EPFL

Thèse n° 8258

Time in cryptography

Présentée le 18 septembre 2020

à la 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

Gwangbae CHOI

Acceptée sur proposition du jury

Prof. A. Lenstra, président du jury
Prof. S. Vaudenay, directeur de thèse
Prof. A. Miyaji, rapporteuse
Prof. T. Jager, rapporteur
Prof. O. Svensson, rapporteur

École
polytechnique
fédérale
de Lausanne

2020

To my family

Acknowledgements

First of all, I would like to acknowledge the role my supervisor Prof. Serge Vaudenay played over these past few years. He gave me the chance to pursue a Ph.D. in cryptography, and I could not have finished it without his guidance. During these years, I also learned a lot from him, especially how to conduct research, and how to properly engage in mathematical formalism. I am convinced, that without his guidance I would not have been able to graduate with a degree in cryptography.
Next, I would like to thank Prof. Tibor Jager, Prof. Atsuko Miyaji and Prof. Ola Svensson, and the jury president Prof. Arjen Lenstra for agreeing to be in the jury, and reading this dissertation and giving valuable comments for improvements.
I gratefully acknowledge the support by Swiss National Science Foundation under project funding number 169110.
I also want to thank all former and current members of LASEC: Prof. Serge Vaudenay, Martine Corval, Dr. Philippe Oechslin, Dr. Betül Durak, Dr. Iraklis Leontiadis, Dr. Alexandre Duc, Dr. Sonia Bogos, Dr. Damian Vizár, Dr. Handan Kılınç, Dr. Subhadeep Banik, Dr. Hailun Yan, Fatih Balli, Khashayar Barooti, Andrea Caforio, Daniel Collins, Loïs Huguenin, and Bénédikt Tran. Thanks to all these people, I enjoyed working in LASEC. I especially thank Betül who gave me the opportunity to work together on the interesting topic and Martine for all the logistics that she had to handle because of me.
I would like to send a big thanks to my father Jongbeom Choi and my mother Hyunhee Gong. They supported me for my entire student life and encouraged me when I was demotivated. My next thanks go to my brother Hyunbae Choi and sister-in-law Youngmi Lee who have always supported me. I also thank my grandfather Seongsu Choi and my grandmother Sangsun Kim who gave me the motivation to continue studying.
In the end, I would like to thank all my friends who studied together during my bachelor's and master's courses in EPFL.

Lausanne, July 3, 2020 Gwangbae Choi

Abstract

Time travel has always been a fascinating topic in literature and physics. In cryptography, one may wonder how to keep data confidential for some time. In this dissertation, we will study how to make private information travel to the future. This dissertation consists of three parts: Timed-release encryption, witness encryption and self-encryption.

With timed-release encryption, one can send to a counterpart a message which cannot be read before some time has elapsed. One possible solution is to use a third party. At every time period, the third party releases a time bound key, and a ciphertext which is encrypted for a time period requires the time bound key of that time period for decryption. Then, a problem occurs when the counterpart somehow loses the release time of ciphertext. We propose a solution by introducing a master time bound key which can be considered as a valid time bound key of all time periods. We propose a provably secure construction and show the experimental results.

In 2018, Liu, Jager, Kakvi and Warinschi introduced a timed-release encryption scheme based on a blockchain with proof-of-work and witness encryption. Current proposals of witness encryption are based on multilinear maps. In this part, we propose a new construction without. We propose the notion of hidden group with hashing and make a witness encryption from it. We show that the construction is secure in a generic model. We propose a concrete construction based on RSA-related problems. Namely, we use an extension of the knowledge-of-exponent assumption and the order problem. We finally estimate the cost of the bitcoin blockchain implementation. Although our estimates are still high (for a release time of one hour / one year, we respectively use ciphertexts of 567 MB / 4.5 TB and a decryption time of 27 min / 5.2 months on a single core), there is room for improvement by a factor 20 000 by adapting the blockchain structure and by adopting a hash function which is better adapted to this type of programming than SHA256.

In self-encryption, a device encrypts some piece of information for itself to decrypt in the future. We are interested in security of self-encryption when the state occasionally leaks. Applications where self-encryption operates are cloud storage, when a client encrypts files to be stored, and in 0-RTT session resumptions, when a server encrypts a resumption key to be stored by the client. Previous works focused on forward secrecy and resistance to replay attacks. In our work, we study post-compromise security. Post-compromise security was already solved in ratcheted instant messaging schemes, at the price of having an inflating state size. However, it was not known whether state inflation was necessary.

Abstract

We prove here that this is the case.
In our results, we prove that post-compromise security implies a super-linear state size in terms of the number of ciphertexts which can still be decrypted by the state. We apply our result to self-encryption for cloud storage and 0-RTT session resumption, and also to secure messaging. We further show how to construct a secure scheme matching our bound on the state size.

Résumé

Le voyage dans le temps a toujours été un sujet fascinant, que ce soit en littérature ou en physique. De même, en cryptographie, nous nous demandons comment conserver des données confidentielles pendant un certain temps. Dans cette thèse, nous étudions comment faire voyager des informations privées vers le futur. Ce travail est divisé en trois parties : le chiffrement temporisé (timed-release encryption), le chiffrement de témoin (witness encryption) et l'auto-chiffrement (self-encryption).

Le chiffrement temporisé permet d'envoyer un message qui ne pourra pas être lu avant un certain temps. Une solution possible consiste à utiliser une tierce partie. Plus précisément, à chaque période de temps la tierce partie génère une clé temporelle ; cette clé est ensuite nécessaire au déchiffrement des textes chiffrés pour ladite période de temps. Cependant, un problème se produit quand le destinataire du message chiffré oublie ou perd le temps de relâche (de déchiffrement) de celui-ci. Nous proposons une solution en introduisant une clé temporelle maîtresse (master time bound key), qui peut être considérée comme une clé temporelle valable pour toutes les périodes. Nous prouvons ensuite formellement la sécurité de notre construction et montrons les résultats expérimentaux obtenus.

En 2018, Liu, Jager, Kakvi et Warinschi ont introduit un chiffrement temporisé basé sur une blockchain avec une preuve de travail et un chiffrement de témoin. Les propositions actuelles de chiffrement de témoin sont basées sur des fonctions multilinéaires. Dans cette partie, nous proposons la notion de groupe caché avec hachage. Puis, en nous basant sur ce concept, nous présentons une nouvelle construction de chiffrement de témoin qui n'utilise pas de fonctions multilinéaires. Nous montrons ensuite que la construction est sécurisée dans un modèle générique. Plus précisément, notre construction est basée sur des problèmes liés à RSA, comme une variante de l'hypothèse de connaisance-de-l'exposant et du problème d'ordre. Enfin, nous estimons la performance de notre proposition quand elle est implémentée avec la blockchain bitcoin. Bien que nos estimations soient encore élevées (pour un temps d'une heure / un an, nous utilisons respectivement des textes chiffrés de 567 Mo / 4, 5 To et un temps de déchiffrement de 27 min / 5, 2 mois sur un seul processeur), une amélioration d'un facteur de 20 000 est possible en adaptant la structure de la blockchain, ainsi qu'en adoptant une fonction de hachage mieux adaptée à ce type de programmation que SHA256.

En auto-chiffrement, un appareil chiffre certaines informations destinées à lui-même et à déchiffrer dans le futur. Nous nous sommes intéressés à la sécurité de l'auto-chiffrement en présence de fuites d'information occasionnelles. Typiquement, l'auto-chiffrement est utilisé

quand un client souhaite chiffrer des informations afin de les stocker de façon sécurisée sur un serveur distant, ou quand un serveur chiffre une clé de reprise qui sera conservée par le client dans une session 0-RTT. Les travaux précédents se sont concentrés sur la confidentialité persistante et la résistance contre des attaques par rejeu. Dans notre travail, nous étudions la sécurité en présence de corruption (sécurité post-compromission). Dans les systèmes de messagerie instantanée à cliquet (ratcheting), la sécurité post-compromission est garantie au prix d'une taille d'état qui augmente. Cependant, savoir si cette inflation de l'état est nécessaire restait un problème en suspens. Nous prouvons ici que c'est le cas. Plus précisément, nous montrons que la sécurité en présence de corruption implique une taille d'état super-linéaire en terme du nombre de textes chiffrés qui peuvent encore être déchiffrés par l'état. Nous appliquons notre résultat à l'auto-chiffrement destiné au stockage d'informations sur des serveurs distants et à la reprise de session 0-RTT, ainsi qu'aux systèmes de messagerie sécurisée. Nous montrons en outre comment construire un système sécurisé atteignant notre borne sur la taille de l'état.

Contents

1 Introduction

Cryptography is the science of information security. In order to construct a *secure* cryptosystem, we need cryptography. Then, how can we assess if a cryptosystem is secure? We usually assess it by the hardness of attacks whose time complexity is somehow bounded. Namely, λ-bit security refers to the hardness of attacks whose time complexity is at least 2^{λ}. The notion of *time* is therefore an important element in cryptography.

The notion of *time* is not only a metric to assess the security but also a metric to evaluate the performance of cryptosystem. When we are evaluating the performance, there are numerous ways to evaluate it depending on the purpose. When a cryptosystem is used for security, encryption eventually slows down the entire processing. The execution time is therefore a metric which is considered in many cases. Cryptographic protocols sometimes deploy some techniques to decrease the number of exchanged messages between two participants in order to decrease the execution time of the protocol. One well-known example is the semi-static Diffie-Hellman key exchange. Instead of using an ephemeral key, a server uses a static key, and a client generates a shared key from the static key without communicating with the server. So, the client can send the first message, which is encrypted by using the shared key, and his/her ephemeral public key to the server at the time. Therefore, the server and the client save a round-trip time in their communication. Another example is the zero round-trip time (0-RTT) key exchange [GHJL17, DJSS18, AGJ19]. The 0-RTT key exchange does not require any communication between a server and a client to establish a shared key and it eventually makes entire communication faster.

When one wants to send to a counterpart a message which should not be read before some time, the best way is to send the message at the time when it can be. However, this can only be done if the sender is available at the time when the message can be read. We now assume that the sender is not available to communicate with the counterpart at the time when the message should be read. One of the easiest solutions is to use a trusted third party which transfers the message at the desired time, but we would ideally prefer

requiring no third party, or at least imposing no specific storage on the third party side.

The time-lock puzzle [RSW96, BGJ+16, MT19] can generate a puzzle which requires the specific amount of computations to retrieve the data in it. By generating a time-lock puzzle which contains a decryption key, we can construct a timed-release encryption scheme which requires the specific amount of computations to decrypt a ciphertext. However, as we can only control the amount of computations, this approach of timed-release encryption cannot fully control the decryption time which will depend on the computational capacity of one who decrypts. There exists another approach of the timed-release encryption which tries to control the time when a ciphertext can be decrypted. By using a third party which periodically releases some information, we can construct a time-release encryption scheme which uses such information to decrypt the ciphertext.

The time that we want to control is not only the decryption time but also the arrival time of new input to the system. As the blockchain-based system does not want to be flooded by the arrival of new blocks, it demands some additional computations for each block to be added to the blockchain. For example, Bitcoin [Nak19], which is one of the most famous system based on a blockchain, requires the hash value of a block to have some leading zeros in order for the block to be added in the blockchain.

Depending on applications, we sometimes need the ability to send a secret to ourself in the future. When a server and a client require some data for future communications, one efficient solution is to make the server encrypt the data and make the client to keep the ciphertext for future communications. So, the server actually encrypts some data for itself in the future. Since the encryption and the decryption are done by the same entity and the server does not know when a ciphertext will return, it will cause different security issues.

In this dissertation, we tackle the problem of sending secrets to the future and bring contributions to these methods.

Outline of Thesis

In Chapter 2, we present the notations that we use through this dissertation and some definitions that we use. We give the following definitions:

- Weil pairing
- Bilinear Diffie-Hellman problem
- NP language
- Subset sum problem

In Chapter 3, we focus on the timed-release encryption. In this chapter, we tackle one potential problem of timed-release encryption: the impossibility of decryption when the release time is lost. We first formally define primitives and security definitions of the timed-release encryption. Then, we propose a construction of timed-release encryption, and analyze its security. Moreover, we implement our construction and show the experimental results.

In Chapter 4, we study another approach of timed-release encryption. Recently, Liu et al. [LJKW18] proposed a timed-release encryption scheme from a witness encryption scheme and the Bitcoin blockchain. In this chapter, we propose the notion of a hidden group with hashing (HiGH) and construct a witness key encapsulation model, which can be easily extended to a witness encryption scheme, on top of HiGH. Moreover, we further analyze practical aspects of the timed-release encryption scheme based on a witness encryption scheme and the Bitcoin blockchain.

In Chapter 5, we study a different problem related to sending secrets for the future: sending a secret to ourself. There exist two different resiliency notions depending on the time corruption happens: forward secrecy and post-compromise security. We first define self-encryption and prove that the state size must super-linearly increase in order to achieve forward secrecy and post-compromise security at the same time. Moreover, we apply our result to the self-ratcheted scheme and the bipartite ratcheted scheme.

We conclude in Chapter 6, and introduce some possible research directions.

Personal Bibliography

This dissertation is based on the following papers:

- **Gwangbae Choi** and Serge Vaudenay. *Timed-release encryption with master time bound key.* In WISA 2019
- **Gwangbae Choi**, F. Betül Durak, and Serge Vaudenay. *Post-compromise security in self-encryption.* (Under submission)
- **Gwangbae Choi** and Serge Vaudenay. *Sending secrets to the future - witness encryption without multilinear maps.* (Under submission)

2 Preliminaries

In this chapter, we will show the notations and definitions that we will use through this dissertation.

2.1 Notations

When E is an elliptic curve and K is a field, $E(K)$ represents a group of points on the elliptic curve E, including the point at infinity, whose x-coordinate and y-coordinate are defined in K. We denote a concatenation of two bit strings a and b as $a||b$ and an empty input or output by $\perp$. We write $x \xleftarrow{\$} G$ if x is uniformly chosen from a set G. We denote an empty string or algorithm by ε. For any probabilistic algorithm $f(x)$, we denote an instance of the algorithm $f(x)$ with a sequence of random coins γ as $f(x;\gamma)$. For any g in some group G, a subgroup generated by g is written as $\langle g \rangle$. Let $X : \Omega \to S$ and $Y : \Omega \to S$ be two random variables. Then, the statistical distance between two random variables X and Y is $d(X,Y) = \frac{1}{2}\sum_{s\in S} |\Pr[X = s] - \Pr[Y = s]|$. We denote the uniform distribution over a set G by $\mathcal{U}_G$. We denote the indicator function by $\mathbb{1}_r$. We consider "words" as bitstrings (i.e. we use a binary alphabet) and $|x|$ denotes the bit length of x. 1^a is the bitstring of length a with all bits set to 1. $\#S$ denotes the cardinality of the set S. $\mathsf{negl}(\lambda)$ denotes any function f such that for all $c > 0$, for any sufficiently large λ, we have $|f(\lambda)| < \frac{1}{\lambda^c}$, and we also say that the function f is *negligible*. Similarly, $\mathsf{Poly}(\lambda)$ denotes any function f such that there exists $c > 0$ such that for any sufficiently large λ, we have $|f(\lambda)| < \lambda^c$.

2.2 Weil Pairing

A pairing is a bilinear map which maps two groups into a third group whose cardinalities are equal. In cryptography, a pairing can be used to construct cryptosystems [BF03, BB04, DT07, CHKO06, CLQ05, HYL05, BC04, CHS07] and it can even be used for

attack [MOV93]. The Weil pairing is an instantiation of pairing. We use the definition of the Weil pairing from Silverman [Sil09, III.8.1]. Let K be a finite field and E be an elliptic curve over K. The Weil pairing $e : E[m] \times E[m] \longrightarrow \mu_m$, where $E[m]$ is the m-torsion subgroup of E and μ_m is the group of the m-th roots of unity in the algebraic closure $\bar{K}$, satisfies the following properties.

1. Bilinear: $\forall P_1, P_2, Q_1, Q_2 \in E[m]$, $e(P_1+P_2, Q_1) = e(P_1, Q_1)e(P_2, Q_1)$ and $e(P_1, Q_1+Q_2) = e(P_1, Q_1)e(P_1, Q_2)$.
2. Non-degenerate: $\forall P \in E[m], \exists Q \in E[m]$ such that $e(P, Q) \neq 1$.
3. Alternating: $\forall P \in E[m], e(P, P) = 1$.
4. Galois invariant: $\forall \sigma \in G_{\bar{K}/K}, e(P^\sigma, Q^\sigma) = e(P, Q)^\sigma$.

We note that the Weil pairing can be efficiently computed by the Miller's algorithm [M$^+$86].

2.3 Bilinear Diffie-Hellman Problem

The Diffie-Hellman problem is one of most famous problems in cryptography, and the bilinear Diffie-Hellman problem is a problem which is similar to the Diffie-Hellman problem on the groups where a bilinear map exist. We define two variations of the bilinear Diffie-Hellman problem as follows.

Definition 1 (Decisional bilinear Diffie-Hellman problem [BF03])**.** *Let* $\mathsf{Gen}(1^\lambda) = \pi = (\lambda, K, E, m, e)$ *be an algorithm which generates an appropriate instance of the decisional bilinear Diffie-Hellman problem, given the security parameter* λ*, where* K *is a field,* E *is an elliptic curve over* K*, and* $e : E[m] \times E[m] \longrightarrow \mu_m$ *is a bilinear map.*

We say that the decisional bilinear Diffie-Hellman problem is hard for Gen *if*

$$\mathsf{Adv}^{\mathsf{DBDH}}_{\mathcal{A}}(\lambda) = \left|\Pr\left[\mathsf{DBDH\text{-}0}^{\mathcal{A}}_{\mathsf{Gen}}(\lambda) \to 1\right] - \Pr\left[\mathsf{DBDH\text{-}1}^{\mathcal{A}}_{\mathsf{Gen}}(\lambda) \to 1\right]\right|$$

is a negligible function in λ *for all probabilistic and polynomial time algorithm* $\mathcal{A}$ *where* $\mathsf{DBDH\text{-}d}$ *is defined as follows for* $d \in \{0, 1\}$*.*

Game $\mathsf{DBDH\text{-}d}^{\mathcal{A}}_{\mathsf{Gen}}(\lambda)$

1: $\pi \leftarrow \mathsf{Gen}(1^\lambda)$
2: $(a_0, b_0, c_0) \xleftarrow{\$} \mathbb{Z}_m^3$
3: $(a_1, b_1, c_1) \xleftarrow{\$} \mathbb{Z}_m^3$
4: $(P, Q) \xleftarrow{\$} E[m] \times E[m]$
5: $d' \leftarrow \mathcal{A}(\pi, P, Q, a_0P, b_0P, c_0P, a_0Q, b_0Q, c_0Q, e(P, Q)^{a_d b_d c_d})$
6: ***return*** d'

Definition 2 (Computational bilinear Diffie-Hellman problem [BF03]). *Let* $\mathsf{Gen}(1^\lambda) = \pi = (\lambda, K, E, m, e)$ *be an algorithm which generates an appropriate instance of the computational bilinear Diffie-Hellman problem, given the security parameter* λ*, where* K *is a field,* E *is an elliptic curve over* K*, and* $e : E[m] \times E[m] \longrightarrow \mu_m$ *is a bilinear map.*

We say that the computational bilinear Diffie-Hellman problem is hard for Gen *if*

$$\mathit{Adv}_{\mathcal{A}}^{\mathit{CBDH}}(\lambda) = \Pr\left[\mathit{CBDH}_{\mathsf{Gen}}^{\mathcal{A}}(\lambda) \to 1\right]$$

is a negligible function in λ *for all probabilistic and polynomial time algorithm* $\mathcal{A}$ *where* CBDH *is defined as follows.*

Game $\mathit{CBDH}_{\mathsf{Gen}}^{\mathcal{A}}(\lambda)$

1: $\pi \leftarrow \mathsf{Gen}(1^\lambda)$
2: $(a, b, c) \xleftarrow{\$} \mathbb{Z}_m^3$
3: $(P, Q) \xleftarrow{\$} E[m] \times E[m]$
4: $Z \leftarrow \mathcal{A}(\pi, P, Q, aP, bP, cP, aQ, bQ, cQ)$
5: ***return*** *whether* $Z = e(P, Q)^{abc}$

2.4 NP Language

In cryptography, we usually need a hard problem to make a system secure, and a hard problem is usually in an NP language. We formally define NP language and the NP-completeness as follows.

Definition 3 (NP language). *Let* L *be a language. The language* L *is in the class* NP *if there exists a predicate* R *and a polynomial* P *such that* L *is the set of all words* x *for which there exists a witness* ω *satisfying* $R(x, \omega)$ *and* $|\omega| \leq P(|x|)$*, and if we can compute* R *in time polynomially bounded in terms of the size of* x*.*

Definition 4 (NP-completeness). *A language* L *is* NP*-complete if it belongs to the class* NP *and for any language* L' *in* NP*, there exists a polynomially bounded deterministic algorithm* Encode-Inst *such that for all* x*,* $x \in L \iff \mathsf{Encode\text{-}Inst}(x) \in L'$*.*

2.5 Subset Sum Problem

The subset sum problem is an NP problem. It is well-known that the subset sum problem is NP-complete [Kar72]. Intuitively, the subset sum problem is a problem of finding a subset of a given set of integers whose sum is equal to a target value. The Subset Sum (SS) NP language is defined by:

Instance: a tuple $x = (x_1, \ldots, x_t, s)$ of non-negative integers.

Witness: a tuple $\omega = (a_1, \ldots, a_t)$ of bits $a_i \in \{0, 1\}$, $i = 1, \ldots, t$.

Predicate $R(x, \omega)$: $a_1 x_1 + \cdots + a_t x_t = s$.

3 Time-Based Cryptography

Consider a user who holds a secret. Suppose that he must commit to disclosing his secret at some given time in the future. A requirement is that no one should learn the secret earlier and the protocol should complete in the future even though the user may not be present. For instance, the secret could be the wills of the user and the user could be deceased at the time of release. This is what timed-release encryption aims at.

The concept of timed-release encryption was first proposed by May [May93]. The idea is to introduce the concept of time into an encryption scheme, especially into the decryption algorithm. There are two distinct approaches. One is to focus on the amount of time it takes to decrypt and the other is to have a third party to unlock encryption in due time. It finds applications [RSW96] in sealing a bid in auction, issuing mortgage electronic payments to be made in the future, encrypting a diary to be released after 50 years, or even enforcing key escrow after a legal period. We could also imagine to commit on a secret value so that the hiding property would fade by itself, to automatically declassify top secret documents, or to enforce a *true* key expiration by releasing the keys after expiration. When used in signatures, this would make sure that (non-registered) signatures become deniable after expiration of their keys.

The first category of timed-release encryptions uses time-lock puzzles [RSW96], which involves heavy computation for the decryption. The second one involves a third party [COR99, BC04, CLQ05, CHKO06, HYL05, CHS07]. It requires a time bound key which is periodically released by the trusted server for the decryption.

The timed-release encryption with a time-lock puzzle was first introduced by Rivest et al. [RSW96] in 1996. The concept of *puzzle* was introduced by Merkle [Mer78] in 1978. However, Rivest et al. showed that the Merkle's puzzle is not suitable for a time-lock puzzle as it is parallelizable, so it offers no guarantee on the amount of time required to decrypt. Rivest et al. proposed a way to achieve timed-release encryption based on RSA. The user generates an RSA modulus n, picks r and uses the key $r^{2^t \bmod \lambda(n)} \bmod n$ to encrypt the secret. This can be done with $\mathcal{O}(\log t + \log n)$ modular multiplications

using the secret factors of n. The ciphertext consists of n, r, and the encryption. To decrypt, anyone could compute $r^{2^t} \bmod n$ in t modular squares. As the ciphertext can be generated without any input from the counterpart, a single ciphertext can be sent to multiple users, and all of them can correctly decrypt it after some sequential computations. However, this solution is a bit unfair because receivers with lower computational power take more time than powerful receivers, and the time of release is not an absolute time.

Solving timed-release encryption with a trusted third party is essentially trivial: one can share the secret with the third party (a notary) and let him disclose it in the future. We can also solve timed-release encryption with a *semi-trusted* third party who works as a beacon: it releases a decryption token at every time period and the encryption encodes the time of release. It is possible to combine this idea with encryption to a target receiver so that the honest-but-curious beacon is not able to decrypt, without colluding with the target receiver. From now on, we call this beacon as *trusted server*. The solutions based on trusted server were thoroughly studied in many papers. Rivest et al. [RSW96] proposed a construction in which the trusted server does not store any message but this scheme suffers from problems of anonymity and confidentiality. Crescenzo et al. [COR99] proposed a construction based on a conditional oblivious transfer which allows a sender to be anonymous. But the receiver cannot be anonymous and the trusted server is a subject to denial-of-service attack. Later, Blake and Chan [BC04] proposed a construction based on the identity-based encryption scheme by Boneh and Franklin [BF03] in which the trusted server interacts with neither the sender nor the receiver. As Blake and Chan did not provide any security notion, Cathalo et al. [CLQ05] proposed its security notions and improved its construction. Based on the construction of Blake and Chan, Hwang et al. [HYL05] proposed a construction with pre-open capability which allows a receiver to decrypt before the release time by using the pre-open key. As security analysis of this construction was not sufficient, Dent and Tang [DT07] introduced additional security models for the construction of Hwang et al. On the other hand, Cheon et al. [CHKO06] proposed a construction of authenticated timed-release encryption. Later, Chalkias et al. proposed a more efficient timed-release encryption scheme [CHS07]. In 2009, Nakai et al. [NMKM09] proposed a generic construction of the timed-release encryption with pre-open capability by using an identity-based encryption and a public key encryption. Their generic construction was improved by Matsuda et al. [MNM10] in terms of efficiency. In 2010, Paterson et al. [PQ10] proposed the time-specific encryption paradigm. In time-specific encryption, a ciphertext can only be decrypted during a chosen time interval rather than after a chosen time. Therefore, the time-specific encryption can be seen as the generalization of the timed-release encryption. Later, Kasamatsu et al. [KME+16] showed how the time-specific encryption can be derived from forward-secure encryption.

The approach with a time-lock puzzle does not require any trusted server, but the sender does not have the full control on the release time of the encrypted message as it depends on the computational power of the receiver and the time it started to decrypt. With the approach which uses a trusted server, the release time can be fully controlled by the

sender as it requires a time bound key which will be released by the trusted server at the release time. However, for the protocol to work, it is necessary to include a trusted server and thus it may lead to security vulnerabilities due to the addition of another participant in the protocol.

In this chapter, we focus on the approach with a trusted server and we will study another potential problem which was not considered in previous works. In the previous works, the release time was usually somehow known to the receiver and the receiver could execute the decryption algorithm with the time bound key of the corresponding release time. Then, what happens if the receiver loses the release time? The receiver obviously cannot deduce which time bound key should be used for the decryption. The receiver therefore cannot correctly decrypt the ciphertext because the time bound key of the release time is required for the decryption. In existing constructions, a ciphertext can be separated into two parts: encrypted message and release time. As the release time cannot be extracted from the encrypted message part, a ciphertext becomes undecryptable if the release time part is somehow corrupted. By introducing the master time bound key, we can avoid the such problem. However, the probability that only the release time is corrupted might be low in the real world. But by the nature of the timed-release encryption, the time that the receiver waited becomes useless along with the loss of the release time. Therefore, the introduction of such functionality might be meaningful for the receiver.

There already exist some easy ways to solve this problem. The sender can for example store the release time after the encryption, and send it again when the receiver asks the release time. This approach however cannot be an actual solution of the problem since an intuitive goal of timed-release encryption is to send a message for the time period when the sender and the receiver do not communicate. Another approach which does not require any communication between the sender and the receiver is to make the receiver to decrypt with all time bound keys. This solution however requires too much computation, compared to the normal decryption, and the receiver requires a way to check the correctness of the decrypted message.

The constructions with pre-open capability [HYL05] might be a solution for the problem of losing the release time by giving the pre-open key which allows the decryption without the time bound key. The sender however needs to know the release time of the ciphertext to generate the corresponding pre-open key, it is equivalent to store the release time on the sender side. If the sender is storing the release time of the ciphertext, the sender can simply resend the release time to the receiver. The problem therefore becomes trivial. We hence consider the case neither the sender nor the receiver knows the release time.

Our Contribution and Structure

In this chapter, we propose a better solution on this problem. We introduce a master time bound key which can be used as a valid time bound key for any release time. The receiver therefore can ask to the trusted server to decrypt a ciphertext of an unknown release time. This however can raise another problem with confidentiality of the message if the receiver needs to send the entire ciphertext to the trusted server for the decryption with master time bound time bound key. Our solution also solves this problem. A ciphertext of our construction consists of three elements. The receiver needs to send a single element to the trusted server to do the computation with master time bound key. As this element is independent from the message, the trusted server cannot learn anything about the message.

The master time bound key moreover can be used when the trusted server terminates its service. As a time bound key of the release time is needed for the decryption, the ciphertext whose release time is after the termination of the trusted server can never be decrypted. If it is more important not to lose the message than being decrypted before its release time, the trusted server needs to reveal its secret key or all future time bound keys to make the users able to decrypt their ciphertexts. If the trusted server reveals its secret key, receivers must implement another decryption algorithm which decrypts with the trusted server secret key instead of a time bound key. If the trusted server generates all future time bound keys (and possibly encrypt them with a timed-release encryption of another server), there might have a problem with the storage complexity if the amount of remaining time periods is huge. All of these solutions therefore require some additional works. However, if the trusted server has the master time bound key, it is enough if the trusted server releases the master time bound key at the end of its service. Moreover, the storage overhead is minimized since the size of master time bound key is equal to the size of time bound key and the trusted server does not need to generate a bunch of time bound keys for future time periods.

Finally, our master time bound key can play the role of a backup solution to decrypt messages in emergency situations (e.g. sudden disappearing of the trusted server).

In this chapter, we propose a timed-release encryption scheme which has the master time bound key that can be used to decrypt a ciphertext of any time period. In Section 3.1, we define primitives of timed-release encryption, and we then define its security models in Section 3.2. In Section 3.3, we propose a construction of timed-release encryption scheme with master time bound key and analyze its security with the security models that we defined. In Section 3.4, we will propose appropriate parameters depending on the security level, and then we will show the experimental results of our construction for different security levels in Section 3.5.

3.1 Primitives of Timed-Release Encryption With Master Time Bound Key

In this section, we formally define the primitives of timed-release encryption with master time bound key. Our primitives are similar to the primitives in literatures [DT07, CHKO06, CLQ05, HYL05, BC04]. The difference however is the key generation algorithm of the trusted server outputs the master time bound key along with the secret key and the public key.

