographic software on ARM cortex-M3 processors. In Junfeng Fan and Benedikt Gierlichs, editors, *COSADE 2018*, volume 10815 of *LNCS*, pages 82–98. Springer, Heidelberg, April 2018. (page 65, 68)

[CJS14] Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, Jan 2014. (page 3)

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 572–601. Springer, Heidelberg, August 2016. (page 4, 5, 85)

[Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin Kaya Koç and Christof Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 292–302. Springer, Heidelberg, August 1999. (page 66, 97)

[Cos19] Craig Costello. Supersingular isogeny key exchange for beginners. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 21–50. Springer, Heidelberg, August 2019. (page 13)

[Cou06] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. https://eprint.iacr.org/2006/291. (page 3)

[CS17] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic: The case of large characteristic fields. Cryptology ePrint Archive, Report 2017/212, 2017. https://ia.cr/2017/212. (page 86)

[CS18] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic - the case of large characteristic fields. *Journal of Cryptographic Engineering*, 8(3):227–240, September 2018. (page 90)

[Dam90] Ivan Damgård. On the randomness of Legendre and Jacobi sequences. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 163–172. Springer, Heidelberg, August 1990. (page 2, 9, 11, 37)

[Dav33] H. Davenport. On the distribution of quadratic residues (mod p). *Journal of the London Mathematical Society*, s1-8(1):46–52, 1933. (page 10)

[DEG+22] Luca De Feo, Nadia El Mrabet, Aymeric Genêt, Novak Kaluđerović, Natacha Linard de Guertechin, Simon Pontié, and Élise Tasso. SIKE channels. Cryptology ePrint Archive, Report 2022/054, 2022. https://eprint.iacr.org/2022/054. (page 6, 81)

[DF17] Luca De Feo. Mathematics of isogeny based cryptography, 2017. (page 13)

[Div16] NIST Computer Security Division. Post-quantum cryptography standardization, 2016. (page 3)

[DJ11] Luca De Feo and David Jao. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, pages 19–34, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (page 3)

[DJP14] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209 – 247, 01 Sep. 2014. (page 3)

[DPN+16] Margaux Dugardin, Louiza Papachristodoulou, Zakaria Najm, Lejla Batina, Jean-Luc Danger, and Sylvain Guilley. Dismantling real-world ECC with horizontal and vertical template attacks. In François-Xavier Standaert and Elisabeth Oswald, editors, *COSADE 2016*, volume 9689 of *LNCS*, pages 88–108. Springer, Heidelberg, April 2016. (page 55, 61, 62, 81, 95)

[DR92] Deutsch David and Jozsa Richard. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society A*, 439:553—-558, 1992. (page 1)

[DZD+17] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, volume 10728 of *Lecture Notes in Computer Science*, pages 105–122. Springer, 2017. (page 101)

[Fei19] Dankard Feist. Legendre pseudo-random function, 2019. (page 3, 9, 37, 51)

[FGD+10] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures. In *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 76–87, 2010. (page 66, 67)

[FLOJRH18] Armando Faz-Hernández, Julio López, Eduardo Ochoa-Jiménez, and Francisco Rodríguez-Henríquez. A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol. *IEEE Transactions on Computers*, 67(11):1622–1636, 2018. (page 29, 86)

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, August 1999. (page 19)

[FS21] Paul Frixons and André Schrottenloher. Quantum security of the legendre PRF. Cryptology ePrint Archive, Report 2021/149, 2021. https://eprint.iacr.org/2021/149. (page 52)

[GdGK21] Aymeric Genêt, Natacha Linard de Guertechin, and Novak Kaluđerović. Full key recovery side-channel attack against ephemeral SIKE on the Cortex-M4. In Shivam

Bhasin and Fabrizio De Santis, editors, *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*, volume 12910 of *Lecture Notes in Computer Science*, pages 228–254. Springer, 2021. (page 6, 62, 63)

[GK22] Aymeric Genêt and Novak Kaluđerović. Single-trace clustering power analysis of the point-swapping procedure in the three point ladder of Cortex-M4 SIKE. In *Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings*, 2022. (page 6, 94)