Let S be a sender, R be a receiver and TS be a trusted server. We define a timed-release encryption scheme with master time bound key as follows:

Definition 5 (Timed-release encryption scheme with master time bound key)**.** *A timed-release encryption scheme consists of the following algorithms:*

- $\mathsf{Setup}(1^\lambda) = \pi$ *is a probabilistic polynomial time algorithm which generates a system parameter* π *given a security parameter* λ*.*
- $\mathsf{KeyGen}_{TS}(\pi) = (\mathsf{sk}_{TS}, \mathsf{pk}_{TS}, \mathsf{mk}_{TS})$ *is a probabilistic polynomial time algorithm of the trusted server* TS *which takes a system parameter* π*, and generates a secret key* sk_{TS}*, a public key of the trusted server* pk_{TS} *and a master time bound key* mk_{TS}*.*
- $\mathsf{KeyGen}_R(\pi) = (\mathsf{sk}_R, \mathsf{pk}_R)$ *is a probabilistic polynomial time algorithm of the receiver* R *which takes a system parameter* π*, and generates a secret key* sk_R *and a public key of the receiver* pk_R*.*
- $\mathsf{Broadcast}(\mathsf{sk}_{TS}, t, \pi) = \tau_t$ *is a probabilistic polynomial time algorithm of the trusted server* TS *which takes a secret key of the trusted server* pk_{TS}*, scheduled broadcast time* t *and a system parameter* π*, and broadcasts time bound key* τ_t*.*
- $\mathsf{Enc}(\mathsf{pk}_{TS}, \mathsf{pk}_R, m, t, \pi) = c$ *is a probabilistic polynomial time algorithm of the sender* S *which takes a trusted server public key* pk_{TS}*, a receiver public key* pk_R*, a message* m*, release time* t*, and a system parameter* π*, and outputs a ciphertext* ct*.*
- $\mathsf{Dec}(\mathsf{sk}_R, \tau_t, \mathsf{ct}, \pi) = m$ *is a deterministic polynomial time algorithm of the receiver* R *which takes a receiver secret key* sk_R*, a time bound key at the release time* t τ_t*, a ciphertext* ct*, and a system parameter* π*, and outputs a message* m *or* $\perp$*.*

Then, we expect a timed-release encryption scheme to satisfy the following condition:

- *For any security parameter* λ*, for any system parameter* $\pi = \mathsf{Setup}(1^\lambda)$*, for any trusted server key pair* $(\mathsf{sk}_{TS}, \mathsf{pk}_{TS}, \mathsf{mk}_{TS}) = \mathsf{KeyGen}_{TS}(\pi)$*, for any receiver key pair* $(\mathsf{sk}_R, \mathsf{pk}_R) = \mathsf{KeyGen}_R(\pi)$*, for any message* m *and for any time period* t*,*

$$\Pr_{\gamma_1,\gamma_2}[\mathsf{Dec}(\mathsf{sk}_R, \mathsf{Broadcast}(\mathsf{sk}_{TS}, t, \pi; \gamma_1), \mathsf{Enc}(\mathsf{pk}_{TS}, \mathsf{pk}_R, m, t, \pi; \gamma_2), \pi) = m] = 1$$

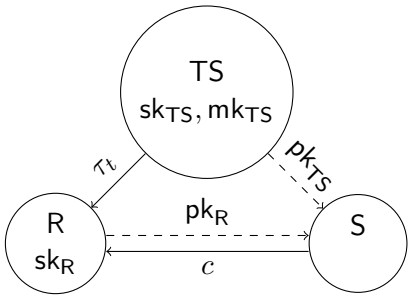


Figure 3.1 – Exchanged values between participants

and

$$\Pr_{\gamma}\left[\mathsf{Dec}(\mathsf{sk}_R, \mathsf{mk}_{TS}, \mathsf{Enc}(\mathsf{pk}_{TS}, \mathsf{pk}_R, m, t, \pi; \gamma), \pi) = m\right] = 1$$

The key generation algorithm of the receiver $\mathsf{KeyGen_R}$ sometimes takes the trusted server public key $\mathsf{pk_{TS}}$ as input. We however define our $\mathsf{KeyGen_R}$ to be independent from $\mathsf{pk_{TS}}$ as it was done in some constructions [NMKM09, MNM10]. If $\mathsf{KeyGen_R}$ is dependent to $\mathsf{pk_{TS}}$, the receiver needs to get the trusted server public key before the generation of its key pair. If they are independent, the receiver does not need any communication with the trusted server before the release time, it will be therefore more efficient.

We consider two security objectives of the timed-release encryption. One is the confidentiality of the message until its release time against the receiver. The other is the anonymity of the sender and the receiver against the trusted server.

3.2 Security Models

In this section, we define security models of timed-release encryption. As a timed-release encryption brings a trusted server into cryptosystem, we can consider the following adversaries:

- A receiver who wants to decrypt a ciphertext before the release time;
- A trusted server which is eavesdropping the communication between a sender and a receiver and wants to break the confidentiality of a message;
- An eavesdropper who wants to decrypt a ciphertext without any secret key.

Similarly, we propose security definitions with three attack models: Chosen plaintext attack (CPA), non-adaptive chosen ciphertext attack (CCA1), and adaptive chosen ciphertext attack (CCA).

We assume that the receiver and the trusted server never collude as the attack is trivial in that case. Along with the decryption key, the release time is also required for the

decryption. We assume that the release time is known to adversaries as it can be found by an exhaustive search if it is unknown.

We adopt the security models of Dent and Tang [DT07]. So, the adversary has always access to the time bound key oracle while the access to the decryption oracle is restricted by the attack models. However, our security models are slightly different from them. The adversary can get any time bound key except that of the challenge time period by querying to the oracle. Therefore, the time periods are not necessary to be an increasing sequence and they can be a decreasing sequence or an arbitrary sequence. Moreover, the decryption oracle takes a ciphertext and a time bound key as inputs, instead of taking a ciphertext and a time period. Thus, only the trusted server type adversary can decrypt any ciphertext, except the challenge ciphertext, of the challenge time period as it can generate corresponding time bound key and other adversaries cannot decrypt any ciphertext of the challenge time period. This is an appropriate setting for the timed-release encryption because all ciphertexts with same time period become decryptable at the same time, along with the release of the time bound key from the trusted server. Therefore, no one can decrypt a ciphertext of the challenge time period as long as the trusted server is not malicious.

In this section, we will describe three security models, which are indistinguishability against the receiver type adversary, the trusted server type adversary and the eavesdropper type adversary, that we will use in the rest of this chapter. We first define security games with oracles in Table 3.1.

Table 3.1 – Outputs of the decryption oracles $\mathcal{O}_1$ and $\mathcal{O}_2$, and the time bound key oracle $\mathcal{Q}$ for security games by attack types.

	$\mathcal{O}_1(\tau', \mathsf{ct}')$	$\mathcal{O}_2(\tau', \mathsf{ct}')$	$\mathcal{Q}(t')$
CPA	ε	ε	$\mathcal{C}.\mathsf{Broadcast}(\mathsf{sk}_{\mathsf{TS}}, t', \pi)$
CCA1	$\mathcal{C}.\mathsf{Dec}(\mathsf{sk}_{\mathsf{R}}, \tau', \mathsf{ct}', \pi)$	ε	$\mathcal{C}.\mathsf{Broadcast}(\mathsf{sk}_{\mathsf{TS}}, t', \pi)$
CCA	$\mathcal{C}.\mathsf{Dec}(\mathsf{sk}_{\mathsf{R}}, \tau', \mathsf{ct}', \pi)$	$\mathcal{C}.\mathsf{Dec}(\mathsf{sk}_{\mathsf{R}}, \tau', \mathsf{ct}', \pi)$	$\mathcal{C}.\mathsf{Broadcast}(\mathsf{sk}_{\mathsf{TS}}, t', \pi)$

When the adversary is on the receiver side, the receiver secret key can be selected by the adversary. Therefore, the decryption oracle can always be simulated by the adversary if it has the access to the time bound key oracle, to which the adversary always has the access. Hence, CPA, CCA1 and CCA are equivalent and then we only consider the CPA model. The IND-R-CPA game is defined as follows:

Game $\mathsf{IND\text{-}R\text{-}CPA\text{-}b}^{\mathcal{A}}_{\mathcal{C}}(\lambda)$:

1: $\pi \leftarrow \mathcal{C}.\mathsf{Setup}(1^\lambda)$
2: $(\mathsf{sk}_{\mathsf{TS}}, \mathsf{pk}_{\mathsf{TS}}, \mathsf{mk}_{\mathsf{TS}}) \leftarrow \mathcal{C}.\mathsf{KeyGen}_{\mathsf{TS}}(\pi)$
3: $(\mathsf{pk}_{\mathsf{R}}, m_0, m_1, t, s_1) \leftarrow \mathcal{A}_1^{\mathcal{Q}(\cdot)}(\mathsf{pk}_{\mathsf{TS}}, \pi)$ ▷ s_1: State of $\mathcal{A}_1$
4: $\mathsf{ct} \leftarrow \mathcal{C}.\mathsf{Enc}(\mathsf{pk}_{\mathsf{TS}}, \mathsf{pk}_{\mathsf{R}}, m_b, t, \pi)$
5: $b' \leftarrow \mathcal{A}_2^{\mathcal{Q}(\cdot)}(\mathsf{ct}, s_1)$
6: **if** t was queried to $\mathcal{Q}$ by $\mathcal{A}_1$ or $\mathcal{A}_2$ **then** *abort*

7: **return** b'

When the adversary is on the trusted server side, the trusted server key pair is under the control of the adversary. Therefore, the adversary can compute a time bound key of any time period, and the access to time bound key oracle $\mathcal{Q}$ is not necessary. In the $\mathsf{CCA1}$ and CCA models, the adversary has the capacity to decrypt a ciphertext with a chosen time bound key by querying to the oracle $\mathcal{O}_1$ when it picks the challenge messages and the challenge time period. In the CCA model, the adversary can decrypt a ciphertext, if it is not the challenge ciphertext, with a chosen time bound key by querying to the oracle $\mathcal{O}_2$ when it outputs the response bit. For $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA}\}$, the $\mathsf{IND\text{-}TS\text{-}ATK}$ game is defined as follows:

Game $\mathsf{IND\text{-}TS\text{-}ATK\text{-}b}^{\mathcal{A}}_{\mathcal{C}}(\lambda)$:

1: $\pi \leftarrow \mathcal{C}.\mathsf{Setup}(1^\lambda)$
2: $(\mathsf{pk_{TS}}, s_0) \leftarrow \mathcal{A}_0(\pi)$ ▷ s_0: State of $\mathcal{A}_0$
3: $(\mathsf{sk_R}, \mathsf{pk_R}) \leftarrow \mathcal{C}.\mathsf{KeyGen_R}(\mathsf{pk_{TS}}, \pi)$
4: $(m_0, m_1, t, s_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_1(\cdot,\cdot)}(\mathsf{pk_R}, s_0)$ ▷ s_1: State of $\mathcal{A}_1$
5: $\mathsf{ct} \leftarrow \mathcal{C}.\mathsf{Enc}(\mathsf{pk_{TS}}, \mathsf{pk_R}, m_b, t, \pi)$
6: $b' \leftarrow \mathcal{A}_2^{\mathcal{O}_2(\cdot,\cdot)}(\mathsf{ct}, s_1)$
7: **if** c was queried to $\mathcal{O}_2$ by $\mathcal{A}_2$ **then** *abort*
8: **return** b'

When the adversary is an eavesdropper, the adversary is passive and the trusted server key pair and the receiver key pair are honestly computed. In the $\mathsf{CCA1}$ and CCA models, the adversary can get the time bound key of a chosen time period by querying to $\mathcal{Q}$, and has the capacity to decrypt a ciphertext with a chosen time bound key by querying to the oracle $\mathcal{O}_1$ when it selects the challenge messages and the challenge time period. If the challenge time period is already queried to $\mathcal{Q}$, the game will be aborted as the time bound key of the challenge time period should not be given to the adversary. In the CCA model, the adversary can still obtain the time bound key of a chosen time period, except the challenge time period, by querying to $\mathcal{Q}$, and can decrypt a ciphertext, if it is not the challenge ciphertext, with a chosen time bound key by querying to the oracle $\mathcal{O}_2$ when it outputs the response bit. For $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA}\}$, the $\mathsf{IND\text{-}ATK}$ game is defined as follows:

Game $\mathsf{IND\text{-}ATK\text{-}b}^{\mathcal{A}}_{\mathcal{C}}(\lambda)$:

1: $\pi \leftarrow \mathcal{C}.\mathsf{Setup}(1^\lambda)$
2: $(\mathsf{sk_{TS}}, \mathsf{pk_{TS}}, \mathsf{mk_{TS}}) \leftarrow \mathcal{C}.\mathsf{KeyGen_{TS}}(\pi)$
3: $(\mathsf{sk_R}, \mathsf{pk_R}) \leftarrow \mathcal{C}.\mathsf{KeyGen_R}(\mathsf{pk_{TS}}, \pi)$
4: $(m_0, m_1, t, s_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_1(\cdot,\cdot),\mathcal{Q}(\cdot)}(\mathsf{pk_{TS}}, \mathsf{pk_R}, \pi)$ ▷ s_1: State of $\mathcal{A}_1$
5: $\mathsf{ct} \leftarrow \mathcal{C}.\mathsf{Enc}(\mathsf{pk_{TS}}, \mathsf{pk_R}, m_b, t, \pi)$
6: $b' \leftarrow \mathcal{A}_2^{\mathcal{O}_2(\cdot,\cdot),\mathcal{Q}(\cdot)}(\mathsf{ct}, s_1)$
7: **if** t was queried to $\mathcal{Q}$ by $\mathcal{A}_1$ or $\mathcal{A}_2$, or c was queried to $\mathcal{O}_2$ by $\mathcal{A}_2$ **then** *abort*
8: **return** b'

Moreover, we also propose weaker security models, which are the security against selective time chosen plaintext attacks (ST-CPA), selective time non-adaptive chosen ciphertext attacks (ST-CCA1), and selective time adaptive chosen ciphertext attacks (ST-CCA). The difference from CPA, CCA1, and CCA is that the adversary needs to claim its challenge time period before getting any public key. Then, there are some differences in security games. For the trusted server type adversary, $\mathcal{A}_0$ will output the challenge time period t. For the receiver type adversary and the eavesdropper type adversary, $\mathcal{A}_0$, which takes π as input and outputs the challenge time period t and the state s_0, will be given as an extra algorithm, and $\mathcal{A}_1$ takes s_0 as input instead of π. For instance, the IND-R-ST-CPA-b game is defined as follows.

Game $\mathsf{IND\text{-}R\text{-}ST\text{-}CPA\text{-}b}^{\mathcal{A}}_{\mathcal{C}}(\lambda)$:

1: $\pi \leftarrow \mathcal{C}.\mathsf{Setup}(1^\lambda)$
2: $(t, s_0) \leftarrow \mathcal{A}_0(\pi)$ ▷ s_0: State of $\mathcal{A}_0$
3: $(\mathsf{sk_{TS}}, \mathsf{pk_{TS}}) \leftarrow \mathcal{C}.\mathsf{KeyGen_{TS}}(\pi)$
4: $(\mathsf{pk_R}, m_0, m_1, s_1) \leftarrow \mathcal{A}_1^{\mathcal{Q}(\cdot)}(\mathsf{pk_{TS}}, s_0)$ ▷ s_1: State of $\mathcal{A}_1$
5: $\mathsf{ct} \leftarrow \mathcal{C}.\mathsf{Enc}(\mathsf{pk_{TS}}, \mathsf{pk_R}, m_b, t, \pi)$
6: $b' \leftarrow \mathcal{A}_2^{\mathcal{Q}(\cdot)}(\mathsf{ct}, s_1)$
7: **if** t was queried to $\mathcal{Q}$ by $\mathcal{A}_1$ or $\mathcal{A}_2$ **then** *abort*
8: **return** b'

3.2.1 Security Notions

Based on the security games that we stated in Section 3.2, we define the following security notions:

Definition 6 (IND-P-ATK security)**.** *Let $\mathcal{C}$ be a timed-release encryption scheme. Then, we say that the timed-release encryption scheme $\mathcal{C}$ is IND-P-ATK secure if*

$$\mathit{Adv}^{\mathit{IND\text{-}P\text{-}ATK}}_{\mathcal{A},\mathcal{C}}(\lambda) = \left| \Pr\left[\mathit{IND\text{-}P\text{-}ATK\text{-}0}^{\mathcal{A}}_{\mathcal{C}}(\lambda) \to 1\right] - \Pr\left[\mathit{IND\text{-}P\text{-}ATK\text{-}1}^{\mathcal{A}}_{\mathcal{C}}(\lambda) \to 1\right] \right|$$

is a negligible function in λ for all probabilistic and polynomial time algorithm $\mathcal{A}$ for $(P, ATK) \in (\{TS, \varepsilon\} \times \{CPA, CCA1, CCA, ST\text{-}CPA, ST\text{-}CCA1, ST\text{-}CCA\}) \cup (\{R\} \times \{CPA, ST\text{-}CPA\})$.

3.2.2 Relation Between Security Models

As the difference between CCA, CCA1 and CPA (resp. ST-CCA, ST-CCA1 and ST-CPA) is about the accessibility of oracles, we can deduce that IND-P-CCA (resp. IND-P-ST-CCA) security implies IND-P-CCA1 (resp. IND-P-ST-CCA1) security and IND-P-CCA1 (resp. IND-P-ST-CCA1) security implies IND-P-CPA (resp. IND-P-ST-CCA) security for $\mathsf{P} \in \{\mathsf{R}, \mathsf{TS}, \varepsilon\}$. Moreover, we have the following relations.

Theorem 1 (IND-P-ATK security ⇒ IND-ATK security)**.** *Let $\mathcal{C}$ be a timed-release encryption scheme. If $\mathcal{C}$ is IND-P-ATK-secure, then $\mathcal{C}$ is IND-ATK-secure for* $(\{TS\} \times \{CPA, CCA1, CCA, ST\text{-}CPA, ST\text{-}CCA1, ST\text{-}CCA\}) \cup (P, ATK) \in (\{R\} \times \{CPA, ST\text{-}CPA\})$.

Theorem 2 (IND-P-ATK security ⇒ IND-P-ST-ATK security)**.** *Let $\mathcal{C}$ be a timed-release encryption scheme. If $\mathcal{C}$ is IND-P-ATK-secure, then $\mathcal{C}$ is IND-P-ST-ATK-secure for* $(P, ATK) \in (\{TS, \varepsilon\} \times \{CPA, CCA1, CCA\}) \cup (\{R\} \times \{CPA\})$.

We can assume that t is in a small set (e.g. one value for each day of the calendar). By guessing t, we obtain the following result.

Theorem 3 (IND-P-ST-ATK security ⇒ IND-P-ATK security)**.** *Let $\mathcal{C}$ be a timed-release encryption scheme. If the set of time periods t is polynomially bounded and $\mathcal{C}$ is IND-P-ST-ATK-secure, then $\mathcal{C}$ is IND-P-ATK-secure for* $(P, ATK) \in (\{TS, \varepsilon\} \times \{CPA, CCA1, CCA\}) \cup (\{R\} \times \{CPA\})$.

Th. 1 and Th. 2 are trivial as the knowledge of some secret values, and given information at the selection of the challenge time period are the differences between them. Consequently, IND-TS-CCA and IND-R-CPA are the strongest security notions and IND-ST-CPA is the weakest security notion. The summary of relations between security notions can be seen in Fig. 3.2.

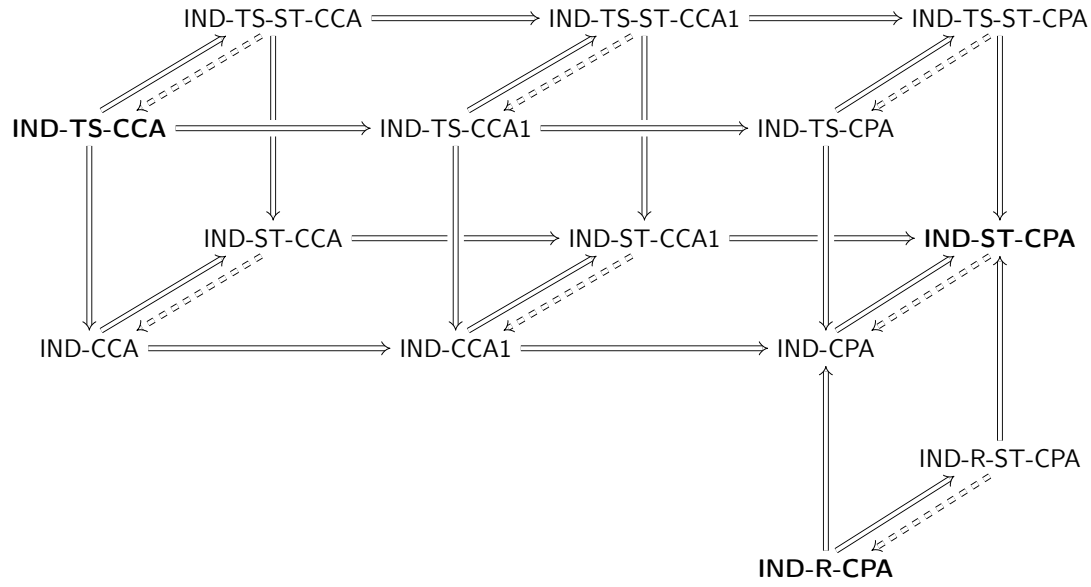


Figure 3.2 – Relations between security models. The reductions with the solid line are tight. The reductions with the dashed line only hold when the time periods are a small set, they, therefore, are not tight.

3.3 Construction With Master Time Bound Key

In this section, we propose a timed-release encryption scheme TRE which has the master time bound key. In addition, our construction does not require $\mathsf{KeyGen_R}$ to be dependent to $\mathsf{pk_{TS}}$ and a hash function which maps to a point on the elliptic curve. Let h_κ be a collision-resistant hash function from $K^* \times E[q]$ to a set F, $\mathcal{E}$ be an asymmetric encryption scheme which consists of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with plaintext space $K \times F$, and f_π be a pseudorandom generator from μ_q to K, i.e. for $\omega \in \mu_q$ uniformly distributed, $f_\pi(\omega)$ is computationally indistinguishable from the uniform distribution over K. Then, our construction with plaintext space K^* is as follows. We note that our $\mathsf{Broadcast}$ is similar to KeyGen of the identity-based encryption scheme of Boneh and Boyen [BB04] and $\mathsf{TS\text{-}release}$ of the timed-release encryption scheme of Cathalo et al. [CLQ05]. In [BB04], KeyGen generates the secret key of a user which can be used to compute the inverse of the random value which is multiplied to the message. In [CLQ05], $\mathsf{TS\text{-}release}$ computes $g^{-(s+H(t))}$ where s is the secret key, $H(t)$ is the hash of a time period t and g is a generator of a group.

- $\mathsf{TRE.Setup}(1^\lambda)$: Pick two prime numbers p and q such that $q|(p \pm 1)$. Pick the finite field $K = \mathbb{F}_{p^2}$ and a supersingular elliptic curve $E(K)$ of cardinality $(p \pm 1)^2$. Then, compute q-torsion subgroup $E[q]$ and the Weil pairing $e : E[q] \times E[q] \longrightarrow \mu_q$ where μ_q is the group of q-th roots of unity in K. Pick κ from the key space of h and output $\pi = (\lambda, K, E, q, e, \kappa)$.

- $\mathsf{TRE.KeyGen_{TS}}(\pi)$: Pick P and Q from $E[q]$ such that $|\langle P \rangle| = |\langle Q \rangle| = q$ and $P \notin \langle Q \rangle$, and pick a, b, c, d uniformly from $\mathbb{Z}_q^*$ until $\langle (1, a) \rangle$, $\langle (b, 1) \rangle$ and $\langle (c, d) \rangle$ are distinct subgroups of $\mathbb{Z}_q \times \mathbb{Z}_q$. Then, compute

$$\mathsf{mk_{TS}} = (1 - ab)(bd - c)^{-1}(bP + Q),$$

$$\mathsf{sk_{TS}} = (a, b, c, d, P, Q)$$

  and

$$\mathsf{pk_{TS}} = (\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)}, \mathsf{pk}_{\mathsf{TS}}^{(2)}) = (P + aQ, bP + Q, cP + dQ),$$

  and output $\mathsf{sk_{TS}}$, $\mathsf{pk_{TS}}$ and $\mathsf{mk_{TS}}$.

  Property 1. *$e(P, P) = e(Q, Q) = 1$, $e(P, Q)e(Q, P) = 1$ and $e(P, Q) \neq 1$. (See the proof below.)*

  Property 2. *$e(\mathsf{pk}_{TS}^{(0)}, \mathsf{pk}_{TS}^{(1)}) = e(P, Q)^{1-ab} \neq 1$ because $\langle (1, a) \rangle$ and $\langle (b, 1) \rangle$ are distinct subgroups of $\mathbb{Z}_q \times \mathbb{Z}_q$.*

- $\mathsf{TRE.KeyGen_R}(1^\lambda)$: Generate a pair of secret and public keys $(\mathsf{sk}, \mathsf{pk})$ by calling $\mathcal{E}.\mathsf{KeyGen}(1^\lambda)$. Then, output $\mathsf{sk_R} = \mathsf{sk}$ and $\mathsf{pk_R} = \mathsf{pk}$.

- $\mathsf{TRE.Broadcast}(\mathsf{sk}_{\mathsf{TS}}, t, \pi)$: Pick s uniformly from $\mathbb{Z}_q^*$. Compute

$$\tau_t = \begin{cases} sP + (ab-1)(c+bt)^{-1}Q, & \text{if } t = -d \\ (1-ab)(d+t)^{-1}P + sQ, & \text{if } t = -cb^{-1} \\ s(d+t)^{-1}P + (s+ab-1)(c+bt)^{-1}Q, & \text{otherwise.} \end{cases}$$

  Property 3. $e(\tau_t, t \cdot \mathsf{pk}_{TS}^{(1)} + \mathsf{pk}_{TS}^{(2)}) = e(\mathsf{mk}_{TS}, t \cdot \mathsf{pk}_{TS}^{(1)} + \mathsf{pk}_{TS}^{(2)}) = e(P,Q)^{1-ab}$ *(See the proof below.)*

- $\mathsf{TRE.Enc}(\mathsf{pk}_{\mathsf{TS}}, \mathsf{pk}_{\mathsf{R}}, m, t, \pi)$: Output $\perp$ if $m \notin K^*$. Pick r_1 uniformly from $\mathbb{Z}_q^*$ and pick r_2 uniformly from K^*. Then, compute

$$\begin{aligned} \mathsf{ct}_0 &= m \cdot r_2, \\ \mathsf{ct}_1 &= r_1 t \cdot \mathsf{pk}_{\mathsf{TS}}^{(1)} + r_1 \cdot \mathsf{pk}_{\mathsf{TS}}^{(2)}, \\ \mathsf{ct}_2 &= \mathcal{E}.\mathsf{Enc}(\mathsf{pk}_{\mathsf{R}}, (r_2 + f_\pi(e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)})^{r_1}), h_\kappa(\mathsf{ct}_0, \mathsf{ct}_1))) \end{aligned}$$

  and output $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$.

  Property 4. $e(\tau_t, \mathsf{ct}_1) = e(\mathsf{pk}_{TS}^{(0)}, \mathsf{pk}_{TS}^{(1)})^{r_1}$

- $\mathsf{TRE.Dec}(\mathsf{sk}_{\mathsf{R}}, \tau_t, \mathsf{ct}, \pi)$: Compute

$$(r_2', \sigma) = \mathcal{E}.\mathsf{Dec}(\mathsf{sk}_{\mathsf{R}}, \mathsf{ct}_2).$$

  Output

$$m = \mathsf{ct}_0 \cdot \left(r_2' - f_\pi(e(\tau_t, \mathsf{ct}_1))\right)^{-1}$$

  if $\sigma = h_\kappa(\mathsf{ct}_0, \mathsf{ct}_1)$, and output $\perp$ otherwise.

Proof of Property 1. $e(P,P) = e(Q,Q) = e(P+Q, P+Q) = 1$ comes from the alternating property of the Weil pairing. Hence, $1 = e(P+Q, P+Q) = e(P,Q)e(Q,P)$ due to bilinearity. Now, assume that there exists $P, Q \in E[q] \setminus \{O\}$ such that $P \notin \langle Q \rangle$ and $e(P,Q) = 1$. Then, we have $e(P, \alpha P + \beta Q) = e(P,Q)^\beta = 1$ for any $\alpha, \beta \in \mathbb{Z}_q$. As q is prime, $\{\alpha P + \beta Q : \alpha, \beta \in \mathbb{Z}_q\} = E[q]$. Hence, it contradicts non-degeneracy, and such P and Q do not exist. Consequently, $e(P,Q) \neq 1$ and $e(P,Q)^{-1} = e(Q,P)$. □

Proof of Property 3. When $t \neq -d$ and $t \neq -cb^{-1}$, we have

$$\begin{aligned} & e(\tau_t, t \cdot \mathsf{pk}_{\mathsf{TS}}^{(1)} + \mathsf{pk}_{\mathsf{TS}}^{(2)}) \\ &= e(s(d+t)^{-1}P + (s+ab-1)(c+bt)^{-1}Q, (c+bt)P + (d+t)Q) \\ &= e(s(d+t)^{-1}P, (d+t)Q)e((s+ab-1)(c+bt)^{-1}Q, (c+bt)P) \\ &= e(P,Q)^s e(Q,P)^{s+ab-1} \\ &= e(P,Q)^{1-ab}. \end{aligned}$$

When $t = -d$, we have

$$\begin{aligned} e(\tau_t, t \cdot \mathsf{pk}_{\mathsf{TS}}^{(1)} + \mathsf{pk}_{\mathsf{TS}}^{(2)}) &= e(sP + (ab-1)(c+bt)^{-1}Q, (c+bt)P) \\ &= e(P,Q)^{1-ab}. \end{aligned}$$

Similarly, when $t = -cb^{-1}$, we have

$$\begin{aligned} e(\tau_t, t \cdot \mathsf{pk}_{\mathsf{TS}}^{(1)} + \mathsf{pk}_{\mathsf{TS}}^{(2)}) &= e((1-ab)(d+t)^{-1}P + sQ, (d+t)Q) \\ &= e(P,Q)^{1-ab}. \end{aligned}$$

With $\mathsf{mk}_{\mathsf{TS}}$, we can also obtain the same result regardless of t.

$$\begin{aligned} &e(\mathsf{mk}_{\mathsf{TS}}, t \cdot \mathsf{pk}_{\mathsf{TS}}^{(1)} + \mathsf{pk}_{\mathsf{TS}}^{(2)}) \\ &= e((1-ab)(bd-c)^{-1}(bP+Q), (c+bt)P + (d+t)Q) \\ &= e((1-ab)(bd-c)^{-1}bP, (d+t)Q)e((1-ab)(bd-c)^{-1}Q, (c+bt)P) \\ &= e(P,Q)^{(1-ab)(bd-c)^{-1}(b(d+t)-c-bt))} \\ &= e(P,Q)^{1-ab}. \end{aligned}$$

□

By the choice of parameters, the q-th torsion subgroup $E[q]$ is a proper subset of E over K. As $E[q] \cong \mathbb{Z}_q \times \mathbb{Z}_q$ [Sil09], there exist $q+1$ distinct subgroups of order q in $E[q]$ and every element in $E[q] \setminus \{O\}$ generates a subgroup of order q. Therefore, we can deduce that $e(P,Q) = 1 \iff P \in \langle Q \rangle$ for all $P, Q \in E[q]$. Hence, in $\mathsf{TRE.KeyGen}_{\mathsf{TS}}$, $|\langle P \rangle| = |\langle Q \rangle| = q$ always holds and $P \notin \langle Q \rangle$ holds with probability of $\frac{q}{q+1}$ for any P and Q randomly chosen from $E[q]$, and $P \notin \langle Q \rangle$ can be easily verified by checking if $e(P,Q)$ is not equal to 1.

Assume that $\mathcal{E}.\mathsf{Dec}(\mathsf{sk}, \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, m)) = m$ always holds for any message m and key pair $(\mathsf{sk}, \mathsf{pk})$ generated by using $\mathcal{E}.\mathsf{KeyGen}$ with some random coin. Then, $\mathsf{TRE.Dec}$ is correct if $e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)})^{r_1} = e(\tau_t, \mathsf{ct}_1)$. From the choice of keys, we have

$$\begin{aligned} e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)})^{r_1} &= e(P + aQ, bP + Q)^{r_1} \\ &= e(P, bP+Q)^{r_1} e(aQ, bP+Q)^{r_1} \\ &= e(P,bP)^{r_1} e(P,Q)^{r_1} e(aQ,bP)^{r_1} e(aQ,Q)^{r_1} \\ &= e(P,Q)^{r_1(1-ab)}. \end{aligned}$$

As $\mathsf{ct}_1 = r_1(t \cdot \mathsf{pk}_{\mathsf{TS}}^{(1)} + \mathsf{pk}_{\mathsf{TS}}^{(2)})$, the decryption is always correct.

3.3.1 Security Analysis

In this section, we will show the following results:

- IND-CPA security of $\mathcal{E}$ implies IND-TS-CPA security of TRE. This security does not depend on h_κ which could be set to a constant function;
- IND-CCA security of $\mathcal{E}$ and the collision-resistance of h_κ imply IND-TS-CCA security of TRE;
- Hardness of the decisional bilinear Diffie-Hellman problem and the PRG property of f_π imply IND-R-ST-CPA security of TRE.

We note that the IND-TS security does not depend at all on the pairing structure. Actually, in this malicious trusted server model, the cryptosystem is equivalent to

$$\mathsf{TRE.Enc}(\mathsf{pk_R}, m; r_2) = (m \cdot r_2, \mathcal{E}.\mathsf{Enc}(\mathsf{pk_R}, (r_2, h_\kappa(\mathsf{ct}_0, \mathsf{ct}_1)))).$$

Therefore, the security solely relies on the one of $\mathcal{E}$.