[GLK21] Aymeric Genêt, Natacha Linard de Guertechin, and Novak Kaluđerović. Full key recovery side-channel attack against ephemeral SIKE on the Cortex-M4. In Fabrizio Bhasin, Shivam and De Santis, editor, *Constructive Side-Channel Analysis and Secure Design*, pages 228–254, Cham, 2021. Springer International Publishing. (page 69)

[GMP91] The GNU multiple precision arithmetic library, 1991. (page 80)

[Gou01] Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 3–15. Springer, Heidelberg, May 2001. (page 81)

[Gou03] Louis Goubin. A refined power-analysis attack on elliptic curve cryptosystems. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 199–210. Springer, Heidelberg, January 2003. (page 69)

[GPST16] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 63–91. Springer, Heidelberg, December 2016. (page 22)

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996. (page 2)

[GRR+16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 430–443. ACM Press, October 2016. (page 2, 9, 37)

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Heidelberg, November 2017. (page 22)

[Hol] ARM Holdings. Cortex-M4 specifications. https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4. (page 4)

[HPR07] Basil Hess, Geovandro Pereira, and Joost Renes. Efficient sidh library written in c language, 2107. (page 86)

[IBM09] Ibm q system one, 2109. (page 1)

[IT03] Tetsuya Izu and Tsuyoshi Takagi. Exceptional procedure attack on elliptic curve cryptosystems. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 224–239. Springer, Heidelberg, January 2003. (page 70)

[JAC+17] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Aaron Hutchinson, Amir Jalali, Koray Karabina, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular isogeny key encapsulation, 2017. https://sike.org/. (page 4, 25, 29, 63, 131)

[JMV09] David Jao, Stephen D. Miller, and Ramarathnam Venkatesan. Expander graphs based on grh with an application to elliptic curve cryptography. *Journal of Number Theory*, 129(6):1491–1504, Jun 2009. (page 18)

[JT01] Marc Joye and Christophe Tymen. Protections against differential analysis for elliptic curve cryptography. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 377–390. Springer, Heidelberg, May 2001. (page 67)

[KAJ17] Brian Koziel, Reza Azarderakhsh, and David Jao. Side-channel attacks on quantum-resistant supersingular isogeny Diffie-Hellman. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017*, volume 10719 of *LNCS*, pages 64–81. Springer, Heidelberg, August 2017. (page 55, 70, 82, 86, 87)

[Kal16] Gil Kalai. The quantum computer puzzle, 2016. (page 1)

[Kal19] Gil Kalai. The argument against quantum computers, 2019. https://arxiv.org/abs/1908.02499. (page 1)

[Kho19] Dmitry Khovratovich. Key recovery attacks on the Legendre PRFs within the birthday bound. Cryptology ePrint Archive, Report 2019/862, 2019. https://eprint.iacr.org/2019/862. (page 37, 48, 49, 50)

[KKK20a] Novak Kaluđerović, Thorsten Kleinjung, and Dušan Kostić. Cryptanalysis of the generalised Legendre pseudorandom function. volume 4, pages 267–282, 2020. (page 6)

[KKK20b] Novak Kaluđerović, Thorsten Kleinjung, and Dusan Kostic. Improved key recovery on the legendre PRF. Cryptology ePrint Archive, Report 2020/098, 2020. https://eprint.iacr.org/2020/098. (page 37, 50, 51)

[KPHS18] Philipp Koppermann, Eduard Pop, Johann Heyszl, and Georg Sigl. 18 seconds to key exchange: Limitations of supersingular isogeny Diffie-Hellman on embedded devices. Cryptology ePrint Archive, Report 2018/932, 2018. https://eprint.iacr.org/2018/932. (page 69)

[KPP20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on Keccak. *IACR TCHES*, 2020(3):243–268, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8590. (page 56)

[Kra86] Evangelos Kranakis. *Primality and Cryptography*. John Wiley & Sons, Inc., New York, NY, USA, 1986. (page 12)

[KRSS19] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. Workshop Record of the Second PQC Standardization Conference, 2019. https://cryptojedi.org/papers/#pqm4. (page 4)