Theorem 4 (IND-TS-CCA security)**.** *Let $\mathcal{A}$ be an IND-TS-CCA adversary against* TRE *which runs in time η with advantage δ. Then, there exist an IND-CCA adversary $\mathcal{B}$ against $\mathcal{E}$ and a collision adversary $\mathcal{C}$ against h_κ. The advantage of adversary $\mathcal{B}$ is at least $\delta - \delta_{h_\kappa}$ and its time complexity is $\eta + \eta_e + \eta_{f_\pi} + \eta_{h_\kappa}$ where η_e is the time to evaluate the pairing $e(\cdot,\cdot)$, η_e is the time to evaluate the pairing $e(\cdot,\cdot)$, η_{f_π} is the evaluation time of f_π, η_{h_κ} is the evaluation time of h_κ and δ_{h_κ} is the advantage of $\mathcal{C}$.*

Proof. Let $\mathcal{B}$ be an IND-CCA adversary against $\mathcal{E}$. Then, $\mathcal{B}$ consists of two algorithms $\mathcal{B}_1^{\mathcal{O}'_1}$, which chooses two messages m_0 and m_1 by using the decryption oracle $\mathcal{O}'_1$, and $\mathcal{B}_2^{\mathcal{O}'_2}$ which guesses the random bit b by using the decryption oracle $\mathcal{O}'_2$, given an encryption of m_b.

Algorithm $\mathcal{B}_1^{\mathcal{O}'_1}(\mathsf{pk})$:

1: $\pi \leftarrow \mathsf{TRE.Setup}(1^\lambda)$
2: $(\mathsf{pk_{TS}}, s'_0) \leftarrow \mathcal{A}_0(\pi)$
3: $(m'_0, m'_1, t, s'_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(\mathsf{pk}, s'_0)$
4: parse $\mathsf{pk_{TS}} = (\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)}, \mathsf{pk}_{\mathsf{TS}}^{(2)})$
5: $r_1 \xleftarrow{\$} \mathbb{Z}_q^*$
6: $r_2 \xleftarrow{\$} K^*$
7: $r'_2 \leftarrow {m'_1}^{-1} \cdot m'_0 \cdot r_2$
8: $\mathsf{ct}_0 \leftarrow m'_0 \cdot r_2$ ▷ equal to $m'_1 \cdot r'_2$
9: $\mathsf{ct}_1 \leftarrow r_1 t \cdot \mathsf{pk}_{\mathsf{TS}}^{(1)} + r_1 \cdot \mathsf{pk}_{\mathsf{TS}}^{(2)}$
10: $m_0 \leftarrow (r_2 + f_\pi(e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)})^{r_1}), h_\kappa(\mathsf{ct}_0, \mathsf{ct}_1))$
11: $m_1 \leftarrow (r'_2 + f_\pi(e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)})^{r_1}), h_\kappa(\mathsf{ct}_0, \mathsf{ct}_1))$

$s_1 \leftarrow (\mathsf{ct}_0, \mathsf{ct}_1, s_1')$
return m_0, m_1, s_1

Algorithm $\mathcal{B}_2^{\mathcal{O}_2'}(c, s_1)$:
parse $s_1 = (\mathsf{ct}_0, \mathsf{ct}_1, s_1')$
$b \leftarrow \mathcal{A}_2^{\mathcal{O}_2}((\mathsf{ct}_0, \mathsf{ct}_1, c), s_1')$
return b

Oracle $\mathcal{O}_1((\mathsf{ct}_0', \mathsf{ct}_1', \mathsf{ct}_2'), \tau')$:
$(r_2', \sigma) \leftarrow \mathcal{O}_1'(\mathsf{ct}_2')$
if $\sigma = h_\kappa(\mathsf{ct}_0', \mathsf{ct}_1')$ **then**
    $m \leftarrow \mathsf{ct}_0' \cdot (r_2' - f_\pi(e(\tau', \mathsf{ct}_1')))^{-1}$
    return m
end if
return $\perp$

Oracle $\mathcal{O}_2((\mathsf{ct}_0', \mathsf{ct}_1', \mathsf{ct}_2'), \tau')$:
if $c = \mathsf{ct}_2'$ **then**
    (adversary $\mathcal{C}$ only): **if** $h_\kappa(\mathsf{ct}_0, \mathsf{ct}_1) = h_\kappa(\mathsf{ct}_0', \mathsf{ct}_1')$ and $(\mathsf{ct}_0, \mathsf{ct}_1) \neq (\mathsf{ct}_0', \mathsf{ct}_1')$ **then** yield a collision
    return $\perp$
end if
$(r_2', \sigma) \leftarrow \mathcal{O}_2'(\mathsf{ct}_2')$
if $\sigma = h_\kappa(\mathsf{ct}_0', \mathsf{ct}_1')$ **then**
    $m \leftarrow \mathsf{ct}_0' \cdot (r_2' - f_\pi(e(\tau', \mathsf{ct}_1')))^{-1}$
    return m
end if
return $\perp$

The adversary $\mathcal{C}$ gets κ as input and simulates everything else in the game played by $\mathcal{B}$. It can only succeed when the oracle $\mathcal{O}_2$ is given $\mathsf{ct}_2' = c$. If $\mathcal{B}$ ends then $\mathcal{C}$ fails.

When c is an encryption of m_0, $(\mathsf{ct}_0, \mathsf{ct}_1, c)$ is an encryption of m_0' with correct distribution. When c is an encryption of m_1, we need to check that $(\mathsf{ct}_0, \mathsf{ct}_1, c)$ is an encryption of m_1' with correct distribution. As ct_1 is independent to r_2, we only need to check if ct_0 has correct distribution. By the construction, $\mathsf{ct}_0 = m \cdot r_2$ is uniform in K^* because r_2 is uniformly chosen from K^* for any m. Then, ct_0 has correct distribution when c is an encryption of m_1. Indeed, $(\mathsf{ct}_0, \mathsf{ct}_1, c)$ has correct distribution and it is an encryption of $m_0' \cdot r_2 \cdot {r_2'}^{-1}$. As $r_2' = {m_1'}^{-1} \cdot m_0' \cdot r_2$, we can deduce that $(\mathsf{ct}_0, \mathsf{ct}_1, c)$ is an encryption of m_1'. As $\mathcal{O}_1'$ can decrypt any ciphertext which is encrypted with $\mathcal{E}$, $\mathcal{O}_1$ can decrypt any ciphertext encrypted with TRE. $\mathcal{O}_2$ should be able to decrypt any ciphertext encrypted with TRE if given ciphertext is not equal to the challenge ciphertext, i.e. $(\mathsf{ct}_0, \mathsf{ct}_1, c)$. However, $\mathcal{O}_2'$ can only decrypt a ciphertext which is encrypted with $\mathcal{E}$ if given ciphertext is not equal to c. Therefore, when $\mathcal{A}_2$ queries a ciphertext $(\mathsf{ct}_0', \mathsf{ct}_1', c)$ to $\mathcal{O}_2$ where $(\mathsf{ct}_0', \mathsf{ct}_1') \neq (\mathsf{ct}_0, \mathsf{ct}_1)$, $\mathcal{O}_2$ can only output $\perp$ because it cannot queries c to $\mathcal{O}_2'$. Then, we

will show that the decryption of $(\mathsf{ct}'_0, \mathsf{ct}'_1, c)$ is $\perp$ when $(\mathsf{ct}'_0, \mathsf{ct}'_1) \neq (\mathsf{ct}_0, \mathsf{ct}_1)$. As h_κ is collision resistant, the advantage of finding $(\mathsf{ct}'_0, \mathsf{ct}'_1)$ such that $(\mathsf{ct}_0, \mathsf{ct}_1) \neq (\mathsf{ct}'_0, \mathsf{ct}'_1)$ and $h_\kappa(\mathsf{ct}_0, \mathsf{ct}_1) = h_\kappa(\mathsf{ct}'_0, \mathsf{ct}'_1)$ is negligible. Then, the advantage of finding $(\mathsf{ct}'_0, \mathsf{ct}'_1, c)$ whose decryption is not $\perp$ is also negligible. Therefore, $\mathcal{O}_2$ cannot correctly decrypt a ciphertext with negligible probability. Hence, when δ_{h_κ} is the advantage of $\mathcal{C}$, the advantage of $\mathcal{B}$ is lower bounded by $\delta - \delta_{h_\kappa}$ where δ is the advantage of $\mathcal{A}$, and the time complexity of $\mathcal{B}$ is $\eta + \eta_e + \eta_{f_\pi} + \eta_{h_\kappa}$ where η is the running time of $\mathcal{A}$, η_e is the time to evaluate the pairing $e(\cdot,\cdot)$, η_{f_π} is the evaluation time of f_π and η_{h_κ} is the evaluation time of h_κ. □

Theorem 5 (IND-TS-CPA security)**.** *Let $\mathcal{A}$ be an IND-TS-CPA adversary against* TRE *which runs in time η with advantage δ. Then, there exists an IND-CPA adversary $\mathcal{B}$ against $\mathcal{E}$. The advantage of $\mathcal{B}$ is at least δ and its time complexity is $\eta + \eta_e + \eta_{f_\pi}$ where η_e is the time to evaluate the pairing $e(\cdot,\cdot)$, η_e is the time to evaluate the pairing $e(\cdot,\cdot)$ and η_{f_π} is the evaluation time of f_π.*

Proof. The proof is same with the proof of Th. 4. We simply replace h_κ by an identity function. We then make the oracles to always output an empty output. □

Theorem 6 (IND-R-ST-CPA security)**.** *Let $\mathcal{A}$ be an IND-R-ST-CPA adversary against* TRE *which runs in time η with advantage δ. Then, there exist an algorithm $\mathcal{B}$ which solves the decisional bilinear Diffie-Hellman problem and a distinguisher $\mathcal{D}$ between $f_\pi(\mathcal{U}_{\mu_q})$ and $\mathcal{U}_K$. The advantage of $\mathcal{B}$ is at least $\delta - 3/q - \delta_{f_\pi}$ and its time complexity is $\eta + 3\eta_e + \eta_{\mathcal{E}.\mathsf{Enc}}$ where δ_{f_π} is the advantage of $\mathcal{D}$, η_e is the time to evaluate the pairing $e(\cdot,\cdot)$, and $\eta_{\mathcal{E}.\mathsf{Enc}}$ is the execution time of $\mathcal{E}.\mathsf{Enc}$.*

Proof. We use IND$-R-ST-CPA security, an equivalent notion to IND-R-ST-CPA in which the adversary selects only one message and obtains the encryption of this message or a random one. The equivalence is proven by Bellare et al. [BDJR97]. Let $\mathcal{A}$ be an IND$-R-ST-CPA adversary. Then, by using $\mathcal{A}$, we can construct $\mathcal{B}$, which solves the decisional bilinear Diffie-Hellman problem, as follows.

Algorithm $B(\pi, P, Q, aP, bP, r_1P, aQ, bQ, r_1Q, Z)$:

1: $(t, s_0) \leftarrow \mathcal{A}_0(\pi)$
2: $(c', d) \xleftarrow{\$} \mathbb{Z}_q \times \mathbb{Z}_q^*$
3: **if** $c'P - tbP = O$ **then**
4: **return** whether $Z = e(aP, cQ)^{c't^{-1}}$
5: **end if**
6: $\mathsf{pk}_{\mathsf{TS}} \leftarrow (P + aQ, bP + Q, c'P - btP + dQ)$
7: **if** $e(\mathsf{pk}^{(0)}_{\mathsf{TS}}, \mathsf{pk}^{(1)}_{\mathsf{TS}}) = 1$ **then**
8: **return** whether $Z = e(P, r_1Q)$
9: **else if** $e(\mathsf{pk}^{(0)}_{\mathsf{TS}}, \mathsf{pk}^{(2)}_{\mathsf{TS}}) = 1$ **then**
10: **return** whether $Z = e(t^{-1}(c'aP - dP), r_1Q)$
11: **else if** $e(\mathsf{pk}^{(1)}_{\mathsf{TS}}, \mathsf{pk}^{(2)}_{\mathsf{TS}}) = 1$ **then**

12: **return** whether $Z = e(aP, r_1Q)^{c'(d-t)^{-1}}$
13: **end if**
14: $(\mathsf{pk}_{\mathsf{R}}, m_0, s_1) \leftarrow \mathcal{A}_1^{\mathcal{Q}_1}(\mathsf{pk}_{\mathsf{TS}}, s_0)$
15: $r_2 \xleftarrow{\$} K^*$
16: $\omega \leftarrow e(P, r_1Q)Z^{-1}$
17: $\mathsf{ct}_0 \leftarrow m_0 \cdot r_2$
18: $\mathsf{ct}_1 \leftarrow c'r_1P + (d+t)r_1Q$
19: $\mathsf{ct}_2 \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}_{\mathsf{R}}, (r_2 + f_\pi(\omega), h_\kappa(\mathsf{ct}_0, \mathsf{ct}_1)))$
20: $b' \leftarrow \mathcal{A}_2^{\mathcal{Q}_2}((\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2), s_1)$
21: **return** $\neg b'$

Oracle $\mathcal{Q}_1(t')$:
22: $s \xleftarrow{\$} \mathbb{Z}_q$
23: $\tau_{t'} \leftarrow (1+sc')(t'+d)^{-1}P + c'(t'-t)^{-1}(t'+d)^{-1}aP + s(t'-t)(t'+d)^{-1}bP + (t'-t)^{-1}aQ + sQ$
24: **return** $\tau_{t'}$

Oracle $\mathcal{Q}_2(t')$:
25: **if** $t' = t$ **then**
26: **return** $\perp$
27: **end if**
28: $s \xleftarrow{\$} \mathbb{Z}_q$
29: $\tau_{t'} \leftarrow (1+sc')(t'+d)^{-1}P + c'(t'-t)^{-1}(t'+d)^{-1}aP + s(t'-t)(t'+d)^{-1}bP + (t'-t)^{-1}aQ + sQ$
30: **return** $\tau_{t'}$

In order for the algorithm $\mathcal{B}$ to get the correct response from the adversary $\mathcal{A}$, $\mathsf{pk}_{\mathsf{TS}}$ should be a valid trusted server public key and the inputs should be correctly distributed. Firstly, we show that the computational bilinear Diffie-Hellman problem can be easily solved when $\mathsf{pk}_{\mathsf{TS}}$ is not valid. $\mathsf{pk}_{\mathsf{TS}}$ is valid if three subgroups generated by $\mathsf{pk}_{\mathsf{TS}}^{(0)}$, $\mathsf{pk}_{\mathsf{TS}}^{(1)}$ and $\mathsf{pk}_{\mathsf{TS}}^{(2)}$ are distinct. Then, we have three possible cases. If $e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)}) = 1$, we have

$$1 = e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)}) = e(P + aQ, bP + Q) = e(P, Q)^{1-ab}.$$

Then, we can deduce that $ab \equiv 1 \pmod{q}$. Therefore, $e(P, Q)^{abc} = e(P, Q)^c = e(P, cQ)$. If $e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(2)}) = 1$, we have

$$1 = e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(2)}) = e(P + aQ, c'P - tbP + dQ) = e(P, Q)^{d-a(c'-tb)}.$$

Then, we can deduce that $ab \equiv (ac'-d)t^{-1} \pmod{q}$ and we can obtain $abP = t^{-1}(c'aP - dP)$. Therefore, $e(t^{-1}(c'aP - dP), cQ) = e(P, Q)^{abc}$. If $e(\mathsf{pk}_{\mathsf{TS}}^{(1)}, \mathsf{pk}_{\mathsf{TS}}^{(2)}) = 1$, we have

$$1 = e(\mathsf{pk}_{\mathsf{TS}}^{(1)}, \mathsf{pk}_{\mathsf{TS}}^{(2)}) = e(bP + Q, c'P - tbP + dQ) = e(P, Q)^{bd-c'-tb}.$$

Then, we can deduce that $b \equiv c'(d-t)^{-1} \pmod{q}$. Hence, we can obtain $e(aP, cQ)^{c'(d-t)^{-1}} = e(P,Q)^{abc}$.

Table 3.2 – Mapping between the variables in TRE and the variables in the algorithm $\mathcal{B}$.

In TRE	In the algorithm $\mathcal{B}$
$P, Q, a, b, d, t, r_1, r_2, \mathsf{pk_R}, \mathsf{sk_R}$	$P, Q, a, b, d, t, r_1, r_2, \mathsf{pk_R}, \mathsf{sk_R}$
m	m_0
c	$c' - tb$

Now, we need to check the distribution of the inputs to the adversary $\mathcal{A}$. As P, Q, d, r_2, $\mathsf{pk_R}$ and $\mathsf{sk_R}$ are selected as they are selected in TRE. However, a, b, and r_1 are uniformly distributed over $\mathbb{Z}_q^*$ in TRE while they are uniformly distributed over $\mathbb{Z}_q$ in the algorithm $\mathcal{B}$ by Def. 1. As m is selected by the adversary $\mathcal{A}$, we only need to check the distribution of $c' - tb$. As c' is uniformly chosen from $\mathbb{Z}_q$, $c' - tb$ is uniformly distributed over $\mathbb{Z}_q$. When $c'P - tbP = O$, we have $c' - tb \equiv 0 \pmod{q}$. Then, we can solve the decisional bilinear Diffie-Hellman problem as we can compute b by $\mathsf{ct}^{-1} \bmod q$ and then compare Z with $e(aP, r_1Q)^b$. Hence, $c' - tb$ is uniformly distributed over $\mathbb{Z}_q^*$ when the trusted server public key $\mathsf{pk_{TS}}$ is computed.

The distribution of the outputs of the oracles $\mathcal{Q}_1$ and $\mathcal{Q}_2$ needs to be checked. For a fixed time period t', there exist exactly q possible values of $\tau_{t'}$ which satisfy $e(\tau_{t'}, \mathsf{ct}_1) = e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)})^{r_1}$ and $\mathsf{TRE.Broadcast}$ outputs one of these values. By the choice of s, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ can output q different values for a fixed t'. Although $\mathcal{Q}_1$ cannot return an output for $t' = t$ or $t' = -d$, and $\mathcal{Q}_2$ cannot return an output for $t' = -d$, the distribution of the outputs of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are computationally indistinguishable from the actual distribution as q is exponential in the security parameter λ. Hence, the distribution of the outputs of the oracles $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are computationally indistinguishable from the real distribution.

When $Z = e(P,Q)^{abr_1}$, $(\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$ is an encryption of the message m_0 with the public key $\mathsf{pk_{TS}}$. When Z is random value from μ_q, $(\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$ is an encryption of the message $m^* = m_0 \cdot r_2 \cdot (r_2 + f_\pi(e(P,Q)^{r_1} Z^{-1}) - f_\pi(e(P,Q)^{r_1(1-ab)}))^{-1}$. In $\mathsf{IND\$\text{-}R\text{-}ST\text{-}CPA}$, the correct distribution of m^* is the uniform distribution over K^*. As f_π makes a computationally indistinguishable distribution from the uniform distribution over K, m^* is also computationally indistinguishable from the uniform distribution over K^* while the random message should be uniformly chosen from K^*. If we replace $f_\pi(\cdot)$ by a uniform distribution over K, m^* is uniformly distributed over K^* which is the correct distribution for m^*. Therefore, the advantage of $\mathcal{B}$ is reduced by the advantage of $\mathcal{D}$ at most where $\mathcal{D}$ is a distinguisher $\mathcal{D}$ between $f_\pi(\mathcal{U}_{\mu_q})$ and $\mathcal{U}_K$. When switching to the distribution for $\mathcal{B}$, the probability of success is reduced by $3/q + \delta_{f_\pi}$ at most. Hence, the advantage of $\mathcal{B}$ to solve the decisional bilinear Diffie-Hellman problem is lower bounded by $\delta - 3/q - \delta_{f_\pi}$ and the time complexity of $\mathcal{B}$ is $\eta + 3\eta_e + \eta_{\mathcal{E}.\mathsf{Enc}}$ where δ_{f_π} is the advantage of $\mathcal{D}$, η_e is the time to evaluate the pairing $e(\cdot,\cdot)$, and $\eta_{\mathcal{E}.\mathsf{Enc}}$ is the execution time of $\mathcal{E}.\mathsf{Enc}$. □

Using Th. 3, we obtain IND-R-CPA security when the domain of t is small.

3.3.2 Decryption With Master Time Bound Key

The biggest difference between our construction and other constructions is the existence of the master time bound key. By using the master time bound key, a ciphertext of unknown release time can be decrypted. By our construction, a ciphertext consists of $(\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$. In order to decrypt a ciphertext, we need to compute $e(\tau_t, \mathsf{ct}_1)$ should be computed. Due to Property 3, the master time bound key $\mathsf{mk}_{\mathsf{TS}}$ can replace any time bound key. Indeed, the receiver only needs to ask the trusted server to compute $e(\mathsf{mk}_{\mathsf{TS}}, \mathsf{ct}_1)$ to decrypt the ciphertext. As ct_1 is independent from the message, the trusted server cannot learn anything about the message while computing $e(\mathsf{mk}_{\mathsf{TS}}, \mathsf{ct}_1)$.

Similarly, the trusted server can terminate its service without any computational and storage overhead while preventing losing the encrypted data of users by revealing the master time bound key. As the master time bound key can replace any time bound key, we do not need any extra algorithm for the decryption with $\mathsf{mk}_{\mathsf{TS}}$. This is an advantage for the trusted server, as it does not need to provide any additional algorithm for the decryption with master time bound key.

On the other hand, the time bound key τ_t which is generated by TRE.Broadcast can be equal to the master time bound key $\mathsf{mk}_{\mathsf{TS}}$ depending on the random value s. Therefore, the master time bound key can be broadcasted by the trusted server as a time bound key of a certain time period. However, it can happen with probability of at most $1/(q-1)$ where q is exponential in the security parameter λ. As $1/(q-1)$ is negligible in λ, the probability that a fresh time bound key is equal to the master time bound key is negligible. Moreover, the trusted server could also easily prevent this problem by comparing the time bound key with master time bound key before the broadcast.

3.3.3 Discussion

In our construction, we are selecting a specific elliptic curve, which is a supersingular elliptic curve with j-invariant 0 or 1728, in order to minimize computation overhead. However, any pairing-friendly curve can be used for our construction if the q-torsion subgroup has the size of q^2.

As our construction uses an elliptic curve over an extension field $\mathbb{F}_{p^2}$, we first need to know what is the computational overhead compared to other constructions which work on $\mathbb{F}_p$. However, it is not easy to compare the exact overhead because some constructions [BC04, CLQ05, CHKO06, HYL05, CHS07] are based on the generic bilinear pairing, and some constructions [NMKM09, MNM10] are based on the generic identity-based encryption. Therefore, their computational cost is dependent on the underlying

bilinear pairing and the underlying identity-based encryption scheme. An identity-based encryption scheme is usually based on the bilinear pairing[1], and it always requires at least one evaluation of the bilinear pairing. One of most common instantiation of the bilinear pairing is to use the Weil pairing or the Tate pairing after applying a distortion map to one of two input points. As the distortion map maps a point defined on the elliptic curve over a field $\mathbb{F}_p$ to $\mathbb{F}_{p^2}$, the computation of the Weil pairing or the Tate pairing is actually the computations on $\mathbb{F}_{p^2}$. Therefore, the asymptotic complexities of our construction and other constructions are similar as long as the bilinear pairing is the most complex computation.

Our construction can also be built on the top of generic bilinear pairings. Let G be an additive cyclic group, G_T be a multiplicative cyclic group, and $\hat{e} : G \times G \longrightarrow G_T$ be a bilinear pairing. If we define $P = (g, 0)$, $Q = (0, g)$ and $e(aP + bQ, cP + dQ) = e((ag, bg), (cg, dg)) = \hat{e}(ag, dg)\hat{e}(cg, bg)^{-1}$, we can obtain the same construction on the top of generic pairing. The computation of e however requires two evaluations of a generic bilinear pairing $\hat{e}$. As we mentioned in the previous paragraph, a generic bilinear pairing is usually instantiated with the Weil pairing or the Tate pairing. We therefore use the Weil pairing over $\mathbb{F}_{p^2}$ for the efficiency. We note that the construction with a generic pairing can be more efficient than our construction with the Weil pairing if one can instantiate a more efficient bilinear pairing.

In our construction, the encryption requires a single evaluation of the Weil pairing e. As the encryption always requires to compute $e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)})$, it can be precomputed by the trusted server and integrated into the trusted server public key. Therefore, we can make the encryption faster by replacing the trusted server public key $\mathsf{pk}_{\mathsf{TS}}$ to $(e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)}), \mathsf{pk}_{\mathsf{TS}}^{(1)}, \mathsf{pk}_{\mathsf{TS}}^{(2)})$.

3.4 Parameter Selection

As our construction is based on the hardness of bilinear Diffie-Hellman problem on an elliptic curve, the discrete logarithm problem on the elliptic curve can be reduced to a discrete logarithm problem on a finite field extension by using a pairing [MOV93]. We therefore need to select parameters while considering attacks on the both elliptic curve and finite field extension. As our elliptic curve always has the embedding degree 2, the selection of p needs to consider the hardness of the discrete logarithm problem on $\mathbb{F}_{p^2}$. For instance, we require 3072-bit of group size and 256-bit of exponent size for finite field crptography, and 256-bit of point order for elliptic curve cryptography for 128-bit security by NIST recommendation [BBB+12]. As P and Q have the order q, we require p to be a 1536-bit prime and q to be a 256-bit prime for 128-bit security. On Table 3.3, we summarized the minimum bit length of p and q by security level.

[1] There also exist several identity-based encryption schemes which do not require a bilinear pairing [AB09, ABB10, DG17], but we do not compare with them.

Table 3.3 – Bit length of p and q by security level λ

λ	$\log_2 p$	$\log_2 q$
80	512	160
112	1024	224
128	1536	256
192	3840	384
256	7680	512

In the parameter generation, one major problem is the generation of p and E. As p should be a multiple of q and E should be a supersingular elliptic curve of cardinality $(p \pm 1)^2$ over $\mathbb{F}_{p^2}$, we need an efficient way to generate p and E. Instead of generating a random prime and a random supersingular elliptic curve, we can use the elliptic curves which are known to be supersingular. One of possible ways is to use an elliptic curve with j-invariant 0 or 1728. For an elliptic curve of j-invariant 0, we need to select p such that $p \equiv 2 \pmod 3$. Instead p should satisfy $p \equiv 3 \pmod 4$ for an elliptic curve of j-invariant 1728 in order for the elliptic curve to be supersingular. In these cases, the cardinality of E is $(p+1)^2$. We therefore only need to sample an appropriate prime number p for the parameter selection.

As p should be a large prime which satisfies either $p \equiv 3 \pmod 4$ or $p \equiv 2 \pmod 3$, and $q|(p+1)$ at the same time, it might require more computational resources compared to the generation of a random prime of the same bit length [JPV00]. The computation of p however is only required once in the lifetime of the system. Due to the constraints on p, the generation of p is 2 times slower by rejection sampling if we use a standard prime number generation. However, it can be improved by modifying standard prime number generations to first check if p satisfies either $p \equiv 3 \pmod 4$ or $p \equiv 2 \pmod 3$ before checking if p is prime.

3.4.1 Relation Between Keys And Release Times

We now study the scenario where multiple trusted servers are sharing a single system parameter, which means that these trusted servers are using the same elliptic curves and all their public keys are in the same q-torsion subgroup $E[q]$. Therefore, there might exist some relations between public keys of trusted servers.

By our trusted server key generation algorithm, each component of the public key $\mathsf{pk}_{\mathsf{TS}}^{(0)}$, $\mathsf{pk}_{\mathsf{TS}}^{(1)}$ and $\mathsf{pk}_{\mathsf{TS}}^{(2)}$ are generators of different subgroups of order q. As there exist $q+1$ subgroups of order q and $q-1$ elements of a subgroup are generators of the subgroup, there exist $q(q+1)(q-1)^4$ possible trusted server public keys. As $e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)}) \in \mu_q$ and $t \cdot \mathsf{pk}_{\mathsf{TS}}^{(1)} + \mathsf{pk}_{\mathsf{TS}}^{(2)} \in E[q]$, there only exist q^3 possible $(e(\mathsf{pk}_{\mathsf{TS}}^{(0)}, \mathsf{pk}_{\mathsf{TS}}^{(1)}), t \cdot \mathsf{pk}_{\mathsf{TS}}^{(1)} + \mathsf{pk}_{\mathsf{TS}}^{(2)})$ pairs. Hence, there must exist two different trusted server public keys such that an encryption of a message with one trusted server public key is also a valid encryption of same message

with another trusted server public key by the pigeonhole principle.

Let $\mathsf{pk}_{\mathsf{TS},0} = (\mathsf{pk}^{(0)}_{\mathsf{TS},0}, \mathsf{pk}^{(1)}_{\mathsf{TS},0}, \mathsf{pk}^{(2)}_{\mathsf{TS},0})$ be a trusted server public key and $(\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$ be an encryption of a message m with $\mathsf{pk}_{\mathsf{TS},0}$ for a release time t. In order for $(\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$ to be a valid ciphertext under another trusted server public key $\mathsf{pk}_{\mathsf{TS},1} = (\mathsf{pk}^{(0)}_{\mathsf{TS},1}, \mathsf{pk}^{(1)}_{\mathsf{TS},1}, \mathsf{pk}^{(2)}_{\mathsf{TS},1})$ for a release time t', $e(\mathsf{pk}^{(0)}_{\mathsf{TS},0}, \mathsf{pk}^{(1)}_{\mathsf{TS},0}) = e(\mathsf{pk}^{(0)}_{\mathsf{TS},1}, \mathsf{pk}^{(1)}_{\mathsf{TS},1})$ and $t \cdot \mathsf{pk}^{(1)}_{\mathsf{TS},0} + \mathsf{pk}^{(2)}_{\mathsf{TS},0} = t' \cdot \mathsf{pk}^{(1)}_{\mathsf{TS},1} + \mathsf{pk}^{(2)}_{\mathsf{TS},1}$, $(\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$ should be satisfied. We can then easily deduce that

$$(-\alpha \cdot \mathsf{pk}^{(1)}_{\mathsf{TS},0},\ \alpha^{-1} \cdot \mathsf{pk}^{(0)}_{\mathsf{TS},0},\ t \cdot \mathsf{pk}^{(1)}_{\mathsf{TS},0} + \mathsf{pk}^{(2)}_{\mathsf{TS},0} - t'\alpha^{-1} \cdot \mathsf{pk}^{(0)}_{\mathsf{TS},0})$$

and

$$(\alpha \cdot \mathsf{pk}^{(0)}_{\mathsf{TS},0},\ \alpha^{-1} \cdot \mathsf{pk}^{(1)}_{\mathsf{TS},0},\ (t - t'\alpha^{-1}) \cdot \mathsf{pk}^{(1)}_{\mathsf{TS},0} + \mathsf{pk}^{(2)}_{\mathsf{TS},0})$$

satisfy given conditions for $\alpha \in \mathbb{Z}_q^*$.

Let $G_{\bar{K}/K}$ be the Galois group of $\bar{K}/K$ where $\bar{K}$ is the algebraic closure of K. For $\sigma \in G_{\bar{K}/K}$, we have $e(P,Q)^\sigma = e(P^\sigma, Q^\sigma)$ where $P^\sigma = (P_x^\sigma, P_y^\sigma)$ where P_x and P_y are x- and y-coordinates of the point P. By the choice of our parameters, there only exists $\sigma = p$ such that $P^\sigma \neq P$. As $q|(p \pm 1)$, we have $p \equiv \mp 1 \pmod{q}$ and we can deduce that $e(\mp P^p, Q^p) = e(P^p, \mp Q^p) = e(P,Q)$ for any $P, Q \in E$. Hence, a ciphertext under a public key $\mathsf{pk}_{\mathsf{TS},0} = (\mathsf{pk}^{(0)}_{\mathsf{TS},0}, \mathsf{pk}^{(1)}_{\mathsf{TS},0}, \mathsf{pk}^{(2)}_{\mathsf{TS},0})$ for a time period t is a valid ciphertext under a public key

$$(\mp\alpha(\mathsf{pk}^{(0)}_{\mathsf{TS},0})^p, \alpha^{-1}(\mathsf{pk}^{(1)}_{\mathsf{TS},0})^p, t \cdot \mathsf{pk}^{(1)}_{\mathsf{TS},0} + \mathsf{pk}^{(2)}_{\mathsf{TS},0} - t'\alpha^{-1}(\mathsf{pk}^{(1)}_{\mathsf{TS},0})^p)$$

or

$$(\pm\alpha(\mathsf{pk}^{(1)}_{\mathsf{TS},0})^p, \alpha^{-1}(\mathsf{pk}^{(0)}_{\mathsf{TS},0})^p, t \cdot \mathsf{pk}^{(1)}_{\mathsf{TS},0} + \mathsf{pk}^{(2)}_{\mathsf{TS},0} - t'\alpha^{-1}(\mathsf{pk}^{(0)}_{\mathsf{TS},0})^p)$$

for a release time t' for any $\alpha \in \mathbb{Z}_q^*$.