[Kwi19] Kris Kwiatkowski. Towards post-quantum cryptography in TLS, 2019. https://blog.cloudflare.com/towards-post-quantum-cryptography-in-tls/. (page 4)

[Lan18] Adam Langley. Post-quantum confidentiality for TLS, 2018. https://www.imperialviolet.org/2018/04/11/pqconftls.html. (page 4)

[M+67] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967. (page 90)

[MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-enhanced power analysis collision attack. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 125–139. Springer, Heidelberg, August 2010. (page 82)

[MO09] Marcel Medwed and Elisabeth Oswald. Template attacks on ECDSA. In Kyo-Il Chung, Kiwook Sohn, and Moti Yung, editors, *WISA 08*, volume 5379 of *LNCS*, pages 14–27. Springer, Heidelberg, September 2009. (page 55)

[Mon87] Peter Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–264, 1987. (page 25, 26)

[Moo18] Dustin Moody. Let's get ready to rumble - The NIST PQC "competition", 2018. https://csrc.nist.gov/presentations/2018/let-s-get-ready-to-rumble-the-nist-pqc-competiti. (page 4)

[Moo19] Dustin Moody. Round 2 of the NIST PQC "competition" - What was NIST thinking?, 2019. https://csrc.nist.gov/presentations/2019/round-2-of-the-nist-pqc-competition-what-was-nist. (page 4)

[Nil91] A. Nilli. On the second eigenvalue of a graph. *Discrete Mathematics*, 91(2):207–210, 1991. (page 18)

[PCBP21] Guilherme Perin, Łukasz Chmielewski, Lejla Batina, and Stjepan Picek. Keep it unsupervised: Horizontal attacks meet deep learning. *IACR TCHES*, 2021(1):343–372, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8737. (page 98, 102)

[PITM14] Guilherme Perin, Laurent Imbert, Lionel Torres, and Philippe Maurine. Attacking randomized exponentiations using unsupervised learning. In Emmanuel Prouff, editor, *COSADE 2014*, volume 8622 of *LNCS*, pages 144–160. Springer, Heidelberg, April 2014. (page 93, 102)

[Piz90] Arnold K. Pizer. Ramanujan graphs and Hecke operators. *Bulletin (New Series) of the American Mathematical Society*, 23(1):127 – 137, 1990. (page 18)

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 513–533. Springer, Heidelberg, September 2017. (page 56)

[PST20] Christian Paquin, Douglas Stebila, and Goutam Tamvada. Benchmarking post-quantum cryptography in TLS. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 72–91. Springer, Heidelberg, 2020. (page 4)

[PZS17] Romain Poussier, Yuanyuan Zhou, and François-Xavier Standaert. A systematic approach to the side-channel analysis of ECC implementations with worst-case horizontal attacks. Cryptology ePrint Archive, Report 2017/629, 2017. https://eprint.iacr.org/2017/629. (page 55)

[Ren18] Joost Renes. Computing isogenies between Montgomery curves using the action of (0, 0). In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 229–247. Springer, Heidelberg, 2018. (page 75)

[RS04] Alexander Russell and Igor E. Shparlinski. Classical and quantum function reconstruction via character evaluation. *Journal of Complexity*, 20(2-3):404–422, 4 2004. (page 2, 9, 37)

[RS06] Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. https://eprint.iacr.org/2006/145. (page 3)

[SAJA20] Hwajeong Seo, Mila Anastasova, Amir Jalali, and Reza Azarderakhsh. Supersingular isogeny key encapsulation (SIKE) round 2 on ARM cortex-M4. Cryptology ePrint

Archive, Report 2020/410, 2020. https://eprint.iacr.org/2020/410. (page 4, 62, 94, 97, 111, 112, 113, 114)

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, Oct 1997. (page 2)

[Sil09] Joseph H Silverman. *The Arithmetic of Elliptic Curves*. Graduate texts in mathematics. Springer, Dordrecht, 2009. (page 13, 14, 15, 16, 17)

[Sim97] Daniel R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997. (page 1)

[SKL+20] Bo-Yeon Sim, Jihoon Kwon, Joohee Lee, Il-Ju Kim, Taeho Lee, Jaeseung Han, Hyo Jin Yoon, Jihoon Cho, and Dong-Guk Han. Single-trace attacks on message encoding in lattice-based KEMs. *IEEE Access*, 8:183175–183191, 2020. (page 56)

[SM11] Mutsuo Saito and Makoto Matsumoto. Tiny mersenne twister pseudo-random number generator, 2011. (page 100)

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 495–513. Springer, Heidelberg, September 2015. (page 81)

[SM16] Tobias Schneider and Amir Moradi. Leakage assessment methodology - extended version. *Journal of Cryptographic Engineering*, 6(2):85–99, June 2016. (page 100)

[ST15] Joseph H. Silverman and John T. Tate. *Rational Points on Elliptic Curves*. Springer Publishing Company, Incorporated, 2nd edition, 2015. (page 13)

[SWP03] Kai Schramm, Thomas J. Wollinger, and Christof Paar. A new class of collision attacks and its application to DES. In Thomas Johansson, editor, *FSE 2003*, volume 2887 of *LNCS*, pages 206–222. Springer, Heidelberg, February 2003. (page 82)

[van02] Dam van. Quantum algorithms for weighing matrices and quadratic residues. *Algorithmica*, 34(4):413–428, Nov 2002. (page 37)

[vDH00] Wim van Dam and Sean Hallgren. Efficient quantum algorithms for shifted quadratic character problems, 2000. (page 37)

[vDHI01] Wim van Dam, Sean Hallgren, and Lawrence Ip. Quantum algorithms for hidden coset problems. 2001. (page 37)

[Vél71] Jacques Vélu. Isogénies entre courbes elliptiques. *Comptes-Rendus de l'Académie des Sciences, Série I*, 273:238–241, 7 1971. (page 16)

[Wei48] André Weil. On some exponential sums. *Proceedings of the National Academy of Sciences*, 34(5):204–207, 1948. (page 11)

[Wei20] Alex Weibel. Round 2 hybrid post-quantum TLS benchmarks, 2020. https://aws.amazon.com/blogs/security/round-2-hybrid-post-quantum-tls-benchmarks/. (page 4)

[ZSP+18] Gustavo Zanon, Marcos A. Simplício Jr., Geovandro C. C. F. Pereira, Javad Doliskani, and Paulo S. L. M. Barreto. Faster isogeny-based compressed key agreement. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 248–268. Springer, Heidelberg, 2018. (page 4, 32)

[ZWMZ14] Zhenbin Zhang, Liji Wu, Zhaoli Mu, and Xiangmin Zhang. A novel template attack on wnaf algorithm of ECC. In *Tenth International Conference on Computational Intelligence and Security, CIS 2014, Kunming, Yunnan, China, November 15-16, 2014*, pages 671–675. IEEE Computer Society, 2014. (page 55)

[ZYD+20] Fan Zhang, Bolin Yang, Xiaofei Dong, Sylvain Guilley, Zhe Liu, Wei He, Fangguo Zhang, and Kui Ren. Side-channel analysis and countermeasure design on ARM-based quantum-resistant SIKE. *IEEE Transactions on Computers*, 69(11):1681–1693, 2020. (page 55, 66, 67, 69)

# NOVAK KALUĐEROVIĆ

Curriculum Vitae


**PERSONAL INFORMATION**

**Nationality** Serbian

**Date of birth** 1992 Aug 11

**Address** Room INF235 on EPFL campus

**CONTACT DETAILS**

+41 79 665 63 80

nk@kolja.rs

kolja.rs


## EDUCATION

**PhD, École polytechnique fédérale de Lausanne** **2017 – 2022**

*"Attacks on some post-quantum cryptographic protocols: The case of the Legendre PRF and SIKE"*