3.5 Experimental Result

We implemented our construction to measure the performance. In our construction, we require an asymmetric scheme $\mathcal{E}$, a collision-resistant hash function h_κ and a pseudorandom generator f_π. We however omitted them in our experiment.

We also applied some optimizations to our construction for the implementation. In Enc, $\mathsf{pk}^{(0)}_{\mathsf{TS}}$ is only used for the computation of $e(\mathsf{pk}^{(0)}_{\mathsf{TS}}, \mathsf{pk}^{(1)}_{\mathsf{TS}})^{r_1}$ and never be reused. We therefore replaced $\mathsf{pk}^{(0)}_{\mathsf{TS}}$ in the trusted server public key by $e(\mathsf{pk}^{(0)}_{\mathsf{TS}}, \mathsf{pk}^{(1)}_{\mathsf{TS}})^{r_1}$. By this, Enc does not require any pairing computation while $\mathsf{KeyGen}_{\mathsf{TS}}$ requires an additional pairing computation.

The experiment was done on a machine with the AMD Opteron 8354 processor with

128 GB of memory. The implementation was done by using the SageMath 8.7. In the experiment, all parameters are selected as we described in Section 3.4 for each security level λ. We executed each instance for 1000 times and computed the average execution time of each algorithm. The results are on Table 3.4 and the source code that we used can be found from Appendix A.1.

Table 3.4 – Execution time for each security level

λ	Setup	$\mathsf{KeyGen}_{\mathsf{TS}}$	Broadcast	Enc	Dec
80	9.863174 s	0.631875 s	0.177999 s	0.090850 s	0.309342 s
112	67.008768 s	1.291937 s	0.315217 s	0.158717 s	0.547850 s
128	164.499172 s	2.067859 s	0.457077 s	0.226914 s	0.796429 s
192	4571.065143 s	9.203592 s	1.553107 s	0.793923 s	2.796017 s

As we can find from the results, Setup requires a lot of computational resources. However, most of its execution time is due to the sampling of a large prime number which satisfies the conditions in Section 3.4. However, this computation can be easily parallelized, and Setup needs to be run only once at the start of the service. The execution times of $\mathsf{KeyGen}_{\mathsf{TS}}$, Enc and Dec are not significant, but we can find that the execution time of Enc is faster than the execution time of Dec as Enc does not require any pairing computation due to the optimization, that we stated above, while Dec requires a pairing computation. When a trusted server is selecting the interval between two consecutive release times, the execution time of Broadcast should be taken into account as the trusted server should be able to compute a time bound key between two release times. For instance, this interval must be larger than 1.5 seconds if we are aiming 192-bit of security. We can however use the interval which is smaller than the execution time of Broadcast by parallelizing the computations of Broadcast. Let $t_0, t_1, \ldots$ be time periods. If the interval between t_i and t_{i+1} for $i \in \{0, 1, \ldots\}$ is x times smaller than the execution time of Broadcast, we can use x threads to compute time bound keys by making the j-th thread to compute time bound keys for time periods t_{ax+j} for $j \in \{0, \ldots, x-1\}$ and non-negative integer a. As the execution time of Broadcast is smaller than the interval between t_i and t_{i+x}, all time bound keys, except first $x-1$ time bound keys, can be computed before its release time. For first $x-1$ time bound keys, the trusted server can precompute them before it starts its service.

We note that Enc requires additional computations to evaluate $\mathcal{E}.\mathsf{Enc}$, h_κ and f_π, and Dec requires additional computations to evaluate $\mathcal{E}.\mathsf{Dec}$, h_κ and f_π in the real world application.

3.6 Conclusion of Chapter

In this chapter, we proposed a timed-release encryption scheme which has the master time bound key. With master time bound key, a ciphertext can be decrypted even if

the release time of the ciphertext is unknown. We also showed that our construction is IND-TS-CCA-secure and IND-R-ST-CPA-secure. Moreover, we proposed the parameters by security levels, and we implemented our construction and showed its performance.

4 Witness Key Encapsulation Model

In Chapter 3, we have studied on timed-release encryption with a third party. In this chapter, we study a way to improve it by using a witness key encapsulation mechanism.

Third parties that we have seen in Chapter 3 supposed to be centralized. But they could be distributed, with the assumption that at least one part is honest. Let assume a set of third parties whom we call *delegates*. The secret holder can use secret sharing to distribute shares to each delegate. Delegates would simply release their shares in the future at the given date. Concretely, the shares would be encrypted with their public key and their secret key would be released. Managing the protocol could be enforced by a *smart contract* which could reward the delegate for participating or punish them for not releasing on time. Such solutions were proposed by Ning et al. [NDHC18] and Li and Palanisamy [LP19]. One problem is that we now need to agree on a contract, reward delegates, and also make everyone (delegates included) to pay some deposit at the beginning of the protocol. The money is blocked until the protocol ends.

Avoiding third parties is more challenging.

Essentially, a smart contract could implement an obfuscated code which only checks time and releases the secret if time has passed. The confidentiality of the secret inside the code is based on obfuscation. There are two difficulties with this:

- we need to have a secure way to verify that some time has passed in a program (smart contracts have no clock);
- currently known obfuscation schemes are based on multilinear maps which are not efficient so far.

The first problem is solved with blockchains based on a proof-of-work. Essentially, a long-enough valid blockchain is a proof that some time has elapsed. Contrarily to timed-release encryption where every receiver must spend a huge amount of work to decrypt,

here we use the fact that some work was already done by the blockchain infrastructure. The verification is easier. Letting ω denote the blockchain, the verification is done by checking a predicate $R(t, \omega)$ on an instance t, telling that ω is a valid blockchain of length at least t. This idea, combined with witness encryption (as presented next) solves the timed-release problem, as shown by Liu et al. [LJKW18].

Witness encryption was first proposed by Garg et al. [GGSW13]. The idea is that a secret is encrypted together with an instance x of an NP language. The resulted ciphertext can be decrypted by using a witness ω for the instance, which is verified by a relation $R(x, \omega)$.

Witness encryption based on an NP-complete language is a powerful primitive as it implies a witness encryption based on any NP language. Anyone can encrypt a message for anyone who could solve a given equation $R(x, .)$. This is very nice to encrypt a bounty.

There are several kinds of witness encryption schemes. Regular schemes offer IND-CPA security when the encryption key x does *not* belong to the language. However, in that case, decryption is not possible either. Extractable schemes are such that for any efficient adversary, there must exist an efficient extractor such that it is hard, either for the adversary to decrypt, or for the extractor having the same inputs not to produce a witness. Like obfuscation, existing constructions of extractable witness encryption are based on multilinear maps which are currently heavy algorithms. To mitigate their complexity, offline schemes allow efficient encryption but have an additional setup algorithm which does the heavy part of the scheme.

Cramer and Shoup proposed the notion of Hash-proof systems which is also based on NP languages [CS02]. Those systems use a special hash function which has a public key and a secret key. We can hash an instance x either with its witness ω together with the public key, or with the secret key alone. Somehow, the secret key is a wildcard for a missing witness. Hash-proof systems are used to build *adaptively secure cryptosystems* [CS03]. We encrypt a message by picking a random (x, ω) pair in the relation R and hashing x to obtain a key to encrypt the message. The value x must be added in the ciphertext. We decrypt using the secret key. In witness encryption, the construction is upside down: we encrypt by generating a key pair for the hash-proof system and we hash using the secret key. The value of the public key must be added in the ciphertext. We decrypt by hashing with a witness and the public key. One problem is to build a hash-proof system with extractable security for an NP-complete problem.

The notion of security with extractor of the witness encryption is non-falsifiable [Nao03]. There exists other non-falsifiable notions which use extractors. For instance, the knowledge-of-exponent assumption (KEA) was proposed by Damgård in 1991 [Dam91]. It says that for any efficient adversary, there must exist an efficient extractor such that given (g, g^y) in a given group, it is hard, either for the adversary to construct a pair of form (g^x, g^{xy}), or for the extractor having the same input not to produce x. KEA can be proven in the

generic group model [Den06, AF07].

Witness encryption can be achieved using obfuscation: the ciphertext is an obfuscated program which takes as input ω and releases the plaintext if $R(x, \omega)$ holds. As shown by Chvojka et al. [CJK19], this can also be turned into an offline witness encryption scheme. Hence, it seems that all solutions reduce to obfuscation.

Our contribution

In this chapter, we construct an efficient witness encryption scheme[1] for a variant of the subset sum problem (which we show to be NP-complete). Concretely, an instance is a tuple $x = (x_1, \ldots, x_t, s)$ of vectors, a witness is a vector $\omega = (a_1, \ldots, a_t)$ of non-negative small integers, and $R(x, \omega)$ is equivalent to the vectorial equation $a_1x_1 + \cdots + a_tx_t = s$. In the regular subset sum problem, all a_i must be boolean and the vectors x_i and s are actually integers. Here, we require the a_i to be polynomially bounded and vectors have some dimension d.

Our encryption scheme is based on the following idea which we explain for $d = 1$ as follows: encryption generates a (n, ℓ, g, k) tuple such that g has multiplicative order ℓ modulo n and k is invertible modulo ℓ. The values k and ℓ are not revealed. Then, the ciphertext consists of $(n, g, y_1, \ldots, y_t, e)$ with $y_i = k^{x_i} \bmod \ell$ and the encryption e of the secret by using the key $h = g^{k^s} \bmod n$. The decryption rebuilds $h = g^{y_1^{a_1} \cdots y_t^{a_t}} \bmod n$ from the ciphertext and the witness, then decrypts e. The key idea in the security is that the operations need to be done in the hidden group of residues modulo ℓ. The product $y_1^{a_1} \cdots y_t^{a_t}$ can only be done over the integers, assuming that the a_i's are small. However, the basis-g exponential reduces it modulo ℓ in a hidden manner.

Interestingly, computing the products $y_1^{a_1} \cdots y_t^{a_t}$ from reduced y_i values resembles to the notion of *graded encoding*, which is the basis of currently existing multilinear maps. In our construction, a 1-level encoding of x is $k^x \bmod \ell$. Hence, each y_i is a 1-level encoding of x_i and $y_1^{a_1} \cdots y_t^{a_t}$ is an encoding of $a_1x_1 + \cdots + a_tx_t$ of level $a_1 + \cdots + a_t$. The level of encoding is proportional to the size of the integer.

Our construction is based on the following homomorphism from $\mathbb{Z}^d$ to the hidden group $\mathbb{Z}_\ell^*$:

$$(x_1, \ldots, x_d) \mapsto k_1^{x_1} \cdots k_d^{x_d} \bmod \ell$$

This hidden group is included in a larger structure $\mathbb{Z}$ in which we can do multiplications which are compatible with the hidden group. However, we later need to reduce elements in a compatible and hidden manner. We call this reduction operation *hashing*. In our construction, it is done by the $y \mapsto g^y \bmod n$ function. We formalize the notion of hidden group with hashing (HiGH).

[1] Actually, we construct a KEM.

To be able to prove security, we need an assumption which generalizes the knowledge-of-exponent assumption: we need to say that computing h implies being able to write it as the exponential of some (multiplicative) linear combination of the y_i's with known exponents. To do so, we must make the group sparse over the integers (so that we cannot find element by chance). For that, we duplicate the basis-k exponential like in the Cramer-Shoup techniques [CS98]. Then, we formulate two computational assumptions. The first one, which we call the *kernel assumption* says that it is hard to find a non-zero vector x mapping to 1 by the above homomorphism, with only public information (i.e., the ciphertext). We show that it is equivalent to the order assumption for the RSA parameter n in Th. 10. The second one, which is non-standard, is similar to the knowledge of exponent assumption, and so is non-falsifiable.

We prove security in a generic HiGH model. We also propose an RSA-based HiGH for which we prove security (under non-standard but realistic assumptions) for instances x which have no $a_1x_1 + \cdots + a_tx_t = 0$ relation with small a_i.

We derive some programming techniques to encode a system of boolean equations into our variation of the subset sum problem. We apply our construction to the timed-release construction of Liu et al. [LJKW18] with the bitcoin infrastructure. We show that using a slightly different block structure, we can significantly reduce the complexity.

4.1 Primitives of Witness Key Encapsulation Mechanism

We adapt the primitives of witness encryption from Garg et al. [GGSW13] so that we have a key encapsulation mechanism instead of a cryptosystem.

Definition 7 (Witness key encapsulation mechanism (WKEM))**.** *Let L be an NP language defined by the predicate R. A witness key encapsulation mechanism for the language L consists of following two algorithms and a domain $\mathcal{K}_\lambda$ defined by a security parameter λ:*

- $\mathsf{Enc}(1^\lambda, x) \to K, \mathsf{ct}$*: A probabilistic polynomial-time algorithm which takes a security parameter λ and a word $x \in L$ as inputs, and outputs a plaintext $K \in \mathcal{K}_\lambda$ and a ciphertext* ct*.*
- $\mathsf{Dec}(\omega, \mathsf{ct}) \to K/\perp$*: A deterministic polynomial-time algorithm which takes a witness ω and a ciphertext* ct *as inputs, and outputs a plaintext K or $\perp$ which indicates the decryption failure.*

Then, the following property is satisfied:

- ***Perfect correctness****: For any security parameter λ, for any word x and witness*

ω such that $R(x,\omega)$ is true, we have

$$\Pr_{\gamma}\left[\mathsf{Dec}(\omega,\mathsf{ct}) = K | (K,\mathsf{ct}) \leftarrow \mathsf{Enc}(1^\lambda, x; \gamma)\right] = 1.$$

Clearly, if we have a WKEM for an NP-complete language, we can make a WKEM for any NP language.

Based on security notions of extractable witness encryption [GKP+13] and KEM [CS03], we define extractable indistinguishability as follows.

Definition 8 (Extractable indistinguishability)**.** *Let $(\mathcal{K}_\lambda, \mathsf{Enc}, \mathsf{Dec})$ be a* WKEM *for L. Given an adversary $\mathcal{A}$, we define the following game with $b \in \{0,1\}$:*

Game $\mathsf{IND\text{-}EWE}^b_{\mathcal{A}}(1^\lambda, x)$*:*

1: $\mathsf{Enc}(1^\lambda, x) \to K_1, \mathsf{ct}$
2: *pick a random* $K_0 \in \mathcal{K}_\lambda$
3: $\mathcal{A}(x, K_b, \mathsf{ct}) \to r$
4: ***return*** r

We define the advantage of $\mathcal{A}$ by

$$\mathsf{Adv}^{\mathsf{IND\text{-}EWE}}_{\mathcal{A}}(x) = \Pr[\mathsf{IND\text{-}EWE}^1_{\mathcal{A}}(x) \to 1] - \Pr[\mathsf{IND\text{-}EWE}^0_{\mathcal{A}}(x) \to 1]$$

We say that WKEM *is extractable indistinguishable for a set X of instances x if for any probabilistic and polynomial-time* IND-EWE *adversary $\mathcal{A}$, there exists a probabilistic and polynomial-time extractor $\mathcal{E}$ such that for all $x \in X$, $\mathcal{E}(x)$ outputs a witness of x with probability at least $\mathsf{Adv}^{\mathsf{IND\text{-}EWE}}_{\mathcal{A}}(x)$ or at least $\frac{1}{2}$ up to a negligible term:*

$$\forall x \in X \quad \Pr[R(x, \mathcal{E}(x))] \geq \min\left(\mathsf{Adv}^{\mathsf{IND\text{-}EWE}}_{\mathcal{A}}(x), \frac{1}{2}\right) - \mathsf{negl}(\lambda)$$

Note that for $x \notin L$, no witness exists so $\mathcal{E}(x)$ outputs a witness with null probability. Hence, it must be the case that $\mathsf{Adv}^{\mathsf{IND\text{-}EWE}}_{\mathcal{A}}(x) = \mathsf{negl}(\lambda)$ for all $x \in X - L$. This property for all $x \notin L$ is actually the weaker (non-extractable) security notion of witness encryption.

Ideally, we would adopt this definition for the set X of all possible words. The reason why we introduce X is to avoid some "pathological" words making our construction insecure.

Chvojka et al. [CJK19] requires $\Pr[R(x, \mathcal{E}(x))]$ to be "non-negligible" (without defining what this means). In our notion, we require more. Namely, we require extraction to be as effective as the attack.

The reason why the extractor extracts with probability "at least $\mathsf{Adv} - \mathsf{negl}$ or at least $\frac{1}{2}$" is that we do not care if the extractor is not as good as the adversary when the adversary

has an advantage close to 1. We only care that it is either "substantially good" or at least as good as $\mathcal{A}$.

We define extractable one-way security, which is a weaker security notion than extractable indistinguishability. Later, we show that an extractable one-way scheme can be transformed into an extractable indistinguishable one by generic transformation. Hence, we will be able to focus on making an extractable one-way WKEM.

Definition 9 (Extractable one-wayness)**.** *Let* $(\mathcal{K}_\lambda, \mathsf{Enc}, \mathsf{Dec})$ *be a* WKEM *for* L*. Given an adversary* $\mathcal{A}$*, we define the following game:*

Game $\mathsf{OW\text{-}EWE}_{\mathcal{A}}(1^\lambda, x)$*:*

1: *(ROM only) pick a random function* H
2: $\mathsf{Enc}(1^\lambda, x) \to K, \mathsf{ct}$
3: $\mathcal{A}(x, \mathsf{ct}) \to h$
4: ***return*** $\mathbb{1}_{h=K}$

In the random oracle model (ROM), the game starts by selecting a random hash function H *and* Enc, Dec*, and* $\mathcal{A}$ *are provided a secure oracle access to* H*. We define the advantage of* $\mathcal{A}$ *by*

$$\mathsf{Adv}^{\mathsf{OW\text{-}EWE}}_{\mathcal{A}}(x) = \Pr[\mathsf{OW\text{-}EWE}_{\mathcal{A}}(x) \to 1]$$

We say that WKEM *is extractable one-way for a set* X *of instances* x *if for any probabilistic and polynomial-time* OW-EWE *adversary* $\mathcal{A}$*, there exists a probabilistic and polynomial-time extractor* $\mathcal{E}$ *such that for all* $x \in X$*,* $\mathcal{E}(x)$ *outputs a witness of* x *with probability at least* $\mathsf{Adv}^{\mathsf{OW\text{-}EWE}}_{\mathcal{A}}(x)$ *or at least* $\frac{1}{2}$ *up to a negligible term:*

$$\forall x \in X \quad \Pr[R(x, \mathcal{E}(x))] \geq \min\left(\mathsf{Adv}^{\mathsf{OW\text{-}EWE}}_{\mathcal{A}}(x), \frac{1}{2}\right) - \mathsf{negl}(\lambda)$$

As for IND-EWE, we observe that we must have $\mathsf{Adv}^{\mathsf{OW\text{-}EWE}}_{\mathcal{A}}(x) = \mathsf{negl}(\lambda)$ when $x \in X - L$.

Indistinguishable implies one-way. As a warm-up, we show the easy result that extractable indistinguishable implies extractable one-way.

Theorem 7. *Let* $(\mathcal{K}_\lambda, \mathsf{Enc}, \mathsf{Dec})$ *be a* WKEM *for* L*. We assume that* $1/|\mathcal{K}_\lambda|$ *is negligible. If* WKEM *is extractable indistinguishable for a set* X *of instances* x*,* WKEM *is also extractable one-way for* X*.*

Proof. Let $\mathcal{A}$ be an OW-EWE adversary against WKEM. We first construct an IND-EWE adversary $\mathcal{B}$ against WKEM which simulates $\mathcal{A}$. The extractor coming from $\mathcal{B}$ will define an extractor for $\mathcal{A}$.

Adversary $\mathcal{B}(x, K_b, \mathsf{ct})$
1: $\mathcal{A}(x, \mathsf{ct}) \to K$
2: **return** $\mathbb{1}_{K_b=K}$

In the IND-EWE game, ct is generated in the same way as it is generated in OW-EWE game. The adversary $\mathcal{B}$ therefore simulates $\mathcal{A}$ with the same input distribution. When b is 1, the decapsulation of ct is K_b. Thus, $\mathcal{B}$ outputs 1 if $\mathcal{A}$ outputs the correct decapsulation of ct. We deduce

$$\Pr[\mathsf{IND\text{-}EWE}^1_{\mathcal{B}}(x) \to 1] = \Pr[\mathsf{OW\text{-}EWE}_{\mathcal{A}}(x) \to 1].$$

When b is 0, K_b is independent from K. So $\Pr[\mathsf{IND\text{-}EWE}^0_{\mathcal{B}}(x) \to 1] = 1/|\mathcal{K}_\lambda|$ which is assumed to be negligible. Then, we can deduce that the advantage of $\mathcal{B}$ in the IND-EWE game is

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IND\text{-}EWE}}_{\mathcal{B}}(x) &= \Pr[\mathsf{OW\text{-}EWE}_{\mathcal{A}}(x) \to 1] - \frac{1}{|\mathcal{K}_\lambda|} \\ &= \mathsf{Adv}^{\mathsf{OW\text{-}EWE}}_{\mathcal{A}}(x) - \mathsf{negl}(\lambda).\end{aligned}$$

We now assume that WKEM is IND-EWE-secure for X. There exists an extractor $\mathcal{E}$ which, for and $x \in X$, outputs a witness of x with probability at least $\mathsf{Adv}^{\mathsf{IND\text{-}EWE}}_{\mathcal{B}}(x)$ or $\frac{1}{2}$, up to a negligible term. We observe that $\mathcal{E}$ is also an extractor for the OW-EWE adversary $\mathcal{A}$. The probability for $\mathcal{E}$ to output ω is at least $\mathsf{Adv}^{\mathsf{OW\text{-}EWE}}_{\mathcal{A}}(x)$ or $\frac{1}{2}$ up to a negligible term.

Hence, WKEM is also OW-EWE-secure. □

Strongly secure from weakly secure transform. We now propose a generic WKEM transformation from OW-EWE-secure to IND-EWE-secure. The construction uses a random oracle. Let $\mathsf{WKEM}_0 = (\mathcal{K}^0_\lambda, \mathsf{Enc}_0, \mathsf{Dec}_0)$ be an OW-EWE-secure WKEM and H be a random oracle from $\mathcal{K}^0_\lambda$ to $\mathcal{K}_\lambda$. Our transformation is $\mathsf{WKEM} = (\mathcal{K}_\lambda, \mathsf{Enc}, \mathsf{Dec})$ as follows:

$\mathsf{Enc}(1^\lambda, x)$:
1: $\mathsf{Enc}_0(1^\lambda, x) \to h, \mathsf{ct}$
2: $K \leftarrow H(h)$
3: **return** K, ct

$\mathsf{Dec}(\omega, \mathsf{ct})$:
4: $\mathsf{Dec}_0(\omega, \mathsf{ct}) \to h$
5: $K \leftarrow H(h)$
6: **return** K

The intuition behind this transformation is the hardness of guessing an input to the random oracle H from the output.

Theorem 8. *If* WKEM_0 *is extractable one-way for a set* X *of instances, then* WKEM *from the above transformation is extractable indistinguishable for* X *in the random oracle model.*

Proof. Let H be a random oracle. Let $\mathcal{A}$ be a IND-EWE adversary for WKEM. Then, we will construct an extractor $\mathcal{E}$.

If H is a random oracle, in one execution of the IND-EWE game, we can define the event E_h that $\mathcal{A}$ queries H with h. We are interested in the h defined during the encryption in IND-EWE which is also a preimage of K_1. If ρ designates the random coins of $\mathcal{A}$ and responses designate the sequence of query responses from H, the probability distribution of $(x, K_0, \mathsf{ct}, \rho, \mathsf{responses})|\neg E_h$ and of $(x, K_1, \mathsf{ct}, \rho, \mathsf{responses})|\neg E_h$ are identical. Hence, $\mathsf{IND\text{-}EWE}^1_{\mathcal{A}}(x)$ and $\mathsf{IND\text{-}EWE}^0_{\mathcal{A}}(x)$ are identical games as long as E_h does not occur. We deduce that E_h occurs with the same probability when $b = 0$ and $b = 1$, and

$$\Pr[E_h] \geq \mathsf{Adv}^{\mathsf{IND\text{-}EWE}}_{\mathcal{A}}(x)$$

by the difference lemma [Sho04]. We construct the following OW-EWE adversary against WKEM_0:

$\mathcal{B}(x, \mathsf{ct})$:

1: pick $K_0 \in \mathcal{K}_\lambda$ at random
2: pick some random coins ρ
3: run $\mathcal{A}^H(x, K_0, \mathsf{ct}; \rho)$ and look at responses
4: pick i at random, $i \leq \#\mathsf{responses}$
5: $K \leftarrow i$th output of H in responses
6: **return** K

where $\#$responses is the number of oracle queries made by $\mathcal{A}$. This adversary in the OW-EWE game simulates perfectly the execution of the $\mathsf{IND\text{-}EWE}^0_{\mathcal{A}}$ game, in which E_h may occur. If E_h occurs and h is the ith input to H, then $\mathcal{B}(x, \mathsf{ct})$ outputs the correct decryption of ct. Hence, the advantage is at least $\frac{\Pr[E_h]}{\#\mathsf{responses}}$. We deduce

$$\mathsf{Adv}^{\mathsf{IND\text{-}EWE}}_{\mathcal{A}}(x) \leq \mathsf{Adv}^{\mathsf{OW\text{-}EWE}}_{\mathcal{B}}(x) \times \mathsf{Poly}(\lambda)$$

Due to OW-EWE security, there exists an extractor $\mathcal{E}(x)$ which extracts a witness for x with probability at least $\mathsf{Adv}^{\mathsf{OW\text{-}EWE}}_{\mathcal{B}}(x)$ or $\frac{1}{2}$. We can run the extractor a polynomial number of times and check if it outputs a valid witness. We can show that by iterating enough, either we extract with probability at least $\mathsf{Adv}^{\mathsf{IND\text{-}EWE}}_{\mathcal{A}}(x)$ or at least $\frac{1}{2}$. We can conclude. □

4.2 Hidden Group With Hashing

We define a new structure HiGH with correctness and security notions.

4.2.1 Definitions

We define the hidden group with hashing (HiGH) by some polynomially bounded algorithms.

Definition 10 (Hidden group with hashing)**.** *A hidden group with hashing (HiGH) in dimension d consists of the following algorithms:*

- $\mathsf{Gen}(1^\lambda) \to \mathsf{pgp}, \mathsf{tgp}$*: A probabilistic polynomial-time algorithm which generates at random some public group parameters* pgp *and some trapdoor group parameters* tgp*.*
- $\mathsf{Hom}(\mathsf{tgp}, x) \to y$*: A deterministic polynomial-time algorithm which maps $x \in \mathbb{Z}^d$ to y. We denote by G_{tgp} the set of all* $\mathsf{Hom}(\mathsf{tgp}, x)$*, for $x \in \mathbb{Z}^d$. When it is clear from context, we omit* tgp *and write* $\mathsf{Hom}(x)$ *and G.*
- $\mathsf{Mul}(\mathsf{pgp}, y, y') \to z$*: A deterministic polynomial-time algorithm which maps a pair (y, y') to a new element z. We denote by $S_{\mathsf{pgp},\mathsf{tgp}}$ the smallest superset of G_{tgp} which is stable by this operation. When it is clear from context, we omit* pgp *and write* $\mathsf{Mul}(y, y')$ *and S.*
- $\mathsf{Prehash}(\mathsf{pgp}, y) \to h$*: A deterministic polynomial-time algorithm which maps an element $y \in S$ to a "pre-hash".*[2] *When it is clear from context, we omit* pgp *and write* $\mathsf{Prehash}(y)$*.*

we define by induction

$$\mathsf{Pow}(y_1, \ldots, y_t, 1^{a_1}, \ldots, 1^{a_t}) = \mathsf{Mul}(\mathsf{Pow}(y_1, \ldots, y_t, 1^{a_1}, \ldots, 1^{a_t - 1}), y_t)$$

for $a_t > 0$ and

$$\mathsf{Pow}(y_1, \ldots, y_{t-1}, y_t, 1^{a_1}, \ldots, 1^{a_{t-1}}, 1^0) = \mathsf{Pow}(y_1, \ldots, y_{t-1}, 1^{a_1}, \ldots, 1^{a_{t-1}})$$

with $\mathsf{Pow}(y_1, \ldots, y_t, 1^0, \ldots, 1^0, 1^1) = y_t$*. In other words,* Pow *reduces a sequence of $\sum_{i=1}^{t} a_i$ elements $(\underbrace{y_1, \ldots, y_1}_{a_1 \text{ times}}, \ldots, \underbrace{y_t, \ldots, y_t}_{a_t \text{ times}})$ into a single element by using* Mul*.*

We write the a_i inputs to Pow *in unary to stress that the complexity is polynomial in terms of $\sum_i a_i$.*

These algorithms must be such that

- *they are all polynomially bounded;*

[2] We call it a *pre-hash* because our standard transformation to strongly-secure WKEM hashes it to make the final key.

- $\mathsf{Prehash}$ *is injective over* G*;*
- *for all* $\mathsf{Gen}(1^\lambda) \to \mathsf{pgp}, \mathsf{tgp}$*,* $x, x' \in \mathbb{Z}^d$*,* $y, y' \in S$

$$\left.\begin{matrix}\mathsf{Prehash}(y)=\mathsf{Prehash}(\mathsf{Hom}(x)) \\ \mathsf{Prehash}(y')=\mathsf{Prehash}(\mathsf{Hom}(x'))\end{matrix}\right\} \Longrightarrow \mathsf{Prehash}(\mathsf{Mul}(y,y'))=\mathsf{Prehash}(\mathsf{Hom}(x+x')) \tag{4.1}$$

Lemma 1. *Given a* HiGH*, we have the following properties.*

1. *For all* $\mathsf{Gen}(1^\lambda) \to \mathsf{pgp}, \mathsf{tgp}$*, for all* t*,* $(x_1, \ldots, x_t) \in (\mathbb{Z}^d)^t$*, and non-negative integers* $a_1, \ldots, a_t$*, if* $y_i = \mathsf{Hom}(x_i)$*,* $i = 1, \ldots, t$*, we have*

$$\mathsf{Prehash}\left(\mathsf{Pow}\left(y_1, \ldots, y_t, 1^{a_1}, \ldots, 1^{a_t}\right)\right) = \mathsf{Prehash}(\mathsf{Hom}(a_1x_1 + \cdots + a_tx_t))$$

2. $\mathsf{Prehash}(S) = \mathsf{Prehash}(G)$
3. *For any* $y \in S$*, there exists a unique* $z \in G$ *such that* $\mathsf{Prehash}(y) = \mathsf{Prehash}(z)$*. We call* z reduced *and we denote it by*

$$\mathit{Red}(y) = G \cap \mathsf{Prehash}^{-1}(\mathsf{Prehash}(y))$$

4. *The* $*$ *operation on* G *defined by* $y * y' = \mathit{Red}(\mathsf{Mul}(y, y'))$ *makes* G *an Abelian group and* Hom *a surjective group homomorphism from* $\mathbb{Z}^d$ *to* G*. We denote by* Ker *the kernel of* Hom*.*

Proof. We note that elements of the set S are constructed by making a finite sequence of Mul on G elements. We prove all properties in sequence.

1. This property is proven by induction using (4.1).
2. As $G \subseteq S$, we have $\mathsf{Prehash}(G) \subseteq \mathsf{Prehash}(S)$.

   Due to (4.1), we show by induction that for any $y \in S$, there exists at least one $z \in G$ such that $\mathsf{Prehash}(y) = \mathsf{Prehash}(z)$. Hence, $\mathsf{Prehash}(S) \subseteq \mathsf{Prehash}(G)$.

   We conclude that $\mathsf{Prehash}(S) = \mathsf{Prehash}(G)$.
3. As $\mathsf{Prehash}$ is injective on G, for any h, there exists no more than one element in $G \cap \mathsf{Prehash}^{-1}(h)$. The previous property shows that for $h = \mathsf{Prehash}(y)$, $y \in S$, there exists at least one. Hence, there is one and only one.
4. As Red maps to G, G is closed for the $*$ operation. Clearly, $\mathsf{Hom}(0)$ is a neutral element, due to (4.1). If $y = \mathsf{Hom}(x)$, then $\mathsf{Hom}(-x)$ is the inverse of y. Finally, if

$y = \mathsf{Hom}(x)$, $y' = \mathsf{Hom}(x')$, and $y'' = \mathsf{Hom}(x'')$,

$$\begin{aligned}(y * y') * y'' &= \mathsf{Red}(\mathsf{Mul}(y * y', y'')) \\ &= \mathsf{Red}(\mathsf{Mul}(\mathsf{Mul}(y, y'), y'')) \\ &= \mathsf{Hom}(x + x' + x'')\end{aligned}$$

and it is the same for $y * (y' * y'')$. So, $*$ is associative. Commutativity in G is inherited from the one in $\mathbb{Z}^d$.

□

Possible generalization. We could replace the domain $\mathbb{Z}^d$ of the x_i by any $\mathbb{Z}$-module. E.g., the domain of x_i could be any Abelian group: an elliptic curve, a lattice, etc.

Intuitive link to graded encoding. In HiGH, we want that running Pow is hard if some a_i are not small. We assume that there are sets $L_1, L_2, \ldots$ such that $S \subseteq L_1 \cup L_2 \cup \cdots$ and $G \subseteq L_1$. L_i is a set of "level-i encodings". Note that L_i's may be overlapping. We assume that if $y_1 \in L_i$ and $y_2 \in L_j$, then $\mathsf{Mul}(y_1, y_2) \in L_{i+j}$. Intuitively, the level of encoding corresponds to a level of complexity to process the elements, so that computing $\mathsf{Pow}(y_1, \ldots, y_t, 1^{a_1}, \ldots, 1^{a_t})$ has complexity which is directly linked to the sum of the a_i.

4.2.2 HiGH Knowledge Exponent Assumption (HiGH-KE)

Definition 11. *A* HiGH *satisfies the* HiGH *Knowledge Exponent Assumption for a set X if for any PPT algorithm $\mathcal{A}'$, there exists a PPT algorithm $\mathcal{E}'$ such that for all $x \in X$, the probability that the following game returns 1 is negligible:*

Game $\mathsf{HiGH\text{-}KE}(1^\lambda, x)$*:*

1: *parse* $x = (x_1, \ldots, x_t, \mathsf{aux})$
2: $\mathsf{Gen}(1^\lambda) \to (\mathsf{pgp}, \mathsf{tgp})$ ▷ *this defines* $G = \mathsf{Hom}_{\mathsf{tgp}}(\mathbb{Z}^d)$
3: $y_i \leftarrow \mathsf{Hom}(\mathsf{tgp}, x_i)$, $i = 1, \ldots, t$
4: *pick* ρ
5: $\mathcal{A}'(x_1, \ldots, x_t, y_1, \ldots, y_t, \mathsf{pgp}, \mathsf{aux}; \rho) \to h$
6: ***if*** $h \notin \mathsf{Prehash}(\mathsf{pgp}, G)$ ***then*** *abort*[3]
7: $\mathcal{E}'(x_1, \ldots, x_t, y_1, \ldots, y_t, \mathsf{pgp}, \mathsf{aux}, \rho) \to (1^{a_1}, \ldots, 1^{a_t})$ ▷ $a_i \in \{0,1,2,\ldots\}$
8: ***if*** $\mathsf{Prehash}(\mathsf{pgp}, \mathsf{Hom}(\mathsf{tgp}, a_1 x_1 + \cdots + a_t x_t)) = h$ ***then return*** *0*
9: ***return*** *1*

The point is that whenever the adversary succeeds to forge an element h of $\mathsf{Prehash}(G)$, the extractor, who has the same view, should almost always manage to express it as the

[3] We stress that this step may not be simulatable by a PPT algorithm.

Prehash of a combination of the known pairs (x_i, y_i), with small coefficients a_i. The nice thing about this assumption is that it allows to get similar results as in the generic group model (i.e., to extract the combination) by remaining in the standard model.

This assumption combines the preimage awareness of Prehash and the knowledge-of-exponent assumption in G.

By Prehash being *preimage aware*, we mean that whenever $\mathcal{A}$ succeeds to forge an element of $\mathsf{Prehash}(S)$ (which is also $\mathsf{Prehash}(G)$), then he must know some preimage in S.

Generic HiGH model. We model what a generic HiGH is and we prove HiGH-KE security in this idealistic model.

The objective of HiGH is to have a sparse subgroup G of a bigger group $\bar{G}$ so that it is hard to forge a G element except by manipulating known ones. Elements of $\bar{G}$ may be represented in many different ways having various "size" which impact the complexity of group operations. For that, a representation of an element of $\bar{G}$ is attached to an integer which we call its *level*, to keep the analogy with graded encoding. The generic model is relative to a probability distribution of a tuple $(\varphi, H, \bar{H})$, where φ is a group homomorphism from $\mathbb{Z}^d$ to $\bar{H}$ and $H = \varphi(\mathbb{Z}^d)$ is a subgroup of $\bar{H}$. (The H and $\bar{H}$ groups are the hidden versions of G and $\bar{G}$.) We define the following oracles.

Gen: (This oracle can only be called once.) Pick a random tuple $(\varphi, H, \bar{H})$, where $\varphi : \mathbb{Z}^d \to \bar{H}$ is a group homomorphism and $H = \varphi(\mathbb{Z}^d)$. Without loss of generality, write $\bar{H} = \{0, \ldots, \#\bar{H} - 1\}$ and denote by $\boxplus$ the group operation. Take $m = \lceil \log_2 \#\bar{H} \rceil$. For $y \in \mathbb{Z}_{2^m}$, denote $[y] = y \bmod \#\bar{H} \in \bar{H}$. For $i = 0, 1, 2, \ldots$, pick a random permutation π_i on $\mathbb{Z}_{2^m}$. Return $\mathsf{pgp} = m$ and tgp empty.

$\mathsf{Hom}(x)$: If x has negative components, abort. Return $(\pi_1(\varphi(x)), 1)$.

$\mathsf{Mul}((y, 1^i), (z, 1^j))$: If $i = 0$ or $j = 0$, abort. Otherwise, return

$$\mathsf{Mul}((y, 1^i), (z, 1^j)) = (\pi_{i+j}([\pi_i^{-1}(y)] \boxplus [\pi_j^{-1}(z)]), 1^{i+j})$$

$\mathsf{Prehash}(y, 1^i)$: Return $\pi_0([\pi_i^{-1}(y)])$.

To link with previous notations, we have $G = \{(\pi_1(y), 1); y \in H\}$, $S = \{(\pi_i(y), 1^i); y \in H, i > 0\}$, and $\bar{G} = \{(\pi_1(y), 1); y \in \bar{H}\}$.

Intuitively, elements $(y, 1^i)$ are represented by an m-bit number which is encoded by π_i so that the adversary does not see the real value. Elements are given with a "level of encoding" i which starts at $i = 1$. When we multiply a level-i encoding with a level-j encoding, we obtain a level-$(i + j)$ encoding. (We cannot multiply by a level-0 encoding.)

Again, encoding levels in inputs of Mul and Prehash are given in unary to stress that the level of encoding is polynomially bounded.

If the adversary picks an m-bit number y at a level i, it "represents" $[\pi_i^{-1}(y)]$ which is in $\bar{H}$. If H is sparse in $\bar{H}$, the adversary cannot forge a $(y, 1^i)$ such that $[\pi_i^{-1}(y)] \in H$, except by making combinations of known ones. The permutation π_0 is only used by Prehash.

Theorem 9. *For a distribution of $(\varphi, H, \bar{H})$ such that $\#H/\#\bar{H}$ is negligible in the above construction, the* HiGH-KE *assumption holds (for any set X).*

Proof. The oracles can define the permutations $\pi_0, \pi_1, \pi_2, \ldots$ by lazy sampling.[4] At the beginning of the game, the adversary $\mathcal{A}'$ receives the instance $x = (x_1, \ldots, x_t, s) \in (\mathbb{Z}^d)^{t+1}$, $y_i = \pi_i(\varphi(x_i))$, and m. During the game, $\mathcal{A}'$ uses the Mul and Prehash oracles which iteratively fill the tables defining the permutations. During the game, we let D be the set of all $(y, 1^i)$ with $y \in \mathbb{Z}_{2^m}$ and $i \geq 0$ for which $\pi_i^{-1}(y)$ is defined in the table of π_i. We say that D is the set of all *defined* encodings. Let $D' = \{\pi_i^{-1}(y); (y, 1^i) \in D\}$. When $\mathcal{A}'$ starts running in the game, we have $D = \{(y_i, 1); i = 1, \ldots, t\}$ and $D' = \{\varphi(x_i); i = 1, \ldots, t\}$.

We can define an extractor $\mathcal{E}'$ who runs $\mathcal{A}'$ and observes the interactions between $\mathcal{A}'$ and the Mul and Prehash oracles. The extractor keeps a list L which is initially $L = ((y_1, 1), \ldots, (y_t, 1))$. Elements of L are called the *free* elements. Whenever Mul or Prehash is queried with one (or two) input $(y, 1^i)$ which is not in D, the extractor appends $(y, 1^i)$ to the list L. Let $L = ((y_1, 1^{i_1}), \ldots, (y_{\#L}, 1^{i_{\#L}}))$.

If the adversary makes a query with $(y, 1^i)$ not in D, the value in $\pi_i^{-1}(y)$ is not defined yet. The probability that it becomes a $D' \cup H$ element is negligible. The probability that Mul returns an element of $D' \cup H$ is negligible too. This means that, except with negligible probability, only the t first elements from L represent an element in H.

By induction, except with negligible probability, the following properties are satisfied:

- for every $(y, 1^i) \in D$ with $i > 0$, the extractor knows $(a_1, \ldots, a_{\#L})$ with $a_1, \ldots, a_{\#L}$ non-negative such that

$$\begin{aligned} \mathsf{Prehash}(\mathsf{Pow}(L, 1^{a_1}, \ldots, 1^{a_{\#L}})) &= \mathsf{Prehash}(y, 1^i) \\ a_1 i_1 + \cdots + a_{\#L} i_{\#L} &= i \end{aligned}$$

- in this representation, either $a_{t+1} = \cdots = a_{\#L} = 0$ or $(y, 1^i)$ does not represent an element of H;
- for every $(h, 1^0) \in D$, either $(h, 1^0) \in L$ and $\pi_0^{-1}(h) \notin H$, or the extractor knows $(y, 1^i) \in D$ such that $h = \mathsf{Prehash}(y, 1^i)$ and $i > 0$.

[4] Lazy sampling is made by filling up, whenever needed, the tables for π_i which are initially empty.

Whenever $\mathcal{A}'$ returns $h \in \mathsf{Prehash}(G)$, the extractor deduces $(y, 1^i) \in D$ such that $h = \mathsf{Prehash}(y, 1^i)$ then $a_1, \ldots, a_t$ such that

$$h = \mathsf{Prehash}(\mathsf{Pow}(y_1, \ldots, y_t, 1^{a_1}, \ldots, 1^{a_t}))$$

Hence, $h = \mathsf{Prehash}(\mathsf{Hom}(a_1 x_1 + \cdots + a_t x_t))$. □

4.2.3 HiGH Kernel Assumption (HiGH-Ker)

Definition 12. *A* HiGH *satisfies the* HiGH *Kernel assumption for a set X of instances $x = (x_1, \ldots, x_{t+1})$ if for any PPT algorithm $\mathcal{A}''$, for any $x \in X$, the probability that the following game returns 1 is negligible.*

Game $\mathsf{HiGH\text{-}Ker}(1^\lambda, x)$*:*

1: *parse* $x = (x_1, \ldots, x_{t+1})$
2: $\mathsf{Gen}(1^\lambda) \to (\mathsf{pgp}, \mathsf{tgp})$
3: $y_i \leftarrow \mathsf{Hom}(\mathsf{tgp}, x_i)$, $i = 1, \ldots, t+1$
4: *run* $\mathcal{A}''(x_1, \ldots, x_{t+1}, y_1, \ldots, y_{t+1}, \mathsf{pgp}) \to z$
5: ***if*** $z = 0$ ***then*** *abort*
6: ***if*** $\mathsf{Prehash}(\mathsf{pgp}, \mathsf{Hom}(\mathsf{tgp}, z)) \neq \mathsf{Prehash}(\mathsf{pgp}, \mathsf{Hom}(\mathsf{tgp}, 0))$ ***then*** *abort*
7: ***return*** 1

This means that even with a few (x_i, y_i) pairs for Hom, it is hard to find a kernel element.

4.3 Our Instantiation of HiGH

We propose an instantiation of HiGH from the hardness of factorization of RSA modulus. Let $\mathsf{GenRSA}(1^\lambda)$ be an algorithm which outputs a tuple (n, ℓ, g) where n is an RSA modulus and g is an element of order ℓ in $\mathbb{Z}_n^*$. Our proposed instance is given in Fig. 4.1. Encryption generates a (n, ℓ, g, k) tuple such that g has multiplicative order ℓ modulo n and k is invertible modulo ℓ. The values k and ℓ are not revealed. They are used to derive a sequence $(k_1, k_1', \ldots, k_d, k_d')$ of elements of $\mathbb{Z}_\ell^*$ such that there is a hidden relation $k_i' = k_i^\theta \pmod{\ell}$.

The hidden group is $\bar{G} = (\mathbb{Z}_\ell^*)^2$ which has representation in $\mathbb{Z}^2$. We can $\mathsf{Prehash}$ with the basis-g exponential modulo n. We use the group

$$G = \{(k^{\alpha_1\xi_1 + \cdots + \alpha_d\xi_d}, k^{\theta(\alpha_1\xi_1 + \cdots + \alpha_d\xi_d)}) \bmod \ell; \xi_1, \ldots, \xi_d \in \mathbb{Z}\} \subseteq \bar{G}$$

Assuming that the α_i are relatively prime, we have $G = \{(k^\xi, k^{\theta\xi}) \bmod \ell; \xi \in \mathbb{Z}\}$. We have $\mathsf{Red}(\nu_1, \nu_2) = (\nu_1 \bmod \ell, \nu_2 \bmod \ell)$. The kernel is a subgroup Ker of $\mathbb{Z}^d$ of all $(\xi_1, \ldots, \xi_d)$ such that $\alpha_1\xi_1 + \cdots + \alpha_d\xi_d = 0$ modulo the order of k. $\mathsf{Prehash}$ is injective on $\mathbb{Z}_\ell^2$, so on G. G is the hidden group of the super-structure $S \subseteq \mathbb{Z}^2$. We stress that Mul makes

operations in the super-structure S of the hidden group G, and that Pow needs the a_i to be small because elements of $\mathbb{Z}^2$ can become huge.

$\underline{\mathsf{Gen}(1^\lambda)}$:

1: $\mathsf{GenRSA}(1^\lambda) \to (n, \ell, g)$ ▷ g of order ℓ in $\mathbb{Z}_n^*$

example

1. pick an RSA modulus ℓ of length λ
2. pick a random α such that $p = \alpha\ell + 1$ is prime
3. pick a random β such that $q = \beta\ell + 1$ is prime
4. set $n = pq$
5. pick g as a random number power $\alpha\beta$ modulo n until it has order ℓ

2: pick $k \in \mathbb{Z}_\ell^*$ at random
3: pick $\alpha_1, \ldots, \alpha_d, \theta \in \mathbb{Z}_{\varphi(\ell)}^*$ at random
4: $k_i \leftarrow k^{\alpha_i} \bmod \ell$, $i = 1, \ldots, d$
5: $k'_i \leftarrow k^{\theta\alpha_i} \bmod \ell$, $i = 1, \ldots, d$
6: $\mathsf{pgp} \leftarrow (n, g)$ ▷ public group parameters
7: $\mathsf{tgp} \leftarrow (\ell, (k_i, k'_i)_{i=1,\ldots,d})$ ▷ trapdoor group parameters
8: **return** $(\mathsf{pgp}, \mathsf{tgp})$

$\underline{\mathsf{Hom}(\mathsf{tgp}, \xi)}$:
9: parse $\mathsf{tgp} = (\ell, (k_i, k'_i)_{i=1,\ldots,d})$
10: parse $\xi = (\xi_1, \ldots, \xi_d)$
11: $\nu_1 \leftarrow k_1^{\xi_1} \cdots k_d^{\xi_d} \bmod \ell$
12: $\nu_2 \leftarrow {k'_1}^{\xi_1} \cdots {k'_d}^{\xi_d} \bmod \ell$
13: **return** (ν_1, ν_2)

$\underline{\mathsf{Mul}(\mathsf{pgp}, \nu, \nu')}$:
14: parse $\nu = (\nu_1, \nu_2)$
15: parse $\nu' = (\nu'_1, \nu'_2)$
16: $z_i \leftarrow \nu_i \nu'_i$, $i = 1, 2$
17: **return** (z_1, z_2)

$\underline{\mathsf{Prehash}(\mathsf{pgp}, \nu)}$:
18: parse $\mathsf{pgp} = (n, g)$
19: parse $\nu = (\nu_1, \nu_2)$
20: $h_i \leftarrow g^{\nu_i} \bmod n$, $i = 1, 2$
21: **return** (h_1, h_2)

Figure 4.1 – Our HiGH Construction

Vulnerability. We can see that if an adversary can implement an algorithm $\mathcal{A}(\xi)$, which outputs an integer, such that for a random $\xi \in \mathbb{Z}^d$, $|\mathcal{A}(\xi)| \leq B$ and $\mathcal{A}(\xi) \equiv k_1^{\xi_1} \cdots k_d^{\xi_d} \pmod{\ell}$ with probability p such that $1/p$ is polynomially bounded (we say that $\mathcal{A}$ weakly implements one component of Hom), then he can run it on random instances ξ, ξ', ξ'' with known relation $\xi'' = \xi + \xi'$ and obtain $\mathcal{A}(\xi)$, $\mathcal{A}(\xi')$, and $\mathcal{A}(\xi + \xi')$. With this, the

adversary can deduce a multiple of ℓ, then all k_i. It works the same for all k_i^θ. This means that the adversary obtains tgp. Hence, implementing $\xi \mapsto \mathsf{Hom}(\mathsf{tgp}, \xi)$ (even weakly) is equivalent to knowing tgp.

Similarly, given some (x_i, y_i) pairs with $y_i = \mathsf{Hom}(x_i)$, if an adversary finds some small[5] a'_i (positive or negative) such that $\sum_i a'_i x_i = 0$, then he can compute

$$\prod_i y_{i,j}^{\max(0,a'_i)} - \prod_i y_{i,j}^{\max(0,-a'_i)}$$

for $j = 1, 2$, which are multiples of ℓ. Their gcd ℓ' is a multiple of ℓ which can be used in its place. Then, the adversary can recover some (possibly big or negative) a''_i such that $\sum_i a''_i x_i = s$ and compute $z = \prod_i y_i^{a''_i} \bmod \ell'$ then $h = g^z \bmod n$.

This means that for the $\mathsf{HiGH\text{-}KE}$ to hold on X, there should be no $x \in X$ with a small relation $\sum_i a'_i x_i = 0$.

Efficient implementation. To reduce the size of tgp, we can replace the $(k_i, k'_i)_i$ family by a seed which generates $k, \alpha_1, \ldots, \alpha_d, \theta$.

Knowledge-of-exponent assumption. In our construction, $\mathsf{Prehash}(S)$ is included in the set of all $(g^{k^\xi} \bmod n, g^{k^{\theta\xi}} \bmod n)$, $\xi \in \mathbb{Z}$. This set has a structure and it seems hard, even when n, g, ℓ are known, to forge an element (h_1, h_2) without knowing a pair of integers (ν_1, ν_2) such that $h_i = g^{\nu_i} \bmod n$, $i = 1, 2$, and $(\nu_1, \nu_2) \bmod \ell \in G$.

The *knowledge-of-exponent assumption* on G says that whenever $\mathcal{A}$ succeeds to forge an element in S from some $(x_i, \mathsf{Hom}(x_i))$ pairs, then he must know some combination of those pairs which has the same Red. We add in our notion that the combination must be with small coefficients because the adversary does not know how to compute powers in S without having data blowing up.

More precisely, Wu and Stinson define the generalized knowledge-of-exponent assumption (GKEA) [WS07] over a group G. It says that for an adversary who gets $y_1, \ldots, y_t \in G$ and succeeds to produce $z \in G$, there must be an extractor who would, with the same view, make $a_1, \ldots, a_t \in \mathbb{Z}$ such that $z = y_1^{a_1} \times \cdots \times y_t^{a_t}$.

In our group $G = \{(k^\xi \bmod \ell, k^{\theta\xi} \bmod \ell); \xi \in \mathbb{Z}\}$, this assumption is usual, even when ℓ is known. In our settings, ℓ is not known but the x_i are known.

We conjecture that the $\mathsf{HiGH\text{-}KE}$ assumption holds in our construction for every $(x_1, \ldots, x_t)$ such that there is no relation $a_1 x_1 + \cdots + a_t x_t = 0$ with small a_i.

[5] By "small", we mean that computing $y_i^{|a_i|}$ over $\mathbb{Z}$ is doable.

Kernel assumption. The *RSA order assumption* says that given an RSA modulus ℓ and a random $k \in \mathbb{Z}_\ell^*$, it is hard to find a positive integer z' such that $k^{z'} \bmod \ell = 1$. The game is defined relative to the distribution P of ℓ as follows:

1: pick a random ℓ following P
2: pick a random $k \in \mathbb{Z}_\ell^*$ uniformly
3: run $\mathcal{B}(\ell, k) \to z'$
4: **if** $z' = 0$ or $k^{z'} \bmod \ell \neq 1$ **then** abort
5: **return** 1

For our construction, the HiGH-Ker problem is at least as hard as the RSA order problem.

Theorem 10. *Let ℓ be a random modulus following distribution P. Given an adversary $\mathcal{A}$ with advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HiGH\text{-}Ker}}$ in the* HiGH-Ker *game, we can construct an adversary $\mathcal{B}$ with same advantage (up to a negligible term) in the order problem with this modulus distribution, and similar complexity.*

Proof. We consider an adversary $\mathcal{A}$ playing the HiGH-Ker game with input x. We define an adversary $\mathcal{B}(\ell, k)$ playing the order game. The adversary $\mathcal{B}$ receives (ℓ, k) from the order game then simulates the rest of the HiGH-Ker game with $\mathcal{A}$. Then, there is a little problem to select $\alpha_1, \ldots, \alpha_d, \theta$ because $\mathcal{B}$ does not know $\varphi(\ell)$. However, by sampling in a domain which is large enough, the statistical distance between the real and simulated distributions of $(\mathsf{pgp}, \mathsf{tgp})$ is negligible. $\mathcal{A}$ may give some kernel elements $z = (z_1, \ldots, z_d)$ from which $\mathcal{B}$ can compute $\alpha_1 z_1 + \cdots + \alpha_d z_d$. The simulation of oracles in the generic HiGH model is straightforward. Hence, $\mathcal{B}$ succeeds in his game with (nearly) the same probability as $\mathcal{A}$ succeeds in the HiGH-Ker game.

The complexity overhead of $\mathcal{B}$ is only to run the simulation of the oracles with lazy sampling. □

4.4 WKEM from HiGH

We now construct a WKEM based on a HiGH and prove its security.

4.4.1 Construction

Our construction is based on a variation Multi-SS of the subset sum problem SS. We extend SS to the Multi-SS NP language defined by a polynomial P by:

> **Instance**: a tuple $x = (x_1, \ldots, x_t, s)$ of *vectors* of non-negative integers.
>
> **Witness**: a tuple $\omega = (1^{a_1}, \ldots, 1^{a_t})$ with non-negative integers a_i, $i = 1, \ldots, t$.
>
> **Predicate** $R(x, \omega)$: $a_1 x_1 + \cdots + a_t x_t = s$ and $a_i \leq P(|x|)$ for $i = 1, ..., t$.

It is not hard to see that Multi-SS is NP-complete. Actually, we provide a Karp reduction of SAT to Multi-SS in Section 4.5 in order to "program" a boolean satisfiability problem into Multi-SS.

We abstract our WKEM scheme with HiGH for the Multi-SS language on Fig. 4.2. Essentially, to encrypt with x, we generate a HiGH, we put all $y_i = \mathsf{Hom}(x_i)$ in the ciphertext, and the plaintext is

$$h = \mathsf{Prehash}(\mathsf{Hom}(s))$$

To decrypt with $\omega = (1^{a_1}, \ldots, 1^{a_t})$, we compute

$$h' = \mathsf{Prehash}(\mathsf{Pow}(y_1, \ldots, y_t, \omega)).$$

We first show that our construction is correct. Due to the correctness property (4.1) of HiGH, we have

$$\mathsf{Prehash}(\mathsf{Pow}(y_1, \ldots, y_t, \omega)) = \mathsf{Prehash}(\mathsf{Hom}(a_1 x_1 + \cdots + a_t x_t)).$$

If $R(x, \omega)$ holds, we have $a_1 x_1 + \cdots + a_t x_t = s$. We can then deduce that $h = h'$.

$\underline{\mathsf{Enc}(1^\lambda, x)}$:
1: parse $x = (x_1, \ldots, x_t, s)$
2: $\mathsf{Gen}(1^\lambda) \to (\mathsf{pgp}, \mathsf{tgp})$
3: $y_i \leftarrow \mathsf{Hom}(\mathsf{tgp}, x_i)$, $i = 1, \ldots, t$
4: $z \leftarrow \mathsf{Hom}(\mathsf{tgp}, s)$
5: $h \leftarrow \mathsf{Prehash}(\mathsf{pgp}, z)$
6: set $\mathsf{ct} = (y_1, \ldots, y_t, \mathsf{pgp})$
7: **return** h, ct

$\underline{\mathsf{Dec}(\omega, \mathsf{ct})}$:
8: parse $\mathsf{ct} = (y_1, \ldots, y_t, \mathsf{pgp})$
9: parse $\omega = (1^{a_1}, \ldots, 1^{a_t})$
10: $z' \leftarrow \mathsf{Pow}(\mathsf{pgp}, y_1, \ldots, y_t, \omega)$
11: $h' \leftarrow \mathsf{Prehash}(\mathsf{pgp}, z')$
12: **return** h'

Figure 4.2 – WKEM construction

We show in the next section that WKEM is an extractable one-way witness key encapsulation mechanism for instances of the NP language $L =$ Multi-SS.

Reusability of the parameters. The generated HiGH parameters $(\mathsf{pgp}, \mathsf{tgp})$ may require some computational effort in each encryption. This could be amortized by reusing some of the values. Namely, in our proposed HiGH, the parameters ℓ, n, g could be reused. As generating the parameters $(k, \alpha_1, \ldots, \alpha_d, \theta)$ requires no effort, it is advised not to reuse them. Indeed, reusing them would help an adversary to pool many (x_i, y_i) pairs in the very same structure. It is also nice not to store them as the dimension d can be very large. Then, finding a linear relation with small coefficients would become easier and

easier with the number of pairs.

However, using a long term ℓ is harming *forward secrecy* because disclosing it allows to decrypt all encryptions.

4.4.2 Security Proof

Now, we show that our construction is extractable one-way.

Theorem 11. *The* WKEM *construction for* Multi-SS *on Fig. 4.2 is extractable one-way for a set X of instances $x = (x_1, \ldots, x_t, s)$ if the underlying* HiGH *satisfies the* HiGH-KE *and the* HiGH-Ker *assumptions for X.*

Proof. Let $\mathcal{A}$ be an OW-EWE adversary. We first construct an algorithm $\mathcal{A}'$ for the HiGH-KE game in Section 4.1 as follows which receives a target s as an auxiliary input aux:

$\underline{\mathcal{A}'(x_1, \ldots, x_t, y_1, \ldots, y_t, \mathsf{pgp}, \mathsf{aux}; \rho)}$:
1: parse s from aux
2: $x \leftarrow (x_1, \ldots, x_t, s)$
3: $\mathsf{ct} \leftarrow (y_1, \ldots, y_t, \mathsf{pgp})$
4: $\mathcal{A}(x, \mathsf{ct}; \rho) \rightarrow h$
5: **return** h

Thanks to the HiGH-KE assumption, there exists an extractor $\mathcal{E}'$ making the HiGH-KE game return 1 with negligible probability for every $x \in X$. We then construct the extractor $\mathcal{E}$ as follows:

$\underline{\mathcal{E}(1^\lambda, x)}$:
1: parse $x = (x_1, \ldots, x_t, s)$
2: $\mathsf{Enc}(1^\lambda, x) \rightarrow (K, \mathsf{ct})$ ▷ K is not needed
3: parse $\mathsf{ct} = (y_1, \ldots, y_t, \mathsf{pgp})$
4: set aux to s
5: pick ρ at random
6: $\mathcal{E}'(x_1, \ldots, x_t, y_1, \ldots, y_t, \mathsf{pgp}, \mathsf{aux}, \rho) \rightarrow \omega$
7: **return** ω

Below, we detail the OW-EWE game (on the left) and the HiGH-KE game (on the right). To make the comparison easier, we expanded Enc and $\mathcal{A}'$ in gray in a line starting with a dot.

$\underline{\mathsf{OW\text{-}EWE}(1^\lambda, x)}$:

. parse $x = (x_1, \ldots, x_t, s)$
. $\mathsf{Gen}(1^\lambda) \to (\mathsf{pgp}, \mathsf{tgp})$
. $y_i \leftarrow \mathsf{Hom}(x_i)$, $i = 1, \ldots, t$
. $z \leftarrow \mathsf{Hom}(s)$
. $K \leftarrow \mathsf{Prehash}(z)$
. $\mathsf{ct} \leftarrow (y_1, \ldots, y_t, \mathsf{pgp})$
pick ρ at random
$\mathcal{A}(x, \mathsf{ct}; \rho) \to h$
return $\mathbb{1}_{h=K}$

$\underline{\mathsf{HiGH\text{-}KE}(1^\lambda, x_1, \ldots, x_t, s)}$:

parse $x = (x_1, \ldots, x_t, s)$
$\mathsf{Gen}(1^\lambda) \to (\mathsf{pgp}, \mathsf{tgp})$
$y_i \leftarrow \mathsf{Hom}(x_i)$, $i = 1, \ldots, t$
pick ρ at random
. $\mathsf{ct} \leftarrow (y_1, \ldots, y_t, \mathsf{pgp})$
. $\mathcal{A}(x, \mathsf{ct}; \rho) \to h$
if $h \notin \mathsf{Prehash}(G)$ **then** abort
$\mathcal{E}'(x_1, \ldots, x_t, y_1, \ldots, y_t, \mathsf{pgp}, s, \rho) \to (1^{a_1}, \ldots, 1^{a_t})$
if $\mathsf{Prehash}(\mathsf{Hom}(a_1x_1 + \cdots + a_tx_t)) = h$ **then return** 0
return 1

Clearly, everything until Step 7 is equivalent, with the same random coins. When OW-EWE returns 1, h is in $\mathsf{Prehash}(G)$ so HiGH-KE does not abort. Instead, it returns 0 or 1. We know that HiGH-KE returns 1 with negligible probability. Hence,

$$\Pr[\mathsf{HiGH\text{-}KE} \to 0] \geq \Pr[\mathsf{OW\text{-}EWE} \to 1] - \mathsf{negl}$$

Cases when HiGH-KE returns 0 are the one when $\mathcal{E}$ extracts successfully. Therefore, $\mathcal{E}$ extracts ω satisfying $\mathsf{Prehash}(\mathsf{Hom}(a_1x_1 + \cdots + a_tx_t)) = h$ with probability at least $\Pr[\mathsf{OW\text{-}EWE} \to 1] - \mathsf{negl}$. Due to the properties of HiGH, this implies that $a_1x_1 + \cdots + a_tx_t - s \in \mathsf{Ker}$.

We construct the following algorithm:

$\underline{\mathcal{A}''(x_1, \ldots, x_t, s, y_1, \ldots, y_t, z, \mathsf{pgp})}$:

set aux to s
pick ρ at random
$\mathcal{E}'(x_1, \ldots, x_t, y_1, \ldots, y_t, \mathsf{pgp}, \mathsf{aux}, \rho) \to \omega$
parse $\omega = (1^{a_1}, \ldots, 1^{a_t})$
return $a_1x_1 + \cdots + a_tx_t - s$

If we expand the HiGH-Ker game with the above algorithm, we can find that HiGH-Ker is similar to the OW-EWE extractor $\mathcal{E}$. The lines starting with a dot in gray (up to the $\mathcal{E}'$ call in $\mathcal{A}''$) correspond to $\mathcal{E}$ except the computation of K in Enc. In the extractor $\mathcal{E}$, K is never used after being computed from Enc. Therefore, the computation of K can be ignored without losing the equivalence.