Thesis advisor: Arjen Lenstra, Serge Vaudenay Average grade 6,0/6,0

**M.S. 110 cum laude, Sapienza – Università di Roma** **2014 – 2016**

*"The Number Field Sieve"*

Thesis advisor: René Schoof Average grade 29,93/30

**B.S. 110 cum laude, Sapienza – Università di Roma** **2011 – 2014**

*"The Three Squares Theorem"*

Thesis advisor: Alessandro D'Andrea Average grade 28,95/30

## RESEARCH INTERESTS

My main interests lie in secure implementations of cryptographic protocols, side-channel attacks as well as defence thereof, and post-quantum cryptographic protocols. I am also fond of algebraic and computational number theory, both in the theoretical and implementation domain. I am passionate about coding in C, and working on low-level and high-efficiency implementations. Further interests include the theory of quantum computing and quantum algorithms.

## PUBLICATIONS

**SIKE channels: Zero-value side-channel attacks on SIKE** **2022**

A practical power-analysis attack on the isogeny computation in SIKE. TCHES 2022

**Single-trace clustering power analysis of the point swapping procedure in the three point ladder of Cortex-M4 SIKE** **2022**

A practical power-analysis attack of the point swap procedure in SIKE. COSADE 2022

**Full key recovery side-channel attack on ephemeral SIKE** **2021**

A practical power-analysis attack on the three point ladder in SIKE. COSADE 2021

**Cryptanalysis of the generalised Legendre pseudorandom function** **2020**

Analysis of a general algorithm for breaking the Legendre PRF. ANTS XIV

**Improved key recovery on the Legendre PRF** **2020**

Algorithms for breaking the Legendre pseudorandom function. e-print

## WORK EXPERIENCE

**Applied scientist intern** – "Protocols, Algorithms, Libraries" lab, AWS Cryptography, Seattle **2022**

Worked on reducing certificate sizes for lattice-based post-quantum public key certificates.

**Research intern** – "Algorithms, Randomisation, Computation" lab, Sapienza – Università di Roma **2016**

Worked on lower bounds for ordering problems and some open problems in computational complexity.

## TEACHING EXPERIENCE

**Computer science class for visiting students** **2018, 2019, 2021**

Lectured cryptography to visiting high school students from all around Switzerland.

**"Path of Excellence" seminar** **2016**

Held a seminar on the topic of primality testing algorithms under the supervision of prof. A. Panconesi.

**Student supervision** **2017 – 2021**

Personally supervised 11 Bachelor and 2 Master level research and implementation-oriented semester projects, and one successful Master thesis.

**Teaching** **2017 – 2021**

Served as a teaching assistant and replacement lecturer for the following Master and Bachelor courses:

| | | |
|---|---|---|
| **CS-450:** | Advanced algorithms | **2019** |
| **CS-101:** | Advanced information, computation, communication I | **2018, 2019, 2021** |
| **COM-102:** | Advanced information, computation, communication II | **2018, 2020** |
| **MATH-111:** | Linear Algebra | **2020** |

## HONOURS AND AWARDS

**Teaching Assistant Award** **2018, 2019**

An award given by EPFL to the best teaching assistants.

**EDIC Fellowship for Doctoral Studies** **2017**

One year fellowship for prospective doctoral students at EPFL.

**Montenegrin national scholarship for excellence** **2014, 2016**

A scholarship awarded by the Montenegrin ministry of education to its highest achieving students in foreign universities.

**Laziodisu scholarship** **2013, 2014, 2015, 2016**

Scholarship providing funding and full tuition waiver to high achieving students during their Bachelors and Masters degree studies awarded by the Lazio region.

## OTHER

**Server administrator** **2018 – 2022**

Responsible for the administration of TRX and Unicorn — random elliptic curve and random number beacons on trx.epfl.ch.

**Paper reviewing** **2018 – 2022**

Reviewed multiple papers for Number-Theoretic Methods in Cryptology, Journal of Mathematical Cryptology, MathCrypt, Journal of Experimental Algorithmics, COSADE, Asiacrypt, Eurocrypt, PKC.

## LANGUAGES

| | |
|---|---|
| **Serbian** | Native |
| **English** | Fluent |
| **Italian** | Fluent |
| **French** | Basic communication skills |

## PROGRAMMING LANGUAGES

C, Python, Sage