$\underline{\mathsf{HiGH\text{-}Ker}(1^\lambda, x)}$:

. parse $x = (x_1, \ldots, x_t, s)$
. $\mathsf{Gen}(1^\lambda) \to (\mathsf{pgp}, \mathsf{tgp})$
. $y_i \leftarrow \mathsf{Hom}(x_i)$, $i = 1, \ldots, t$

4: . $z' \leftarrow \mathsf{Hom}(s)$
5: . run $\mathcal{A}''(x_1, \ldots, x_t, s, y_1, \ldots, y_t, z', \mathsf{pgp}) \to z$
6: **if** $z = 0$ **then** abort
7: **if** $\mathsf{Prehash}(\mathsf{pgp}, \mathsf{Hom}(\mathsf{tgp}, z)) \neq \mathsf{Prehash}(\mathsf{pgp}, \mathsf{Hom}(\mathsf{tgp}, 0))$ **then** abort
8: **return** 1

Due to the HiGH-Ker assumption, the probability that the game returns 1 is negligible. Then, the probability that $\mathcal{E}$ returns ω such that $a_1x_1 + \cdots + a_tx_t \neq s$ is negligible. Hence, we deduce that what $\mathcal{E}$ returns is actually such that $a_1x_1 + \cdots + a_tx_t = s$ with probability $\mathsf{Adv}^{\mathsf{OW\text{-}EWE}}_{\mathcal{A}}(x) - \mathsf{negl}(\lambda)$. □

4.5 Subset Sum Programming

The WKEM is supposed to be decapsulated by one who knows the corresponding witness of the Multi-SS language instance. We propose below a way to "program" some equations and constraints into a Multi-SS instance.

The first step is to express an equation in a Multi-SS way. Namely, it should be encoded into a linear equation (or several) with positive terms, the linear part on the left-hand side, the constant term on the right-hand side.

For instance, we "program" $b = a_1 \mathsf{XOR}\ a_2$ as follows

$$\left.\begin{array}{c} b = a_1 \mathsf{XOR}\ a_2 \\ a_1, a_2, b \in \{0,1\} \end{array}\right\} \leftrightarrow \left\{\begin{array}{rcl} a_1 + a_2 + b + 2c & = & 2 \\ b + \bar{b} & = & 1 \\ a_1 + \bar{a}_1 & = & 1 \\ a_2 + \bar{a}_2 & = & 1 \\ a_1, a_2, b, \bar{a}_1, \bar{a}_2, \bar{b}, c \in \{0,1,2,\ldots\} & & \end{array}\right.$$

Here, c appears as a necessary "garbage variable". The $\bar{b}$ variable is necessary to keep non-negative coefficients. The equations $a_i + \bar{a}_i = 1$ encode the constraint $a_i \in \{0,1\}$.

The boolean equation $b = a_1 \mathsf{NOR}\ a_2$ with constraint $a_1, a_2, b \in \{0,1\}$ can be encoded into

$$\begin{array}{rcl} a_1 + a_2 + c + 2\bar{b} & = & 2 \\ b + \bar{b} & = & 1 \\ c + \bar{c} & = & 1 \\ a_1 + \bar{a}_1 & = & 1 \\ a_2 + \bar{a}_2 & = & 1 \end{array}$$

with garbage c and additional variable $\bar{b}$.

Interestingly, the system $b_1 = a_1 \mathsf{XOR}\ a_2$ and $b_2 = a_1 \mathsf{NOR}\ a_2$ with constraints

$a_1, a_2, b_1, b_2 \in \{0,1\}$ can be encoded with no garbage variable into

$$\begin{aligned} a_1 + a_2 + b_1 + 2\bar{b}_2 &= 2 \\ b_1 + \bar{b}_1 &= 1 \\ b_2 + \bar{b}_2 &= 1 \\ a_1 + \bar{a}_1 &= 1 \\ a_2 + \bar{a}_2 &= 1 \end{aligned}$$

Any Boolean circuit can be formulated by equations with XOR, NOR, and NOT gates. If we have n_V Boolean variables, n_{NOT} NOT gates, n_{XOR} XOR gates, and n_{NOR} NOR gates, by taking that all variables will be negated, we obtain a system of

$$t = 2n_V - 2n_{\mathsf{NOT}} + n_{\mathsf{XOR}} + 2n_{\mathsf{NOR}}$$

variables ($n_V - n_{\mathsf{NOT}}$ non-negated variables, their negations, $n_{\mathsf{XOR}} + n_{\mathsf{NOR}}$ garbage variables, and the negation of the NOR garbage), $n_{\mathsf{XOR}} + n_{\mathsf{NOR}}$ equations of form $a_1 + a_2 + a_3 + 2a_4 = 2$, and $n_V - n_{\mathsf{NOT}} + n_{\mathsf{NOR}}$ equations of form $a_1 + a_2 = 1$. That is,

$$d = n_{\mathsf{XOR}} + 2n_{\mathsf{NOR}} + n_V - n_{\mathsf{NOT}}$$

equations.

Based on this programming technique, we can reduce the following SAT problem to Multi-SS to prove that Multi-SS is also NP-complete.

> **Instance**: a set of d equations of form $v_i = v_j * v_k$ for any Boolean operator $*$ and a set of t variables $v_1, \ldots, v_t$.
>
> **Witness**: a tuple $\omega = (a_1, \ldots, a_t)$ of bits $a_i \in \{0,1\}$, $i = 1, \ldots, t$.
>
> **Predicate** $R(x, \omega)$: if $v_i = a_i$ for $i = 1, \ldots, t$, then every equation is satisfied.

Theorem 12. *For any polynomial P such that for all $\ell \geq 1$ we have $P(\ell) \geq 1$,* Multi-SS *is NP-complete.*

Proof. We reduce SAT to Multi-SS. As all Boolean operators can solely be written with NOR gates, we assume without loss of generality that all SAT instances x have only equations of form $v_i = v_j$ NOR v_k.

We use our programming technique to express each SAT equation with 5 Multi-SS equations and 5 additional variables. In total, we obtain $d' = 2d + t$ equations with $t' = 2t + d$ variables. This is a Multi-SS instance $x' = \mathsf{Encode\text{-}Inst}(x)$. Clearly, x' has a Multi-SS solution if and only if x has a SAT solution.

Hence, SAT reduces to Multi-SS. Since SAT is NP-complete, so is Multi-SS. □

Example 1. *The system of equations*

$$\begin{aligned} u &= v \ \mathsf{NOR}\ w \\ v &= u \ \mathsf{XOR}\ w \\ w &= \mathsf{NOT}\ u \\ u, v, w &\in \{0,1\} \end{aligned}$$

has unique solution $(u, v, w) = (0, 1, 1)$*. If* u*,* v*,* w *are encoded into integers* a_1*,* a_2*,* a_3 *respectively, we can encode the equations into*

$$\begin{aligned} c_1 + 2a_1 + a_2 + a_3 &= 2 \\ a_2 + 2c_2 + a_1 + a_3 &= 2 \\ a_3 + a_1 &= 1 \\ a_2 + \bar{a}_2 &= 1 \\ c_1 + \bar{c}_1 &= 1 \\ a_1, a_2, a_3, c_1, c_2, \bar{a}_2, \bar{c}_1 &\in \{0,1\} \end{aligned}$$

where c_1 *and* c_2 *are garbage variables, leading us to*

$$\begin{pmatrix} 2 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 2 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ c_1 \\ c_2 \\ \bar{a}_2 \\ \bar{c}_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

The unique solution with non-negative coefficients is $(0, 1, 1, 0, 0, 0, 1)$*.*

In WKEM*, we write* $k_i = k^{\alpha_i} \bmod \ell$ *and* $k'_i = k_i^{\theta} \bmod \ell$*. The ciphertext is*

$$\mathsf{ct} = \left(n \ , \ g \ , \ \begin{matrix} k_1^2 k_2 k_3 \ , k_1 k_2 k_4 \ , k_1 k_2 k_3 \ , k_1 k_5 \ , \ k_2^2 \ , k_4 \ , k_5 \\ {k'_1}^2 k'_2 k'_3 \ , k'_1 k'_2 k'_4 \ , k'_1 k'_2 k'_3 \ , k'_1 k'_5 \ , {k'_2}^2 \ , k'_4 \ , k'_5 \end{matrix} \right)$$

(We omitted the $\bmod \ell$ *for simplicity.) The key is*

$$h = \mathsf{Prehash}\left(k_1^2 k_2^2 k_3 k_4 k_5, {k'_1}^2 {k'_2}^2 k'_3 k'_4 k'_5 \right)$$

Clearly, expressing it as a combination of the known values is equivalent to solving the Multi-SS *problem.*

However, this toy example also nicely illustrates the vulnerability we indicated: a non-zero kernel element of the equations would leak ℓ *and be insecure. If we write the equation* $Xa = s$ *in a matrix form, we can see that the vector* $a = (1, 0, -1, -1, 0, 0, 1)^T$ *is such*

that $Xa = 0$. *This translates into*

$$(k_1^2 k_2 k_3) \times (k_1 k_2 k_3)^{-1} \times (k_1 k_5)^{-1} \times (k_5) = 1$$

hence

$$(k_1^2 k_2 k_3) \times (k_5) - (k_1 k_2 k_3) \times (k_1 k_5) \equiv 0 \pmod{\ell}$$

4.6 Case Studies: Timed-Release Encryption from Bitcoins

Liu et al. [LJKW18] proposed a way to construct a time-lock encryption by using a witness encryption and the Bitcoin blockchain. The authors defined an NP-relation from the Bitcoin blockchain, namely the hardness of finding new blocks, and showed that we can construct a time-lock encryption scheme which uses the arrival of a new block in the Bitcoin blockchain as a reference clock. The time-lock encryption eventually allows a sender to encrypt a message which can be decrypted after some blocks are added to the blockchain.

However, the authors did not show how we can reduce it to the underlying NP problem of the witness encryption. In this chapter, we will show how to reduce it to Multi-SS in order to study its practical aspects.

We first recap the structure of the Bitcoin blockchain. According to the Bitcoin developer reference [bit], a block consists of two parts: a block header and transactions. The block header is always 80 bytes and consists of following values:

- ver: 4-byte integer which indicates the block version;
- prev_hash: 32-byte hash of the previous block header computed with SHA256-SHA256;
- merkle_root: 32-byte root hash of the Merkle tree derived from transactions;
- time: 4-byte unsigned integer which indicates the time that the miner started hashing the block header;
- nBits: 4-byte unsigned integer which indicates the difficulty target;
- nonce: 4-byte unsigned integer which can be arbitrary selected by the miner.

In order for a block to be added into the Bitcoin blockchain, the block is verified in several ways. We however focus on the verification of the hash of the block header. If a block is added into the Bitcoin blockchain, the SHA256-SHA256 hash of the block header should be smaller than nBits in the block header, i.e. the hash of the block header starts with a number of leading zero bits which is a function of the nBits field.

Let $B_i = (\mathsf{ver}_i, \mathsf{prev\_hash}_i, \mathsf{merkle\_root}_i, \mathsf{time}_i, \mathsf{nBits}_i, \mathsf{nonce}_i)$ be a Bitcoin block header. Then, the NP language on the Bitcoin blockchain is defined as follows:

Instance: a tuple $x = (t, \delta_1, \ldots, \delta_t, h)$ of non-negative integers.

Witness: a tuple $\omega = (B_1, \ldots, B_t)$ of t Bitcoin block headers $B_i = (\mathsf{ver}_i, \mathsf{prev\_hash}_i, \mathsf{merkle\_root}_i, \mathsf{time}_i, \mathsf{nBits}_i, \mathsf{nonce}_i)$.

Predicate $R(x, \omega)$: for $i = 1, \ldots, t$

$$\mathsf{prev\_hash}_1 = h$$

$$\mathsf{SHA256}(\mathsf{SHA256}(\mathsf{ver}_i \| \mathsf{prev\_hash}_i \| \mathsf{merkle\_root}_i \| \mathsf{time}_i \| \mathsf{nBits}_i \| \mathsf{nonce}_i)) = \mathsf{prev\_hash}_{i+1}$$

$$\mathsf{SHA256}(\mathsf{SHA256}(\mathsf{ver}_i \| \mathsf{prev\_hash}_i \| \mathsf{merkle\_root}_i \| \mathsf{time}_i \| \mathsf{nBits}_i \| \mathsf{nonce}_i)) \leq 2^{\delta_i}$$

As a new block is added to the Bitcoin blockchain approximately every 10 minutes, a corresponding witness of an NP instance will be found after around $10 \cdot t$ minutes if we use the hash of the block header of latest block of the Bitcoin blockchain as h.

We want to implement some subset sum programming for the predicate verification.

The SHA256 separates the input into chunks of 512-bit after padding the input, and then it iteratively applies the compression function, which takes a 256-bit hash value and a 512-bit chunk and outputs a 256-bit updated hash value, to each 512-bit chunk with an initial hash value. One block takes two chunks. Hence, we need *three compressions* to verify one block.

The SHA256 compression uses the two following functions:

$$\begin{aligned} \mathtt{Ch}(X, Y, Z) &= (X \wedge Y) \oplus (\neg X \wedge Z) \\ \mathtt{Ma}(X, Y, Z) &= (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z) \end{aligned}$$

A SHA256 compression starts by extending the input chunk. For that, it needs 48 additions of four 32-bit inputs and $48 \times 2 = 96$ XOR of three 32-bit inputs. Then, the main loop, with 64 iterations, uses for each iteration two 3-ary 32-bit XOR, one 32-bit `Ch`, one 32-bit `Ma`, one 5-ary 32-bit addition, and three 2-ary 32-bit additions. Finally, it terminates with eight 2-ary 32-bit additions. We do not count rotations and shifts as they come for free in subset sum programming.

Using the subset sum programming technique, every bit variable a must have a complement $\bar{a}$ which sum to 1 by design. A 3-ary XOR of a_1, a_2, a_3 bits can be programmed by the equation

$$a_1 + a_2 + a_3 + \bar{b} + 2c = 3 \qquad b + \bar{b} = 1 \tag{4.2}$$

with result b and garbage c. (No constraint needed on c.) Hence, it generates 3 output variables using 2 equations. A q-ary addition modulo 2^{32} of 32-bit inputs $a_i = \sum_{j=0}^{31} a_{i,j} 2^j$,

$i = 1, \ldots, q$, can be programmed by

$$\sum_{i=1}^{q}\sum_{j=0}^{31} a_{i,j}2^j + \sum_{j=0}^{31} \bar{b}_j 2^j + 2^{32}c = 2^{32+\lceil \log_2 q \rceil} - 1 \qquad \begin{matrix} b_j + \bar{b}_j = 1 \\ 0 \le j \le 31 \end{matrix} \tag{4.3}$$

with a garbage variable c which can be any non-negative integer (hence with no additional constraint). It generates 65 output variables using 33 equations. By enumerating all possibilities, we can prove that $\texttt{Ch}(a_1, a_2, a_3)$ can be programmed by

$$5a_1 + 6a_2 + a_3 + 2b + c_1 + 4c_2 + 10c_3 = 14 \qquad b + \bar{b} = 1 \qquad \begin{matrix} c_1 + \bar{c}_1 = 1 \\ c_2 + \bar{c}_2 = 2 \end{matrix} \tag{4.4}$$

with result b and boolean garbage c_1, c_2, c_3, c_4. It generates 7 output variables using 4 equations. By enumerating all possibilities, we can prove that $\texttt{Ma}(a_1, a_2, a_3)$ can be programmed by

$$a_1 + a_2 + a_3 + 2\bar{b} + c = 3 \qquad b + \bar{b} = 1 \qquad c + \bar{c} = 1 \tag{4.5}$$

with result b and garbage c. It generates 4 output variables using 3 equations.

Table 4.1 – SS instances and corresponding operations

Number	Operation	Ref.	#variables	#equations
48	$b = a_1 + a_2 + a_3 + a_4 \bmod 2^{32}$	Eq. (4.3)	65	33
$48 \times 2 \times 32$	$b = a_1 \oplus a_2 \oplus a_3$	Eq. (4.2)	3	2
$64 \times 2 \times 32$	$b = a_1 \oplus a_2 \oplus a_3$	Eq. (4.2)	3	2
64×32	$b = \texttt{Ch}(a_1, a_2, a_3)$	Eq. (4.4)	7	4
64×32	$b = \texttt{Ma}(a_1, a_2, a_3)$	Eq. (4.5)	4	3
64	$b = a_1 + a_2 + a_3 + a_4 + a_5 \bmod 2^{32}$	Eq. (4.3)	65	33
64×3	$b = a_1 + a_2 \bmod 2^{32}$	Eq. (4.3)	65	33
8	$b = a_1 + a_2 \bmod 2^{32}$	Eq. (4.3)	65	33

Table 4.1 summarizes the number of equations and added variables. Hence, one compression generates 64 312 output variables with 38 968 equations. We deduce that the length-t blockchain verification generates $192\,936t$ variables with $116\,904t$ equations. As we have $640t$ input variables, we obtain a Multi-SS instance with $t' = 193\,576t$ variables and $d = 116\,904t$ equations. Equations are quite sparse, so easy to store.

We assume that the RSA modulus ℓ is of 2048 bits and that a modulo ℓ power of size $|\ell|$ takes 1.4 ms on a single core.

Using a witness for one hour duration requires $t = 6$. Hence, we use about 1.2 Million variables and 700 000 equations. As equations have a constant number of non-zero terms, to encrypt requires to compute several millions of linear combinations and $2(t'+1)$ powers of k modulo ℓ. The ciphertext will take $2t'|\ell| \approx 567$ MB. To decrypt requires to compute t' integer multiplications, followed by two big exponentials modulo n with exponent of

size $t'|\ell|$. We can estimate the time to 27 core.minutes.

Using a witness for one year duration requires t of order 50 000. Hence, about 9.7 Billion variables and 5.8 Billion equations. The ciphertext is now of 4.5 TB. Decryption will take 5.2 core.months.

We can clearly do better with blockchains which would be optimized for this purpose. For instance, by having a block structure of form $\mathsf{block}_i = (\mathsf{prev\_hash}_i, \mathsf{block\_hash}_i)$ where $\mathsf{block\_hash}_i$ contains the hash of everything but $\mathsf{prev\_hash}_i$, we now have only one compression to compute per block instead of three for verification. We improve performance by a factor 3. We can also suggest alternate hash functions which would have a subset sum programming which is more efficient. We can hope for another improvement factor of 3. Finally, we can also enrich the linear chaining scheme of blocks with shortcuts. If we want to protect against an adversary who processes 1% of the computational power of bitcoin miners, we can shortcut the chain by chunks of 100 blocks. We improve performance by another factor 100. Finally, we reduce the ciphertext to 5 GB and the decryption complexity to 4 core.hours for a witness of 1 year.

We can further investigate hash functions design which are adapted to subset sum programming. There are hash functions based on ring-LWE [LPR10, PR06]. We consider a ring $R = \mathbb{Z}_p[\theta]/P(\theta)$ where P is a polynomial of degree n. The hash function is defined by a random (public) key $(x_1, \ldots, x_m) \in R^m$. To hash a list $(a_1, \ldots, a_t)$ of integers such that $0 \leq a_i < 2B$ $i = 1, \ldots, t$ and $t \leq nm$, we form $y_j = \sum_{i=0}^{n-1}(a_{jn+i-n+1} - B)\theta^i$ and output $\sum_{j=1}^{m} x_j y_j \bmod P(\theta) \bmod p$, to be written in basis $2B$. We can program this into

$$\left[\sum_{i=0}^{n-1}\sum_{j=1}^{m} x_j\theta^i(a_{jn+i-n+1} - B) \bmod P(\theta)\right]_{\theta^k} = \sum_{\ell=0}^{\lceil \log_{2B} p\rceil - 1} (b_{k,\ell} - B)(2B)^\ell + pc_k$$

for $k = 0, \ldots, n-1$ and $b_{k,\ell} + \bar{b}_{k,\ell} = 2B$. Above, we denote by $[\cdot]_{\theta^k}$ the coefficient of θ^k of a polynomial in θ. Hence, we have $n + n\lceil \log_{2B} p\rceil$ equations with $n + 2n\lceil \log_{2B} p\rceil$ output variables.

One suggested parameter vector is $n = 126$, $m = B = 8$, $p \approx 2^{23}$ [LPR10]. One compression can compress $mn \log_2(2B) = 4032$ bits and produce $n \log_2 p = 2898$ bits. We obtain 882 equations producing 1638 variables. If $\mathsf{prev\_hash}$ has 2898 bits, the rest of the bitcoin block fits within 4032 bits and we can verify one block with one compression. Thus, we need $d = 882t$ equations and $t' = 1638t$ variables to verify a chain of t blocks. Hence, for $t = 6$, we have a ciphertext of $2t'|\ell| \approx 4.8$ MB and decryption needs 14 core.seconds. For $t = 50\,000$, we have a ciphertext of 20 GB and decryption needs 32 core.hours. With shortcuts every 100 blocks, this becomes 200 MB and 19 core.minutes, respectively. This is a total improvement factor of 20 000.

4.7 Circuit to System of Equations

In the previous section, we have shown that we can transform SHA-256 into a system of equations. In this section, we will show how to do the same for an arbitrary circuit. As we have already shown that we can represent a NOR operation in an equation, it is clear that we can represent any circuit as a system of equations. However, it may not necessarily be efficient in terms of number of variables, as we need a garbage variable in the system for every NOR operation. So, we require a way to transform a small circuit to a system of equations and at the same time reduce the number of garbage variables in the system.

Let $f(b_1, \ldots, b_n) = b'_1, \ldots, b'_m$ be a circuit which takes an n-bit input and produces an m-bit output. When we transform it into a system of equations, we already know that we need at least n variables for input bits and m variables for output bits, but the number k of garbage variables changes depending on the circuit. After the transformation, we will get several equations $a_1x_1 + \cdots a_{n+m+k}x_{n+m+k} = s$ where $(a_1, \ldots, a_n)$ correspond to input variables $(b_1, \ldots, b_n)$, $(a_{n+1}, \ldots, a_{n+m})$ correspond to output variables $(b'_1, \ldots, b'_m)$, and $(a_{n+m+1}, \ldots, a_{n+m+k})$ correspond to garbage variables for some k. We therefore need an efficient algorithm to find $x_1, \ldots, x_{n+m+k}, s$ which implement the given circuit.

More formally, we want to construct a system of affine equations in $a_1, \ldots, a_{n+m+k}$ such that:

- for any solution $a_1, \ldots, a_{n+m+k}$ of non-negative integers, we have $a_1, \ldots, a_{n+m}$ in $\{0, 1\}$;
- for any $a_1, \ldots, a_{n+m}$ in $\{0, 1\}$, $(a_{n+1}, \ldots, a_{n+m}) = f(a_1, \ldots, a_n)$ if and only if there exists some non-negative integers $a_{n+m+1}, \ldots, a_{n+m+k}$ such that $a_1, \ldots, a_{n+m+k}$ is a solution.

We first show that an arbitrary circuit f, with n-bit input and m-bit output, can be implemented in $n + m + 1$ equations with $k = 2^n + n + m$ garbage variables. As circuit f is deterministic, there exist 2^n input/output pairs. Then, we can construct the following system of equations:

$$\begin{cases} a_1x_1 + \cdots + a_{n+m+2^n}x_{n+m+2^n} = s \\ a_i + a_{n+m+2^n+i} = 1 \text{ for } i = 1, ..., n + m \end{cases}$$

where

- $x_i \leftarrow 2^{i-1}$, for $i \in \{1, \ldots, n + m\}$
- $x_t \leftarrow 2^{n+m+1} - \sum_{j=1}^{n+m} b_j x_j$, where $t = n+m+1+\sum_{i=1}^{n} 2^{i-1}b_i$ for $b_1, \ldots, b_n \in \{0, 1\}$ and $(b_{n+1}, \ldots, b_{n+m}) = f(b_1, \ldots, b_n)$

- $s \leftarrow 2^{n+m+1}$

By construction, we have $\sum_{i=1}^{n+m} b_i x_i < 2^{n+m}$ for any $b_1, \ldots, b_{n+m} \in \{0,1\}$. This implies $x_{n+m+i} > 2^{n+m}$ for $i \in \{1, \ldots, 2^n\}$. Due to $a_i + a_{n+m+2^n+i} = 1$, $a_1, \ldots, a_{n+m} \in \{0,1\}$. We deduce

$$\begin{aligned}
& a_1 x_1 + \cdots + a_{n+m+2^n} x_{n+m+2^n} = s \\
\Leftrightarrow\ & \underbrace{a_1 x_1 + \cdots + a_{n+m} x_{n+m}}_{<2^{n+m}} + a_{n+m+1} x_{n+m+1} + \cdots + a_{n+m+2^n} x_{n+m+2^n} = 2^{n+m+1} \\
\Rightarrow\ & 2^{n+m} < a_{n+m+1} \underbrace{x_{n+m+1}}_{>2^{n+m}} + \cdots + a_{n+m+2^n} \underbrace{x_{n+m+2^n}}_{>2^{n+m}} \leq 2^{n+m+1} \\
\Rightarrow\ & \sum_{i=1}^{2^n} a_{n+m+i} = 1.
\end{aligned}$$

For any solution of the system of equations, there always exists exactly one non-zero value in $a_{n+m+1}, \ldots, a_{n+m+2^n}$. Assume that $a_{n+m+i} = 1$ for some $i \in \{1, \ldots, 2^n\}$. As the solution should satisfy the equality $a_1 x_1 + \cdots + a_{n+m+2^n} x_{n+m+2^n} = s$, we have

$$\begin{aligned}
& a_1 x_1 + \cdots + a_{n+m+2^n} x_{n+m+2^n} = s \\
\Leftrightarrow\ & a_1 x_1 + \cdots + a_{n+m} x_{n+m} + x_{n+m+i} = s \\
\Leftrightarrow\ & a_1 x_1 + \cdots + a_{n+m} x_{n+m} + 2^{n+m+1} - \sum_{j=1}^{n+m} b_j x_j = s \\
& \qquad \left(\text{where } i - 1 = \sum_{j=1}^{n} 2^{j-1} b_j \text{ and } b_{n+1}, \ldots, b_{n+m} = f(b_1, \ldots, b_n)\right) \\
\Leftrightarrow\ & a_1 x_1 + \cdots + a_{n+m} x_{n+m} = \sum_{j=1}^{n+m} b_j x_j
\end{aligned}$$

Thus, $a_1, \ldots, a_{n+m}$ satisfy $a_{n+1}, \ldots, a_{n+m} = f(a_1, \ldots, a_n)$. As all input/output pairs of f are used for $x_{n+m}, \ldots, x_{n+m+2^n}$, for any $a_1, \ldots, a_{n+m} \in \{0,1\}$ such that $a_{n+1}, \ldots, a_{n+m} = f(a_1, \ldots, a_n)$, there exists an index i where the given $a_1, \ldots, a_{n+m}$ and $a_{n+m+k} = 1$ is a solution of the system of equations. Hence, the system of equations is equivalent to the circuit f.

In practice, we can possibly find better transformation in terms of number of garbage variables and s. We could find better implementation by bruteforcing on $(x_1, \ldots, x_{n+m+k}, s)$ after fixing some value for k, and we found some interesting observations.

Firstly, the minimal number k of garbage variables is upper bounded by $n + m$.

Moreover, if there exist two input values such that swapping these input values do not change the output, i.e. there exist i, j in $\{1, \ldots, n\}$ such that $i < j$ and

$$f(b_1, \ldots, b_i, \ldots, b_j, \ldots, b_n) = f(b_1, \ldots, b_j, \ldots, b_i, \ldots, b_n)$$

for every $b_1, \ldots, b_n \in \{0, 1\}$, the minimal system of equations has $x_i = x_j$.

From these observations, we can expect that the hash computation in the Bitcoin block chain can be transformed into a system of equations with 3072 variables (1280 input variables, 256 output variables and 1536 garbage variables). However, it is not easy to find the actual system of equations because it requires to compute hashes for all possible inputs.

4.8 Conclusion of Chapter

We have shown how to construct a WKEM for a variant of the subset sum problem, based on HiGH. This is secure in the generic HiGH model. We proposed an HiGH construction based on RSA which has a restriction on the subset sum instances. One open question is to make it work even for instances having a small linear combination which vanishes. We can use this WKEM for timed-release encryption using blockchains. However, using it in practice may need further optimizations, including in the blockchain infrastructure.

Another interesting challenge is to build a post-quantum HiGH.

5 Self-Encryption

In many deployed applications, the design of the application involves various devices communicating with each other securely. They sometimes require one of the devices to encrypt some piece of information that will be used in the future by itself. We call it *self-encryption*: encrypting data which will be decrypted by the same entity generating it. One application is massive client-server connections where there are millions of clients connecting to a server, causing the server not being able to afford to store any client-specific information. On the other hand, recent protocols such as TLS 1.3 empowers the server to resume past sessions without going through a new round-trip handshake when a client reconnects to the server.[1] While clients, surely, would benefit from a smooth connection experience, the server has to "remember" each session in a secure manner, possibly by keeping a (small or big) size of state. More precisely, when a client connects to a website for the first time, the web server generates a cookie including a ticket for the client. This ticket is a piece of information that helps the server to remember the session. Somehow, this is a helper that the server encrypts for itself and gives to the client to store locally. When the client reconnects to the same website with her ticket, the server can resume their previous session by decrypting the ticket. As desired, it gives the freedom not to store any client-specific information on the server-side. However, the server needs a secret state for the cryptographic operations which are used in generating and decrypting tickets. From the security point of view, then, the concern becomes to provide security against replay attacks or occasional exposures of the internal state of the server.

In general, the internal state is any type of information that would let a device decrypt (some part of) the communication. In this work, we investigate the security of a self-encryption state which comes in two forms: forward secrecy (FS) and post-compromise security (PCS). Intuitively, forward secrecy provides security for the *past* communication when exposure happens, whereas post-compromise security aims to heal the *future* communication when exposure occurs [CGCG16]. Before going forward with security, we list three applications in different settings with different functionality where the security

[1] As of November 2019, 34% of TLS connections use session resumption [HARV19].

of self-encryption would be critical.

0-RTT in TLS 1.3. In the TLS 1.3 protocol, a client connects to a server and establishes a common secret key through a handshake key agreement protocol. This is succeeded with a full round trip time (1-RTT) communication. Ideally, when the client reconnects to the same server after a while, the connection should be resumed with no round trip time (0-RTT). Although 0-RTT has been an active research domain in the last few years, currently, in practice, it is achieved through two elementary approaches called *Session Caches* and *Session Tickets* as described very thoroughly by Aviram, Gellert, and Jager (AGJ) [AGJ19]. In the former technique, the server resumes the session by assigning each client a different resumption key for each connection and sending the client a look-up index that links to the resumption key. The ticket is that index. When the client comes back, it includes the ticket and the payload data. This provides forward secrecy. Nevertheless, the solution depends on maintaining a big database on the server, which is not alluring.

The other approach for 0-RTT in TLS 1.3 configurations is to create session tickets for each client by using a long-term secret key K (the ticket encryption key). Therefore, instead of storing a unique key for each session, the server generates a secret material for each client and encrypts it under K. The secret material is called resumption key whereas the encrypted resumption key is the ticket. The client stores both the resumption key and the ticket. Later on, the client encrypts the payload with the resumption key and includes her ticket in 0-RTT message to remind herself. The server can clearly decrypt the ticket with K and retrieve the resumption secret to decrypt the payload. This approach surely avoids storing a big database, it is easy to implement and integrate in existing systems, yet, it does not provide any kind of security in the case of a key exposure.[2]

In their recent work, Aviram, Gellert, and Jager (AGJ) [AGJ19] studied the forward secrecy and the resistance to replay attacks of session resumption, specifically focusing on session tickets. However, they did not consider PCS in their security model which is the main focus of the present work.

Cloud storage. In a single client-server cloud storage, the client wants to outsource her files in a remote storage (cloud) in an encrypted form. The encryption of the files occur locally on the client who keeps the secret decryption material. If the client encrypts all files with the same key, the leakage of the key becomes catastrophic as all files (even the removed ones) become compromised. Besides, the client aims to minimize the storage on her local while maintaining strong security in case of a compromise of her internal state. This cloud storage problem is the same as the 0-RTT problem: the cloud client and the 0-RTT server do not want to store any file-specific information while conserving security.

[2] In TLS 1.3, the long-term key K is updated every few hours by assuming that all the clients will resume their sessions in the "life-time" of K. Nevertheless, as soon as the key K is compromised, there is neither FS nor PCS for the period where K is active.

On the other hand, regulations mandate the encrypted files to be updated from an old key to a new key often enough. This is called *key rotation*. It is part of a common good practice in key management. System admins should not let keys age too much nor be used more than what the encryption method can guarantee to be secure. The fundamental motivation, however, comes with the desire to achieve resilience to key exposure. Besides encryption and decryption, key rotation uses two other algorithms: one computing a key update token from an old key, a new key, and a ciphertext header, and one computing the new ciphertext from the old one and the key update token. Essentially, the adversary, who sees some ciphertexts and obtains some keys as well as some key update tokens, should not be able to decrypt, except what is trivially implied by correctness. Key rotation was formally studied by Boneh et al. [BLMR13]. More recently, Everspaugh et al. [EPRS17] considered the integrity problem with key rotation.

The naive way to achieve key rotation would be that the client downloads the encrypted files on the local, decrypts them with the existing key, generates a new fresh key and updates the old key, re-encrypts and finally outsources back. However, it is a very cumbersome solution for the client. The main task of key rotation is to avoid the complexity of communication and the complexity of treatment on the client side. In practice, AWS and Google deploy a more practical methods based on hybrid encryption: a header is formed by encrypting an ephemeral key and the rest of the ciphertext is formed by encrypting the plaintext with the ephemeral key. Key rotation is done by updating the header but keeping the same ephemeral key. This was argued to be a bit cheating with the concept of key rotation as the encryption of data under the same key was remaining.

We tackle the privacy problem differently. Instead of updating a ciphertext to be decryptable with a chosen key, we let ciphertexts unchanged but update the state which is stored by the client[3]. Naturally, our concern becomes more focused on the storage space on the client side. Key rotation assumes that a client who has stored n files would keep all k keys which are necessary to decrypt these files. And as soon as a key needs to be rotated, some operation would be required for all ciphertexts using this key, hence $\frac{n}{k}$ on average. It means that the number k of stored keys (memory complexity) multiplied by the number of ciphertext updates is the number of stored files which is n. In our setting, the client stores one state (which is shorter than storing n keys) and needs no operation on ciphertexts.

Instant messaging. Post-compromise security in instant messaging was formally studied during the last few years [PR18, JS18, JMM19, ACD19, DV19]. Bidirectional secure communication applications can be seen as a particular form of self-encryption. In fact, roughly speaking, we can merge both participants into one single device which would encrypt for itself. A *ratcheted* scheme is normally FS and PCS secure, hence defines an

[3]We do not mean to pick a fresh key to "rotate" the key and update the header as practiced by AWS.

FS and PCS secure self encryption which we call a *self-ratchet*.

Our Perspective. In order to study the security of self-encryption, we consider a scheme which generates ciphertexts with the ability to decrypt later, even when the state to decrypt evolves. We define it in a way that it covers the three (and potentially more) applications we described earlier. Furthermore, we are interested in forward secrecy and post-compromise security of these systems. The former captures that the system generates ciphertexts that should remain decryptable for a limited time and that are not going to be decryptable anymore after they "expire" (it could happen either because the settings allow the ciphertexts to stay alive for a limited time or because the keys are "punctured" on purpose). The ciphertexts that are still decryptable are called *active ciphertexts*. Making a ciphertext become inactive is a way to have forward secrecy: if the state of the scheme is exposed after a ciphertext becomes inactive, this ciphertext is still safe. The PCS defines what happens to the security *after* an exposure of a state. When an exposure takes place, the post-compromise secure system should be able to heal the state such that the ciphertexts after the healing are secure. In many studies, PCS is interchangeably used with *healing*.

While studying self-ratcheted schemes with PCS guarantees (as well as FS), it was intuitive to expect that the state size of any post-compromise secure self-ratcheted scheme will grow. However, it was not clear why and with what bounds we could achieve it. The first and primary contribution of our work is to show that we cannot achieve post-compromise security better than adding a trivial solution to already existing efficient forward secure schemes.

As for forward security, AGJ [AGJ19] specifically consider the session resumption in TLS 1.3 and they designed solutions for FS and replay attacks without providing any PCS. Their construction is practical. In another study by Günther et al. [GHJL17] and Derler et al. [DJSS18], the authors consider a slightly different solution to the session resumption with forward secrecy. In these works, the motivation is to let the clients resume connections without having to store any session-specific information on her local. The authors decided to do so by letting the client keep only the long-term public key pk of the server. Therefore, they look for forward-secure solutions when the long term secret key sk evolves throughout time although the associated public key never changes, hence the clients never updates its state. Although it is remarkable that such schemes with forward secrecy exist, both constructions are far from being practical due to the heavy cryptographic tools they use. Therefore, we rather focus on the FS scheme AGJ to add PCS.

Our contribution

In the present work, we start with the definition of a minimal primitive called Self-Encrypted Queue (SEQ) with correctness and one-way (OW) security. It gives the minimal functionality for any PCS construction, more particularly self-encryption schemes. Then, we prove that for every SEQ primitive with states of bounded length, there is an adversary with small complexity and high probability of success to break OW security. This result led us to conclude that when self-encryption is post-compromise secure, it must have a state which grows more than linearly in the number of active ciphertexts, say n.[4] It shows that we can achieve post-compromise security *only* with a state size larger than linear in n. This does not provide the practicality we were hoping for. Therefore, we define a refinement which is a relaxed version of post-compromise security. In layman terms, we look into the following case: Maybe the first ciphertext that will be generated after an exposure is not secure, but the system could be designed to heal the security after the generation of Δ ciphertexts, where Δ is a constant parameter of our scheme. We call it Δ-PCS. We show that in refined definitions, the state size is super-linear in $\frac{n}{\Delta}$ as opposed to growing super-linear in n.

We prove that this impossibility result applies both in self-encryption and in secure messaging. In addition to this, we prove that this result is tight by constructing a simple self-encryption scheme achieving Δ-PCS with a state size matching our bounds.

After our impossibility results, we focus on few applications by borrowing already existing formal interfaces from AGJ [AGJ19] in order to add PCS security in the discussed settings. We modify the interface in a way that decryption and puncturing happens with separate function calls in case the puncturing is not always necessary. Later on, we look at secure ratcheted protocols which provides PCS security from the literature. We show that the state of these protocols grows linearly (in terms of number of keys) as they "ratchet" every time a new message is generated, hence falling into the case where $\Delta = 1$. On the other hand, we have two secure communication protocols given by Alwen, Coretti, and Dodis (called ACD and ACD-PK) [ACD19] which model well what Signal is deploying. We observe that the state in both schemes does not grow linearly like other PCS schemes. This is due to the fact that these two protocols do not guarantee Δ-PCS for any constant Δ. In fact, healing happens only when the direction of communication changes.

We conclude that adding PCS to FS-secure systems can be succeeded at the price of a minimal state growth with proven bounds and we cannot hope for better.

[4] It grows linearly if we take the key size as a memory unit. (The key size cannot have a constant bit length. Otherwise, exhaustive search breaks it with constant complexity.)

Structure of Chapter

In Section 5.1, we define a basic PCS-secure primitive called SEQ and we prove that its state size must grow super-linearly. In Section 5.2, we apply this result to self-encryption. We construct a scheme based on AGJ with super-linear growth and PCS security. Finally, in Section 5.3, we show how to apply our result to instant secure messaging.

5.1 Impossibility Result

In this section, we first define a minimal primitive called Self Encrypted Queue (SEQ) achieving post-compromise security. This primitive is not meant to have any concrete application. However, we will prove that (examples of) useful primitives imply SEQ, and that SEQ must have a linearly growing state.

5.1.1 Definition of a Minimal Primitive

We define below a *minimal* primitive which works in two phases: It iteratively generates a sequence of plaintext/ciphertext pairs $(\mathsf{pt}, \mathsf{ct})$ by updating its state. Then, it takes the sequence of ct in the same order as generated and recovers the exact sequence of pt. The primitive is minimal in the sense that all considered applications which claim PCS must achieve this functionality and even more (such as being able to receive the list of ct in different order, or to have encryption and decryption steps mixed up). We build self-encryption with the help of SEQ primitive in the following sections.

Definition 13 (SEQ)**.** *A **Self Encrypted Queue (**SEQ**)** is a primitive defined by*

$\mathsf{Gen}(1^\lambda) \to \mathsf{st}$ *which generates an initial state;*

$\mathsf{Enc}(\mathsf{st}) \to (\mathsf{st}', \mathsf{pt}, \mathsf{ct})$ *which updates the state and adds to the queue a new message which is* pt *in clear and* ct *in encrypted form;*

$\mathsf{Dec}(\mathsf{st}, \mathsf{ct}) \to (\mathsf{st}', \mathsf{pt}/\perp)$ *which updates the state and decrypts* ct *which leads the queue. This is deterministic.*

We say that SEQ *is **correct to level-**n if the correctness game in Fig. 5.1 never returns 1.*

The principle of this primitive is that a state is updated at every encryption/decryption so that the new state can decrypt the released ciphertext in the order they have been released. In the correctness game, the queue is filled up with $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$, then emptied.

Definition 14. *Let $n(\lambda)$ and $\Delta(\lambda)$ be polynomially bounded positive integer functions of a security parameter λ.* SEQ *security is defined by the $\mathsf{OW}_{m,\Delta,\lambda}$ game in Fig. 5.1. We*

Correctness at level-n:
1: $\mathsf{Gen}(1^\lambda) \to \mathsf{st}_0$
2: **for** $i = 1$ to n **do** ▷ fill up the queue
3: $\quad \mathsf{Enc}(\mathsf{st}_{i-1}) \to (\mathsf{st}_i, \mathsf{pt}_i, \mathsf{ct}_i)$
4: **end for**
5: **for** $i = 1$ to n **do** ▷ empty the queue
6: $\quad \mathsf{Dec}(\mathsf{st}_{n+i-1}, \mathsf{ct}_i) \to (\mathsf{st}_{n+i}, \mathsf{pt}'_i)$
7: $\quad$ **if** $\mathsf{pt}_i \neq \mathsf{pt}'_i$ **then return** 1
8: **end for**
9: **return** 0

Game $\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A})$:
1: $\mathsf{Gen}(1^\lambda) \to \mathsf{st}_0$
2: **for** $i = 1$ to m **do**
3: $\quad \mathsf{Enc}(\mathsf{st}_{i-1}) \to (\mathsf{st}_i, \mathsf{pt}_i, \mathsf{ct}_i)$
4: **end for**
5: $\mathcal{A}(1^\lambda, \mathsf{st}_{m-\Delta}, \mathsf{ct}_1, \ldots, \mathsf{ct}_m) \to z$
6: **return** $\mathbb{1}_{z=\mathsf{pt}_m}$

Figure 5.1 – Correctness and OW games for SEQ

$\mathsf{Gen}(1^\lambda)$:
1: $\mathsf{st} \leftarrow (\lambda, [])$ ▷ a list of length 0
2: **return** st

$\mathsf{Enc}(\mathsf{st})$:
3: parse $\mathsf{st} = (\lambda, L)$
4: pick pt of length λ at random
5: $L \leftarrow (L, \mathsf{pt})$ ▷ append pt in L
6: $\mathsf{st} \leftarrow (\lambda, L)$
7: $\mathsf{ct} \leftarrow \bot$
8: **return** $(\mathsf{st}, \mathsf{pt}, \mathsf{ct})$

$\mathsf{Dec}(\mathsf{st}, \mathsf{ct})$:
9: parse $\mathsf{st} = (\lambda, L)$
10: parse $L = (\mathsf{pt}, L')$
▷ pt is the first length-λ element of st
11: $\mathsf{st} \leftarrow (\lambda, L')$
12: **return** $(\mathsf{st}, \mathsf{pt})$

Figure 5.2 – A trivial SEQ

say that SEQ ***with level*** n ***is*** Δ***-secure*** *if for any PPT adversary* $\mathcal{A}$*,*

$$\lambda \mapsto \max_{1 \leq m \leq n} \Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1]$$

is a negligible function.

The value of Δ represents the time the scheme needs to heal security after an exposure. This means that Δ steps after exposing the state, the new state has become safe again and the encryptions to follow will protect confidentiality. In the game, $\mathsf{st}_{m-\Delta}$ is exposed and the goal of the adversary is to decrypt ct_m. Most secure schemes are 1-secure, because security heals after $\Delta = 1$ encryption.

It is easy to design a secure SEQ of level n with a state with $O(n)$ keys inside. For instance, for any n, the scheme in Fig. 5.2 is a 1-secure SEQ to level n with state of size $n\lambda$, where λ is the security parameter. This SEQ is trivially correct: st accumulates all pt in a queue during encryption and releases them during decryption. It is also perfectly secure: pt is independent from the corresponding ct and from the previous states. Hence, any $\mathsf{OW}_{m,\Delta,\lambda}$ adversary has an advantage of $2^{-\lambda}$.

Ideally, states should not inflate. For that, one can count on ct to transport a helper to

recover pt without having to store it in st. However, we prove next that a correct and OW-secure SEQ primitive with st in a space $\mathcal{ST}$ of size $2^{o(n \log n)}$ does not exist.

5.1.2 Impossibility Result

Lemma 2. *There exists a (small) constant c such that for every probability $\alpha > 0$ and integers n, ℓ, Δ, k, for every correct* SEQ *primitive of level n as in Def. 13 with* st *in a space $\mathcal{ST}$ of size $|\mathcal{ST}| \leq 2^{\ell}$, there exist $m \leq n$ and an $\mathsf{OW}_{m,\Delta,\lambda}$ adversary $\mathcal{A}$ of complexity $(n - m + \Delta)T_{\mathsf{Enc}} + mT_{\mathsf{Dec}} + c$, and advantage at least*

$$\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{\alpha}{n}\left(1 - \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\left\lfloor \frac{n}{\Delta} \right\rfloor} 2^{\ell}\right)$$

where T_{Enc} and T_{Dec} are the complexities of Enc *and* Dec.

Interestingly, for $k = 2$ and $\alpha = \frac{1}{\lfloor n/\Delta \rfloor}$, this lemma gives $\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{\Delta}{n^2}(1 - e2^{\ell - \lfloor \frac{n}{\Delta} \rfloor})$. Thus, it is clear that $\ell \leq \lfloor \frac{n}{\Delta} \rfloor - 2$ is insecure.

We can be more precise and obtain insecurity when $\frac{\ell\Delta}{n}$ is bounded by a logarithmic term (of the security parameter). Let $\varepsilon = 2^{-\frac{\ell+1}{\lfloor n/\Delta \rfloor}}$. Lemma 2 with $\alpha = \frac{\varepsilon^2}{2}$ and $k = \left\lceil \frac{2}{\varepsilon} \right\rceil$ gives the following result:

Theorem 13. *There exists a (small) constant c such that for every integers n, ℓ, and Δ, for every correct* SEQ *primitive of level n as in Def. 13 with* st *in a space $\mathcal{ST}$ of size $|\mathcal{ST}| \leq 2^{\ell}$, there exist $m \leq n$ and an $\mathsf{OW}_{m,\Delta,\lambda}$ adversary $\mathcal{A}$ of complexity $(n - m + \Delta)T_{\mathsf{Enc}} + mT_{\mathsf{Dec}} + c$, and advantage at least*

$$\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{1}{4n} 2^{-2\frac{\ell+1}{\lfloor n/\Delta \rfloor}}$$

where T_{Enc} and T_{Dec} are the complexities of Enc *and* Dec.

This means that the state needs a size ℓ such that $\frac{\ell\Delta}{n}$ is superlogarithmic to achieve Δ-security up to n encryptions.

We can now prove Lemma 2.

Proof of Lemma 2. Let us consider a correct primitive of level n with st in a space $\mathcal{ST}$ such that $|\mathcal{ST}| \leq 2^{\ell}$. We will show that it is insecure. To do so, we will first express that the state st after n encryptions is subject to some constraints. Namely, constraints are defined as st being able to decrypt the generated sequence of ct correctly. The constraints increase with n, and the set of possible st values which make decryption correct decreases. The set of constrained states does *not* decrease exponentially because

of the surprising existence of "super states" which are able to decrypt more than their constraints. Namely, super states can decrypt encryptions from the "future" which have not been generated yet. This is counter-intuitive. This set of super states is a hard core in the set of constrained states. We show that the set of constrained but non-super states *does* decrease exponentially. Hence, by taking n large enough, constrained states become all super states: the state after n encryptions must be a super state. We use the property of the super state to mount an attack.

We first define notations. We extend the Enc and Dec functions. First of all, with random coins ρ, we write $\mathsf{Enc}(\mathsf{st};\rho) = (\mathsf{st}', \mathsf{pt}, \mathsf{ct})$ and consider Enc as deterministic with explicit coins. For $X \in \{\mathsf{Enc}, \mathsf{Dec}\}$ and $y \in \{\mathsf{st}, \mathsf{pt}, \mathsf{ct}\}$, we denote by $X_{\mathsf{o\_}y}$ the generated **o**utput of type y by the X operation: for both Enc and Dec, the output components define subfunctions $\mathsf{Enc}_{\mathsf{o\_st}}$, $\mathsf{Enc}_{\mathsf{o\_pt}}$, $\mathsf{Enc}_{\mathsf{o\_ct}}$, $\mathsf{Dec}_{\mathsf{o\_st}}$, $\mathsf{Dec}_{\mathsf{o\_pt}}$ by

$$\begin{aligned}\mathsf{Enc}(\mathsf{st};\rho) &= (\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st};\rho), \mathsf{Enc}_{\mathsf{o\_pt}}(\mathsf{st};\rho), \mathsf{Enc}_{\mathsf{o\_ct}}(\mathsf{st};\rho))\\ \mathsf{Dec}(\mathsf{st},\mathsf{ct}) &= (\mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{st},\mathsf{ct}), \mathsf{Dec}_{\mathsf{o\_pt}}(\mathsf{st},\mathsf{ct}))\end{aligned}$$

We further extend those functions with a variable number of inputs ρ or ct. We define

$$\begin{aligned}\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st},\rho_1,\ldots,\rho_i) &= \mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st},\rho_1,\ldots,\rho_{i-1});\rho_i)\\ \mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{st},\mathsf{ct}_1,\ldots,\mathsf{ct}_i) &= \mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{st},\mathsf{ct}_1,\ldots,\mathsf{ct}_{i-1}),\mathsf{ct}_i)\end{aligned}$$

with the convention that $\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}) = \mathsf{st}$ and $\mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{st}) = \mathsf{st}$, i.e., the functions with zero coins do nothing but returning st unchanged. Next, $\mathsf{Enc}_{\mathsf{o\_pt}}(\mathsf{st},\rho_1,\ldots,\rho_i)$ is the list of generated pt, $\mathsf{Enc}_{\mathsf{o\_ct}}(\mathsf{st},\rho_1,\ldots,\rho_i)$ is the list of generated ct, and $\mathsf{Dec}_{\mathsf{o\_pt}}(\mathsf{st},\mathsf{ct}_1,\ldots,\mathsf{ct}_i)$ is the list of decrypted pt:

$$\begin{aligned}\mathsf{Enc}_{\mathsf{o\_pt}}(\mathsf{st},\rho_1,\ldots,\rho_i) &= \big(\mathsf{Enc}_{\mathsf{o\_pt}}(\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st},\rho_1,\ldots,\rho_{j-1});\rho_j)\big)_{j=1,\ldots,i}\\ \mathsf{Enc}_{\mathsf{o\_ct}}(\mathsf{st},\rho_1,\ldots,\rho_i) &= \big(\mathsf{Enc}_{\mathsf{o\_ct}}(\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st},\rho_1,\ldots,\rho_{j-1});\rho_j)\big)_{j=1,\ldots,i}\\ \mathsf{Dec}_{\mathsf{o\_pt}}(\mathsf{st},\mathsf{ct}_1,\ldots,\mathsf{ct}_i) &= \big(\mathsf{Dec}_{\mathsf{o\_pt}}(\mathsf{Dec}_{\mathsf{o\_st}}(\mathsf{st},\mathsf{ct}_1,\ldots,\mathsf{ct}_{j-1});\mathsf{ct}_j)\big)_{j=1,\ldots,i}\end{aligned}$$

Let st_n be the state which is obtained after n encryptions, before starting the decryption phase. In order to characterize the constraints on st_n coming from the first i encryptions, we introduce a set $C[r_i]$ corresponding to (and indexed with) each update operation $r_i = (\mathsf{st}_0,\rho_1,\ldots,\rho_i)$. Due to correctness, st_n must decrypt $\mathsf{Enc}_{\mathsf{o\_ct}}(r_i)$ to $\mathsf{Enc}_{\mathsf{o\_pt}}(r_i)$, due to correctness. Hence, we define

$$C[r_i] = \big\{\mathsf{st} \in \mathcal{ST}; \mathsf{Dec}_{\mathsf{o\_pt}}(\mathsf{st},\mathsf{Enc}_{\mathsf{o\_ct}}(r_i)) = \mathsf{Enc}_{\mathsf{o\_pt}}(r_i)\big\}$$

Clearly, for any i and any $\mathsf{st}_0,\rho_1,\ldots,\rho_n$, we have

$$\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0,\rho_1,\ldots,\rho_n) \in C[\mathsf{st}_0,\rho_1,\ldots,\rho_i]$$

We note that $C[r_0]$, where $r_0 = \mathsf{st}_0$ is the set of states subject to no restriction, hence $C[\mathsf{st}_0] = \mathcal{ST}$. Furthermore, we note that

$$C[r_n] \subseteq \cdots \subseteq C[r_2] \subseteq C[r_1] \subseteq C[r_0] = \mathcal{ST}$$

A state in $C[r_{i-\Delta}]$ decrypts well the first $i-\Delta$ ciphertexts. It may also be in $C[r_{i-\Delta}, \rho_{i-\Delta+1}, \ldots, \rho_i]$ if it decrypts the next Δ ciphertexts produced with coins $\rho_{i-\Delta+1}, \ldots, \rho_i$. It may also be in $C[r_{i-\Delta}, \rho'_{i-\Delta+1}, \ldots, \rho'_i]$ and decrypt Δ ciphertexts produced with other coins. With good probability, some state may actually have the "super-power" to decrypt ciphertexts produced with Δ more random coins. We call those stated the *super states*. Intuitively, this is unexpected to happen but we show below that super-states exist and an adversary can build some easily.

More concretely, let $\alpha > 0$ be the probability from the statement of the theorem. We define a set of super states for $r_{j-\Delta} = (\mathsf{st}_0, \rho_1, \ldots, \rho_{j-\Delta})$:

$$S[r_{j-\Delta}] = \left\{ \mathsf{st} \in \mathcal{ST}; \Pr_{\rho'_{j-\Delta+1}, \ldots, \rho'_j} [\mathsf{st} \in C[r_{j-\Delta}, \rho'_{j-\Delta+1}, \ldots, \rho'_j]] > \alpha \right\}$$

This set $S[r_{j-\Delta}]$ defines a set of states which are α-likely to decrypt a "fork" in the sequence of random coins. (See Fig. 5.3.)

We note that $S[r_{j-\Delta}] \subseteq C[r_{j-\Delta}]$ as for $\mathsf{st} \in S[r_{j-\Delta}]$, there must exist (due to a non-zero probability) $\rho'_{j-\Delta+1}, \ldots, \rho'_j$ such that

$$\mathsf{st} \in C[r_{j-\Delta}, \rho'_{j-\Delta+1}, \ldots, \rho'_j] \subseteq C[r_{j-\Delta}]$$

We define a union of super states as follows:

$$S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}] = S[\mathsf{st}_0] \cup S[\mathsf{st}_0; \rho_1] \cup \cdots \cup S[\mathsf{st}_0; \rho_1, \ldots, \rho_{n-\Delta}]$$

Clearly

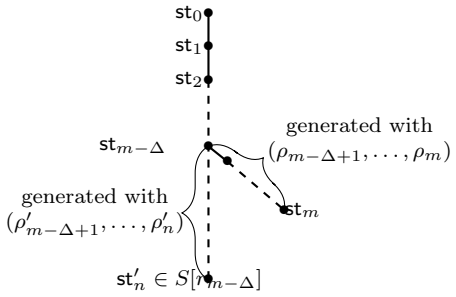
$$S^{\cup}[r_{n-\Delta}] \supseteq \cdots \supseteq S^{\cup}[r_1] \supseteq S^{\cup}[r_0]$$

The idea of the proof is to show that states with too many constraints tend to become super-states. Namely, we first prove that for n large enough, $C[r_n]$ is included in $S^{\cup}[r_{n-\Delta}]$ with large probability p. This means that after n encryptions, a state becomes a super-state. Hence, this state belongs to some $S[r_{m-\Delta}]$, with a random $m \leq n$. We now take a fixed value of m which is taken with probability at least $\frac{1}{n}$. (It exists, due to the pigeon-hole principle.) We take n encryptions from random coins $\mathsf{st}_0, \rho_1, \ldots, \rho_{m-\Delta}, \rho'_{m-\Delta+1}, \ldots, \rho'_n$. We deduce that there is a probability at least $\frac{p}{n}$ to get a state st'_n in $S[r_{m-\Delta}]$. If it happens, st'_n decrypts what is generated by the fork $\mathsf{st}_0, \rho_1, \ldots, \rho_m$ with probability at

least α (by definition of the super states). We define an adversary that exploits this fact in Fig. 5.3. The m encryptions with $\mathsf{st}_0, \rho_1, \ldots, \rho_m$ are generated by the game, the state $\mathsf{st}_{m-\Delta}$ leaks, and the adversary can fork to construct st'_n from it. We obtain the success probability of the adversary in the $\mathsf{OW}_{m,\Delta,\lambda}$ game:

$$\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{\alpha p}{n} \tag{5.1}$$

In what follows, we show that $p \geq 1 - \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\lfloor \frac{n}{\Delta} \rfloor} 2^{\ell}$.



$\underline{\mathcal{A}(1^\lambda, \mathsf{st}_{m-\Delta}, \mathsf{ct}_1, \ldots, \mathsf{ct}_m)}$:
1: $\mathsf{st}'_{m-\Delta} \leftarrow \mathsf{st}_{m-\Delta}$
2: **for** $i = m - \Delta + 1$ to n **do**
3: pick ρ'_i
4: $\mathsf{Enc}(\mathsf{st}'_{i-1}; \rho'_i) \to (\mathsf{st}'_i, \mathsf{pt}'_i, \mathsf{ct}'_i)$
5: **end for**
6: **for** $i = 0$ to $m - 1$ **do**
7: $\mathsf{Dec}(\mathsf{st}'_{n+i}, \mathsf{ct}_{i+1}) \to (\mathsf{st}'_{n+i+1}, \overline{\mathsf{pt}}_{i+1})$
8: **end for**
9: **return** $\overline{\mathsf{pt}}_m$

Figure 5.3 – Starting from state st_0 and applying m encryption that generates $\mathsf{ct}_1, \ldots, \mathsf{ct}_m$, we hope that leaking st_m and forking to n encryptions in total will end up in $\mathsf{st}'_n \in S[r_{m-\Delta}]$. Therefore, st'_n decrypts all the ciphertext with probability at least α.

Let i be an integer. We consider for the moment that $\mathsf{st}_0, \rho_1, \ldots, \rho_{i-\Delta}$ are fixed. For simplicity, we denote

$$\begin{aligned} C_{i-\Delta} &= C[\mathsf{st}_0, \rho_1, \ldots, \rho_{i-\Delta}] \\ C_i(\vec{\rho}) &= C[\mathsf{st}_0, \rho_1, \ldots, \rho_{i-\Delta}, \vec{\rho}] \\ S^{\cup}_{i-\Delta} &= S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{i-2\Delta}] \\ S^{\cup}_i &= S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{i-\Delta}] \end{aligned}$$

for a vector $\vec{\rho}$ of dimension Δ. We take k independent random Δ-dimensional vectors $\vec{\rho}_j$, for integers $j = 1, \ldots, k$ and we define $C_{i,j} = C_i(\vec{\rho}_j)$. ($k$ is defined in the statement of the Lemma.) Given $\vec{\rho}_j$ fixed and some $\mathsf{st} \in C_{i,j} - S^{\cup}_i$ fixed, we have $\mathsf{st} \notin S^{\cup}_i$ meaning that $\mathsf{st} \notin S[\mathsf{st}_0, \rho_1, \ldots, \rho_{i-\Delta}]$, thus

$$\Pr_{\vec{\rho}_{j'}}[\mathsf{st} \in C_{i,j'}] \leq \alpha$$

for any $\vec{\rho}_{j'}$ independent vector indexed with $j' \neq j$, by definition of S_i and $C_{i,j'}$. We count

$$\left|(C_{i,j'} - S^{\cup}_i) \cap (C_{i,j} - S^{\cup}_i)\right| = \sum_{\mathsf{st} \in C_{i,j} - S^{\cup}_i} \mathbb{1}_{\mathsf{st} \in C_{i,j'}}$$

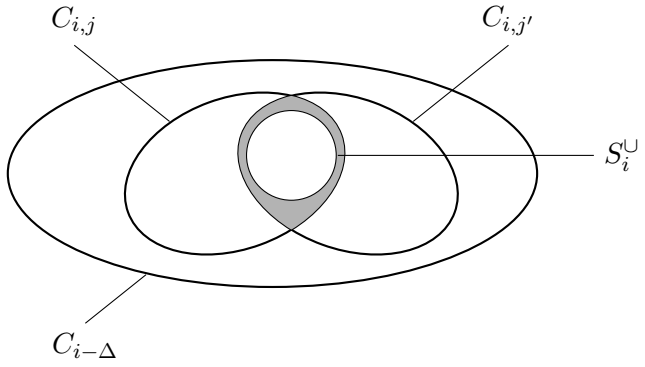


Figure 5.4 – Illustration of the intersection $(C_{i,j'} - S_i^{\cup}) \cap (C_{i,j} - S_i^{\cup})$

We obtain

$$\mathop{\mathsf{E}}_{\vec{\rho}_{j'}} \left[\left| (C_{i,j'} - S_i^{\cup}) \cap (C_{i,j} - S_i^{\cup}) \right| \right] \leq \alpha |C_{i,j} - S_i^{\cup}| \leq \alpha |C_{i-\Delta} - S_{i-\Delta}^{\cup}|$$

for any j, j', and $\vec{\rho}_j$ with $j \neq j'$. This is illustrated in Fig. 5.4. Clearly, we can then randomize $\vec{\rho}_j$ and obtain

$$\mathsf{E} \left[\left| (C_{i,j'} - S_i^{\cup}) \cap (C_{i,j} - S_i^{\cup}) \right| \right] \leq \alpha |C_{i-\Delta} - S_{i-\Delta}^{\cup}|$$

for any j and j' with $j \neq j'$.

Let $A_j = C_{i,j} - S_i^{\cup}$. This denotes one of the k subsets of $A = C_{i-\Delta} - S_{i-\Delta}^{\cup}$. We have

$$\sum_{j=1}^{k} |A_j| \leq |A| + \sum_{1 \leq j < j' \leq k} |A_j \cap A_{j'}|$$

Indeed, any element x of A occurring in exactly m subsets A_j is counted m times on the left-hand side and $1 + \frac{m(m-1)}{2}$ times on the right-hand side. However, $m \leq 1 + \frac{m(m-1)}{2}$ for every integer m. We deduce

$$\mathsf{E} \left[\sum_{j=1}^{k} |C_{i,j} - S_i^{\cup}| \right] \leq \left(1 + \frac{k(k-1)}{2} \alpha \right) |C_{i-\Delta} - S_{i-\Delta}^{\cup}|$$

Given that all $\mathsf{E} \left[|C_{i,j} - S_i^{\cup}| \right]$ are equal, we have proven that

$$\mathop{\mathsf{E}}_{\vec{\rho}} \left[|C_i(\vec{\rho}) - S_i^{\cup}| \right] \leq \left(\frac{1}{k} + \frac{k-1}{2} \alpha \right) |C_{i-\Delta} - S_{i-\Delta}^{\cup}|$$

We can now randomize $\rho_1, \ldots, \rho_{n-\Delta}$ as well and obtain

$$\mathsf{E}\left[|C[\mathsf{st}_0, \rho_1, \ldots, \rho_n] - S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}]|\right] \leq \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\lfloor \frac{n}{\Delta} \rfloor} |\mathcal{ST}|$$

We bound $|\mathcal{ST}| \leq 2^{\ell}$ and $\Pr[E \neq \emptyset] \leq \mathsf{E}[|E|]$ (due to the Markov inequality) for a random set E and obtain

$$\Pr[C[\mathsf{st}_0, \rho_1, \ldots, \rho_n] - S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}] \neq \emptyset] \leq \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\lfloor \frac{n}{\Delta} \rfloor} 2^{\ell}$$

By assumption on the size of $\mathcal{ST}$, for n large enough, we obtain that the set difference $C[\mathsf{st}_0, \rho_1, \ldots, \rho_n] - S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}]$ is likely to be empty which means that the states in $C[\mathsf{st}_0, \rho_1, \ldots, \rho_n]$ are super states. By the definition of $C[r_n]$, $\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0; \rho_1, \ldots, \rho_n) \in C[\mathsf{st}_0; \rho_1, \ldots, \rho_n]$. Hence, $\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0; \rho_1, \ldots, \rho_n)$ is likely to be in $S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}]$. More precisely,

$$\Pr[\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0, \rho_1, \ldots, \rho_n) \notin S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}]] \leq \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\lfloor \frac{n}{\Delta} \rfloor} 2^{\ell}$$

If $\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0, \rho_1, \ldots, \rho_n) \in S^{\cup}[\mathsf{st}_0, \rho_1, \ldots, \rho_{n-\Delta}]$, it means there exists (at least) one $m \leq n$ such that

$$\Pr_{\vec{\rho'}}[\mathsf{Enc}_{\mathsf{o\_st}}(\mathsf{st}_0, \rho_1, \ldots, \rho_n) \in C(\mathsf{st}_0, \rho_1, \ldots, \rho_{m-\Delta}, \vec{\rho'})] > \alpha$$

Therefore, we obtain the success probability in the $\mathsf{OW}_{m,\Delta,\lambda}$ game (from Eq. (5.1)):

$$\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{\alpha}{n}\left(1 - \left(\frac{1}{k} + \frac{k-1}{2}\alpha\right)^{\lfloor \frac{n}{\Delta} \rfloor} 2^{\ell}\right)$$

The complexity of $\mathcal{A}$ is $n - m + \Delta$ encryptions and m decryptions. □

5.2 Self-Ratchet

5.2.1 Definitions

Consider a self-ratcheted scheme $\mathsf{SR} = (\mathsf{lg}, \mathsf{Init}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Punc})$ with the following syntax:

- $\mathsf{lg}(\lambda)$ (length of the plaintext)
- $\mathsf{SR.Init}(1^{\lambda}) \xrightarrow{\$} \mathsf{st}$ (output an initial state for the device)
- $\mathsf{SR.Enc}(\mathsf{st}, \mathsf{pt}) \xrightarrow{\$} (\mathsf{st}', \mathsf{ct})$ (update the state while producing a pt/ct pair, with $\mathsf{pt} \in \{0,1\}^{\mathsf{lg}(\lambda)}$)

- $\mathsf{SR.Dec}(\mathsf{st}, \mathsf{ct}) \to \mathsf{pt}$ or $\perp$ (decrypt ct into pt)
- $\mathsf{SR.Punc}(\mathsf{st}, \mathsf{ct}) \to \mathsf{st}'$ (update the state by puncturing ct in st)

In our settings, there exists a device following a protocol which produces some pt/ct for itself so that it can eventually decrypt ct to recover pt in the future. Encryption is stateful. The protocol makes sure that when the device should no longer be able to decrypt ct and should be secure against any future state exposure, it can "puncture" the state. This means that the state st which can decrypt ct is replaced by a new (punctured) state st' so that ct is not decryptable by st' any more. With this notion, we aim at forward secrecy. We aim at post-compromise security as well: a state exposure should not compromise future encryptions.

Definition 15 (SR)**.** *A* ***self-ratcheted scheme*** *(*SR*) is a primitive* $\mathsf{SR} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Punc})$ *which is* n***-correct*** *in the sense that for any sequence* sched*, the game in Fig. 5.5 never returns 1. Here,* sched *is a sequence of scheduled instructions which can be of three different types:* $(\text{“}\mathsf{Enc}\text{”}, \mathsf{pt})$ *(encrypt plaintext* pt*),* $(\text{“}\mathsf{Dec}\text{”}, j)$ *(decrypt the* j*-th produced ciphertext), and* $(\text{“}\mathsf{Punc}\text{”}, j)$ *(puncture the* j*-th produced ciphertext).*

The correctness notion must consider any order of $\mathsf{Enc}/\mathsf{Dec}/\mathsf{Punc}$ instructions. This is what sched is modeling. We describe what should happen when this sequence of instructions is sched. Actually, we declare in L_{ct} the ciphertexts which are "active" and we put in L_{pt} how they are expected to decrypt.

Compared to SEQ, an SR does not update the state in decryption (this is rather done by a separate function) and decryption can be done in any order of the ciphertexts (i.e., not only in the oder they have been created).

This definition assumes that the number of "active" ciphertexts remains bounded by a parameter n (line number 5).

Application to cloud storage. SR schemes can be used for cloud storage where a client wants to store her files on the cloud in an encrypted form. Ideally, a single file is encrypted with $\mathsf{SR.Enc}$ to obtain a ct. For retrieval, the $\mathsf{SR.Dec}$ is run to decrypt the file. Eventually, when the client wants to remove the file from the cloud, the protocol will puncture her state for ct. The first desired security is that after a client erases an encrypted file, even though a copy was illegally kept and the state of the client later leaks, the file is unrecoverable. This is forward secrecy. It is achieved by puncturing. The second desired security is that after the state of a client has leaked, if the client wants to store a new file in the cloud, this file should be safe, as long as no exposure occurs during the activity time of this file. This is post-compromise security. It is achieved by what we call self-ratchet.

```
1: SR.Init(1^λ) →$ st
2: set lists L_pt and L_ct to empty
3: for i = 1 to |sched| do
4:     if sched_i parses as ("Enc", pt) for some pt then
5:         if the number of L_ct entries which are different from ⊥ is at most n − 1 then
6:             SR.Enc(st, pt) → (st, ct)
7:             L_pt ← (L_pt, pt)
8:             L_ct ← (L_ct, ct)
9:         end if
10:    else if sched_i parses as ("Dec", j) for some j then
11:        if L_ct[j] exists and L_ct[j] ≠ ⊥ then
12:            SR.Dec(st, L_ct[j]) → pt
13:            if pt ≠ L_pt[j] then return 1
14:        end if
15:    else if sched_i parses as ("Punc", j) for some j then
16:        if L_ct[j] exists and L_ct[j] ≠ ⊥ then
17:            SR.Punc(st, L_ct[j]) → st
18:            L_ct[j] ← ⊥
19:        end if
20:    end if
21: end for
22: return 0
```

Figure 5.5 – Correctness game for SR of level n

One problem specific to cloud storage is that files are typically big and SR should handle them in encryption, decryption, and puncturing. One common approach is to use a domain expander based on a hybrid construction. Like the KEM/DEM hybrid cryptosystems, we can use SR to encrypt an ephemeral key K and symmetrically encrypt the plaintext with K.

We could also add key rotation, if required, by using SR to encrypt the encryption key: to encrypt a file pt, we pick a random key k (in the key domain of the key rotation scheme) and we run $\tilde{C}_1 \leftarrow \mathsf{SR.Enc}(\mathsf{st}, k)$. Then, we encrypt pt with k following the key rotation scheme and obtain a header $\tilde{C}_2$ and C. The ciphertext is $\mathsf{ct} = (\tilde{C}_1, \tilde{C}_2, C)$. To rotate the key k, we puncture st with $\tilde{C}_1$, produce a new value for $\tilde{C}_1$ with a new k and run the key rotation scheme on $(\tilde{C}_2, C)$.

Application to 0-RTT session resumption. SR schemes can be used for 0-RTT session resumption. Essentially, a server having a secure connection with a client using a key K would use $\mathsf{SR.Enc}(\mathsf{st}, \mathsf{K})$ to issue a ticket ct and send ct to the client. To resume a session, the client, who kept K and ct, would resend ct to the server who would use SR.Dec to recover K. The server might also immediately puncture it to avoid any replay of the ticket ct and for forward secrecy.

Previous work on 0-RTT session resumption. Def. 15 is more general than the definition of 0-RTT session resumption [AGJ19]. The differences are as follows:

- SR separates SR.Dec and SR.Punc instead of having both functionalities in the same algorithm;
- the notations for 0-RTT session resumption are Setup, TicketGen, and ServerRes instead of Init, Enc, Dec.

There is no formal definition of correctness for 0-RTT session resumption in Aviram et al. [AGJ19]. However, we can fairly assume it is the same as our notion of correctness in Def. 15, but when sequences sched are limited such that every decryption is followed by puncturing: for all i and j, if $\mathsf{sched}_i = (\text{“Dec”}, j)$ then $\mathsf{sched}_{i+1} = (\text{“Punc”}, j)$. In 0-RTT session resumption, it makes sense to merge SR.Dec with SR.Punc as one of the security goal is precisely to prevent a ct to be replayed. For cloud storage, the client may need to decrypt the same ct several times before she removes the file from the cloud. Hence, we keep SR.Dec and SR.Punc separate.

We adapt the security definition of 0-RTT session resumption with our notations to which we add specific instructions for post-compromise security. We define the $\mathsf{IND}^{\mathsf{SR,opt}}_{b,n,\Delta,\lambda}(\mathcal{A})$ game in Fig. 5.6. We also generalize it to adaptive security. In the AGJ security model, the game starts with many OEnc and only after that, the adversary can play with oracles except OEnc (it is somehow non-adaptive). The AGJ model uses $\mathsf{opt} = \{\mathsf{FS}, \mathsf{replay}\}$ and it is formalized for key establishment rather than encryption. (This means that there is a Test oracle to test a decryption instead of a Challenge oracle to get an encryption challenge.)

Definition 16 (SR security)**.** *Let $n(\lambda)$ and $\Delta(\lambda)$ be polynomially bounded positive integer functions of a security parameter λ. The option set* opt *specifies some variants in the game in Fig. 5.6. The advantage is*

$$\mathsf{Adv}^{\mathsf{IND}^{\mathsf{SR,opt}}}_{n,\Delta,\lambda}(\mathcal{A}) = \left|\Pr\left[\mathsf{IND}^{\mathsf{SR,opt}}_{1,n,\Delta,\lambda}(\mathcal{A}) \to 1\right] - \Pr\left[\mathsf{IND}^{\mathsf{SR,opt}}_{0,n,\Delta,\lambda}(\mathcal{A}) \to 1\right]\right|$$

We say that SR ***is*** IND-opt ***secure at level*** n ***with delay*** Δ *if for any PPT adversary $\mathcal{A}$, $\lambda \mapsto \mathsf{Adv}^{\mathsf{IND}^{\mathsf{SR,opt}}}_{n,\Delta,\lambda}(\mathcal{A})$ is a negligible function.*

When “replay” $\in$ opt, the security notion aims to address replay attacks. Hence, decryption must puncture, as well. When “FS” $\in$ opt, the security notion aims to capture forward secrecy *without* post-compromise security. Absence of FS in opt is a stronger security notion as it captures FS and PCS together.

Game $\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}_{b,n,\Delta,\lambda}(\mathcal{A})$:

1: $\mathsf{Init}(1^\lambda) \to \mathsf{st}$
2: $\mathsf{Active}, \mathsf{Revealed} \leftarrow \emptyset$
3: $\mathsf{challenged} \leftarrow \mathsf{false}$
4: $\mathsf{AfterExp} \leftarrow \Delta$
5: $\mathcal{A}^{\mathsf{OEnc},\mathsf{ODec},\mathsf{Challenge},\mathsf{OPunc},\mathsf{OExp}}(1^\lambda) \to b^*$
6: **return** b^*

Oracle $\mathsf{OEnc}(\mathsf{pt})$:

7: **if** $|\mathsf{Active}| \geq n$ **then return** $\perp$
8: $\mathsf{SR.Enc}(\mathsf{st}, \mathsf{pt}) \to (\mathsf{st}, \mathsf{ct})$
9: $\mathsf{Active} \leftarrow \mathsf{Active} \cup \{\mathsf{ct}\}$
10: $\mathsf{Revealed} \leftarrow \mathsf{Revealed} \cup \{\mathsf{ct}\}$
11: $\mathsf{AfterExp} \leftarrow \mathsf{AfterExp} + 1$
12: **return** ct

Oracle $\mathsf{ODec}(\mathsf{ct})$:

13: **if** $\mathsf{ct} \in \mathsf{Active} - \mathsf{Revealed}$ **then**
14: **return** $\perp$
15: **end if**
16: $\mathsf{SR.Dec}(\mathsf{st}, \mathsf{ct}) \to r$
17: **if** "replay" $\in \mathsf{opt}$ **then** $\mathsf{OPunc}(\mathsf{ct})$
18: **return** r

Oracle $\mathsf{OPunc}(\mathsf{ct})$:

19: $\mathsf{SR.Punc}(\mathsf{st}, \mathsf{ct}) \to \mathsf{st}$
20: $\mathsf{Active} \leftarrow \mathsf{Active} - \{\mathsf{ct}\}$
21: $\mathsf{Revealed} \leftarrow \mathsf{Revealed} - \{\mathsf{ct}\}$
22: **return**

Oracle $\mathsf{Challenge}(\mathsf{pt}_1)$:

23: **if** challenged **then return** $\perp$
24: **if** $|\mathsf{Active}| \geq n$ **then return** $\perp$
25: **if** $\mathsf{AfterExp} < \Delta$ **then return** $\perp$
26: pick pt_0 of same length as pt_1 at random
27: $\mathsf{SR.Enc}(\mathsf{st}, \mathsf{pt}_b) \to (\mathsf{st}, \mathsf{ct})$
28: $\mathsf{Active} \leftarrow \mathsf{Active} \cup \{\mathsf{ct}\}$
29: $\mathsf{AfterExp} \leftarrow \mathsf{AfterExp} + 1$
30: $\mathsf{challenged} \leftarrow \mathsf{true}$
31: **return** ct

Oracle $\mathsf{OExp}()$:

32: **if** ($\neg$challenged and "FS" $\in$ opt) or ($\mathsf{Active} - \mathsf{Revealed} \neq \emptyset$) **then**
33: **return** $\perp$
34: **end if**
35: $\mathsf{AfterExp} \leftarrow 0$
36: **return** st

Figure 5.6 – Indistinguishability game for self-ratchet

5.2.2 Impossibility Result

Theorem 14. *For every integer n, ℓ, $\Delta > 0$ and any n-correct self-ratcheted scheme* SR *following Def. 15, and such that* st *belongs to a space of size bounded by 2^ℓ, there exist a (small) constant c and an adversary of complexity $(n - m + \Delta)T_{\mathsf{Enc}} + m(T_{\mathsf{Dec}} + T_{\mathsf{Punc}}) + (n + \Delta)T_{\$} + m + c$ having advantage*

$$\mathsf{Adv}^{\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}}_{n,\Delta,\lambda}(\mathcal{A}) > \frac{1}{4n} 2^{-2\frac{\ell+1}{\lfloor n/\Delta \rfloor}} - 2^{-\mathsf{lg}(\lambda)}$$

for $\mathsf{opt} = \perp$ *and* $\mathsf{opt} = \mathsf{replay}$*, and where $T_{\$}$ is the complexity to pick an element of $\{0,1\}^{\mathsf{lg}(\lambda)}$ at random and T_{Enc}, T_{Dec} and T_{Punc} are the complexities of* Enc, Dec *and* Punc.

Proof. We construct a SEQ from a self-ratcheted protocol SR in Fig. 5.7. Clearly, the n-correctness of SR implies the n-correctness of S for any n. The SEQ scheme only imposes ciphertexts to be received in the same order as they have been produced.

Due to Th. 13, there exists m and an $\mathsf{OW}_{m,\Delta,\lambda}$ adversary $\mathcal{B}$ such that $\Pr[\mathsf{OW}_{m,\Delta,\lambda} \to 1] = p$ with $p > \frac{1}{4n} 2^{-2\frac{\ell+1}{\lfloor n/\Delta \rfloor}}$. Then, we can construct an $\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}_{b,n,\Delta,\lambda}$ adversary $\mathcal{A}$ as in

$\underline{S.\mathsf{Gen}} = \mathsf{SR.Init}$

$\underline{S.\mathsf{Enc}(\mathsf{st})}$:
1: pick $\mathsf{K} \in \{0,1\}^{\mathsf{lg}(\lambda)}$ at random
2: **return** $\mathsf{SR.Enc}(\mathsf{st}, \mathsf{K})$

$\underline{S.\mathsf{Dec}(\mathsf{st}, \mathsf{ct})}$:
3: $\mathsf{SR.Dec}(\mathsf{st}, \mathsf{ct}) \to \mathsf{K}$
4: **if** $\mathsf{K} \neq \perp$ **then** $\mathsf{SR.Punc}(\mathsf{st}, \mathsf{ct}) \to \mathsf{st}$
5: **return** $(\mathsf{st}, \mathsf{K})$

Figure 5.7 – SEQ from SR

$\underline{\mathcal{A}^{\mathsf{OEnc},\mathsf{ODec},\mathsf{Challenge},\mathsf{OPunc},\mathsf{OExp}}(1^\lambda)}$:
1: **for** $i = 1$ to $m - \Delta$ **do**
2: pick pt_i at random
3: $\mathsf{OEnc}(\mathsf{pt}_i) \to ct_i$
4: **end for**
5: $\mathsf{OExp}() \to st_{m-\Delta}$
6: **for** $i = m - \Delta + 1$ to $m - 1$ **do**
7: pick pt_i at random
8: $\mathsf{OEnc}(\mathsf{pt}_i) \to ct_i$
9: **end for**
10: pick pt_m at random
11: $\mathsf{Challenge}(\mathsf{pt}_m) \to \mathsf{ct}_m$
12: $\mathcal{B}(\mathsf{st}_{m-\Delta}, \mathsf{ct}_1, \ldots, \mathsf{ct}_m) \to z$
13: **return** $\mathbb{1}_{z=\mathsf{pt}_m}$

Figure 5.8 – Adversary against SR based on an adversary for SEQ

Fig. 5.8.

The Challenge oracle encrypts pt_m which is either pt_m or random. As $\mathcal{A}$ simulates well the $\mathsf{OW}_{m,\Delta,\lambda}$ game, we have $\Pr[z = \mathsf{pt}_m] = p$. Hence, $\Pr[\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}_{1,n,\Delta} \to 1] = p$ and $\Pr[\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}_{0,n,\Delta} \to 1] = 2^{-\mathsf{lg}(\lambda)}$. Hence, the advantage is $p - 2^{-\mathsf{lg}(\lambda)}$.

The adversary $\mathcal{A}$ picks m plaintexts and issues $m - 1$ OEnc queries, one OExp query and one Challenge query, and then simulates an $\mathsf{OW}_{m,\Delta,\lambda}$ adversary $\mathcal{B}$. The complexity of $\mathcal{B}$ is the complexity of $n - m + \Delta$ encryptions and m decryptions, and the complexities of $S.\mathsf{Enc}$ and $S.\mathsf{Dec}$ are respectively $T_{\mathsf{Enc}} + T_{\$}$ and $T_{\mathsf{Dec}} + T_{\mathsf{Punc}}$. The complexity of $\mathcal{A}$ therefore is $n - m + \Delta$ encryptions, m decryptions, m punctuations, $m + 1$ oracle calls and $(n + \Delta)$ random selections. □

5.2.3 Constructions

We provide a generic construction SR from an FS-secure self-ratcheted scheme FSSR providing forward secrecy. For every Δ, we create a new structure with forward secrecy and store it. Given a scheme FSSR offering only forward secrecy, we construct SR as in Fig. 5.9.

Theorem 15. *Let $n(\lambda)$ and $\Delta(\lambda)$ be polynomially bounded positive integer functions of a security parameter λ. Let* opt *be either $\perp$ or* {replay}*. Let* FSSR *be a self-ratcheted scheme which is* IND-(opt $\cup$ {FS}) *secure at level Δ. Then,* SR *(in Fig. 5.9, with parameter Δ) is a self-ratcheted scheme which is* IND-opt *secure at level n with delay Δ.*

$\underline{\mathsf{SR.Init}(1^\lambda)}$:
1: $\mathsf{st} \leftarrow (0, [])$
▷ a counter set to 0 and an empty list
2: **return** st

$\underline{\mathsf{SR.Dec}(\mathsf{st}, \mathsf{ct})}$:
3: parse $\mathsf{st} = (c, L)$ and $\mathsf{ct} = (i, \mathsf{ct}_0)$
4: $\mathsf{FSSR.Dec}(L[i], \mathsf{ct}_0) \to \mathsf{pt}$
5: **return** pt

$\underline{\mathsf{SR.Punc}(\mathsf{st}, \mathsf{ct})}$:
6: parse $\mathsf{st} = (c, L)$ and $\mathsf{ct} = (i, \mathsf{ct}_0)$
7: $\mathsf{FSSR.Punc}(L[i], \mathsf{ct}_0) \to L[i]$
▷ $L[i]$ is updated
8: $\mathsf{st} \leftarrow (c, L)$
9: **return** st

$\underline{\mathsf{SR.Enc}(\mathsf{st}, \mathsf{pt})}$:
10: parse $\mathsf{st} = (c, L)$
11: **if** $c = 0$ **then**
12: $c \leftarrow \Delta$
13: $\mathsf{FSSR.Init}(1^\lambda) \to s$
14: $L \leftarrow (L, s)$
▷ add a new FSSR state in L
15: **end if**
16: $c \leftarrow c - 1$
17: set ℓ to the length of L
18: $\mathsf{FSSR.Enc}(L[\ell], \mathsf{pt}) \to (L[\ell], \mathsf{ct}_0)$
▷ $L[\ell]$ is updated
19: $\mathsf{st} \leftarrow (c, L)$
20: $\mathsf{ct} \leftarrow (\ell, \mathsf{ct}_0)$
21: **return** $(\mathsf{st}, \mathsf{ct})$

Figure 5.9 – Post-compromise secure self-ratchet from forward secure self-ratchet

Proof. Let opt be either $\perp$ or $\{\mathsf{replay}\}$ and $\mathcal{B}$ be an IND-opt adversary against SR with delay Δ. Assume that $\mathcal{B}$ queries at most q encryption and challenge queries. Then, we can construct an IND-$(\mathsf{opt} \cup \{\mathsf{FS}\})$ adversary $\mathcal{A}$ against FSSR at level Δ as shown on Fig. 5.10.

By the construction, SR generates a new state of FSSR for each Δ encryptions. The adversary $\mathcal{A}$ therefore simulates the IND-opt security game with delay Δ while trying to replace Δ ciphertexts by the ciphertexts that the adversary is challenging. If the oracle $\mathsf{Challenge}'$ does not abort the game, the adversary $\mathcal{A}$ can correctly guess b if $\mathcal{B}$ can correctly guess it. The probability that the game is not aborted by $\mathsf{Challenge}'$ is about Δ/q. Then, the advantage of $\mathcal{A}$ is

$$\mathsf{Adv}^{\mathsf{IND}^{\mathsf{FSSR},(\mathsf{opt}\cup\{\mathsf{FS}\})}}_{\Delta,\cdot,\lambda}(\mathcal{A}) = \frac{1}{\lceil q/\Delta \rceil} \mathsf{Adv}^{\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}}_{n,\Delta,\lambda}(\mathcal{B})$$

As q is polynomially bounded and $\Delta \geq 1$, $\mathsf{Adv}^{\mathsf{IND}^{\mathsf{SR},\mathsf{opt}}}_{n,\Delta,\lambda}(\mathcal{B})$ is negligible if $\mathsf{Adv}^{\mathsf{IND}^{\mathsf{FSSR},(\mathsf{opt}\cup\{\mathsf{FS}\})}}_{\Delta,\cdot,\lambda}(\mathcal{A})$ is negligible. Hence, SR is IND-opt secure at level n with delay Δ if FSSR is IND-$(\mathsf{opt} \cup \{\mathsf{FS}\})$ secure at level Δ. □

Optimization. Our SR scheme can obviously be optimized for storage. For each state $L[i]$, we can add a counter of active ciphertexts with $L[i]$ which is incremented by Enc and decremented by Punc (after checking that decryption works). Then, when the counter becomes 0, $L[i]$ can be erased.

Another convenient optimization holds when the application wants to operate bulk puncturing of too old ciphertexts. This implies to erase all first $L[i]$. It is quite compatible

$\mathcal{A}^{\mathsf{OEnc},\mathsf{ODec},\mathsf{Challenge},\mathsf{OPunc},\mathsf{OExp}}(1^\lambda)$:

1: $\mathsf{idx} \xleftarrow{\$} \{1,\ldots,\lceil q/\Delta \rceil\}$
2: $\mathsf{SR.Init}(1^\lambda) \to \mathsf{st}$
3: $\mathsf{Active}, \mathsf{Revealed} \leftarrow \emptyset$
4: $\mathsf{challenged} \leftarrow \mathsf{false}$
5: $\mathsf{AfterExp} \leftarrow \Delta$
6: $\mathcal{B}^{\mathsf{OEnc}',\mathsf{ODec}',\mathsf{Challenge}',\mathsf{OPunc}',\mathsf{OExp}'}(1^\lambda) \to b'$
7: **return** b'

Subroutine $\mathsf{OEnc}'(\mathsf{pt})$:

8: **if** $|\mathsf{Active}| \geq n$ **then return** $\perp$
9: $\mathsf{SR.Enc}(\mathsf{st},\mathsf{pt}) \to (\mathsf{st},\mathsf{ct})$
10: parse $\mathsf{ct} = (\ell, \mathsf{ct}_0)$
11: **if** $\ell = \mathsf{idx}$ **then**
12: $\quad \mathsf{OEnc}(\mathsf{pt}) \to \mathsf{ct}_0$
13: **end if**
14: $\mathsf{ct} \leftarrow (\ell, \mathsf{ct}_0)$
15: $\mathsf{Active} \leftarrow \mathsf{Active} \cup \{\mathsf{ct}\}$
16: $\mathsf{Revealed} \leftarrow \mathsf{Revealed} \cup \{\mathsf{ct}\}$
17: $\mathsf{AfterExp} \leftarrow \mathsf{AfterExp} + 1$
18: **return** ct

Subroutine $\mathsf{ODec}'(\mathsf{ct})$:

19: **if** $\mathsf{ct} \in \mathsf{Active} - \mathsf{Revealed}$ **then**
20: $\quad$ **return** $\perp$
21: **end if**
22: parse $\mathsf{ct} = (\ell, \mathsf{ct}_0)$
23: **if** $\ell = \mathsf{idx}$ **then**
24: $\quad \mathsf{ODec}(\mathsf{ct}_0) \to \mathsf{pt}$
25: **else**
26: $\quad \mathsf{SR.Dec}(\mathsf{st},\mathsf{ct}) \to (\mathsf{st},\mathsf{pt})$
27: **end if**
28: **if** "replay" $\in \mathsf{opt}$ **then** $\mathsf{OPunc}'(\mathsf{ct})$
29: **return** pt

Subroutine $\mathsf{OPunc}'(\mathsf{ct})$:

30: parse $\mathsf{ct} = (\ell, \mathsf{ct}_0)$
31: **if** $\ell = \mathsf{idx}$ **then**
32: $\quad \mathsf{OPunc}(\mathsf{ct}_0)$
33: **else**
34: $\quad \mathsf{SR.Punc}(\mathsf{st},\mathsf{ct}) \to \mathsf{st}$
35: **end if**
36: $\mathsf{Active} \leftarrow \mathsf{Active} - \{\mathsf{ct}\}$
37: $\mathsf{Revealed} \leftarrow \mathsf{Revealed} - \{\mathsf{ct}\}$
38: **return**

Subroutine $\mathsf{Challenge}'(\mathsf{pt})$:

39: **if** $|\mathsf{Active}| \geq n$ or $\mathsf{AfterExp} < \Delta$ **then**
40: $\quad$ **return** $\perp$
41: **end if**
42: parse $\mathsf{st} = (c, L)$
43: **if** $(c \neq 0$ or $|L| \neq \mathsf{idx} - 1)$ and $(c = 0$ or $|L| \neq \mathsf{idx})$ **then**
44: $\quad$ abort the game
45: **end if**
46: $\mathsf{SR.Enc}(\mathsf{st},\mathsf{pt}) \to (\mathsf{st},\mathsf{ct})$
47: $\mathsf{Challenge}(\mathsf{pt}) \to \mathsf{ct}$
48: $\mathsf{Active} \leftarrow \mathsf{Active} \cup \{\mathsf{ct}\}$
49: $\mathsf{AfterExp} \leftarrow \mathsf{AfterExp} + 1$
50: $\mathsf{challenged} \leftarrow \mathsf{true}$
51: **return** ct

Subroutine $\mathsf{OExp}'()$:

52: parse $\mathsf{st} = (c, L)$
53: **if** $|L| \geq \mathsf{idx}$ **then**
54: $\quad \mathsf{OExp}() \to \mathsf{st}'$
55: $\quad$ **if** $\mathsf{st}' = \perp$ **then return** $\perp$
56: $\quad L[\mathsf{idx}] \leftarrow \mathsf{st}'$
57: **end if**
58: $\mathsf{AfterExp} \leftarrow 0$
59: **return** (c, L)

Figure 5.10 – FS adversary for FSSR based on an adversary for SR

with recent policies of session resumption: a session which is too old cannot be resumed.

5.2.4 FS-Secure Self-Ratcheted Scheme (Adapted from AGJ)

We adapt the generic construction from Aviram et al. [AGJ19] based on a puncturable PRF denoted as PPRF. We define PPRF as a set of following algorithms:

- $\mathsf{Setup}(1^\lambda) \to k_{\mathsf{PPRF}}$ which generates an initial PPRF key;
- $\mathsf{Eval}(k_{\mathsf{PPRF}}, x) \to y/\perp$ which computes a pseudo-random output from the PPRF key

and x;

- $\mathsf{Punc}(k_{\mathsf{PPRF}}, x) \to k'_{\mathsf{PPRF}}$ which updates the PPRF key.

We use authenticated encryption with associated data $\mathsf{AEAD} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. (In our notation, the second input to Enc and Dec is the associated data i.e. the header to be authenticated.) The construction is in Fig. 5.11. The complete security proof is presented in the following subsection along with the security of PPRF and AEAD. We only need AEAD to be a secure encryption scheme. We kept AEAD as it was in AGJ [AGJ19] and it would still be useful to authenticate ct.

FSSR.Init(1^λ):
1: PPRF.Setup(1^λ) $\to k_{\mathsf{PPRF}}$
2: st $\leftarrow (k_{\mathsf{PPRF}}, 0)$
3: **return** st

FSSR.Enc(st, pt):
4: parse st $= (k_{\mathsf{PPRF}}, \mathsf{cnt})$
5: $\kappa \leftarrow$ PPRF.Eval$(k_{\mathsf{PPRF}}, \mathsf{cnt})$
6: **if** $\kappa = \perp$ **then return** $\perp$
7: $\mathsf{ct}_0 \leftarrow$ AEAD.Enc$(\kappa, \mathsf{cnt}, \mathsf{pt})$
8: ct $\leftarrow (\mathsf{cnt}, \mathsf{ct}_0)$
9: st $\leftarrow (k_{\mathsf{PPRF}}, \mathsf{cnt} + 1)$
10: **return** st, ct

FSSR.Dec(st, ct):
11: parse st $= (k_{\mathsf{PPRF}}, \mathsf{cnt})$
12: parse ct $= (\mathsf{cnt}', \mathsf{ct}_0)$
13: $\kappa \leftarrow$ PPRF.Eval$(k_{\mathsf{PPRF}}, \mathsf{cnt}')$
14: **if** $\kappa = \perp$ **then return** $\perp$
15: pt $\leftarrow$ AEAD.Dec$(\kappa, \mathsf{cnt}', \mathsf{ct})$
16: **return** pt

FSSR.Punc(st, ct):
17: parse st $= (k_{\mathsf{PPRF}}, \mathsf{cnt})$
18: parse ct $= (\mathsf{cnt}', \mathsf{ct}_0)$
19: PPRF.Punc$(k_{\mathsf{PPRF}}, \mathsf{cnt}') \to k_{\mathsf{PPRF}}$
20: **if** $k_{\mathsf{PPRF}} = \perp$ **then return** $\perp$
21: st $\leftarrow (k_{\mathsf{PPRF}}, \mathsf{cnt})$
22: **return** st

Figure 5.11 – FS-secure SR

AGJ presented two possible PPRF constructions. One is based on the Camenisch-Lysyanskaya RSA accumulator [CL02]. The other is based on a Merkle tree [Mer87].

RSA-based PPRF. The RSA-based construction uses a PPRF key of linear size in terms of the number of encryptions and can only handle a polynomial number of encryptions. This is the total number of encryptions, i.e. not only the ones remaining active. We give the construction in Fig. 5.12, using a random oracle H and the list of first odd primes $(p_1, \ldots, p_m)$. In the original paper [AGJ19], the authors have shown that the above construction is a secure PPRF in the random oracle model, under the strong RSA assumption. The PPRF key is of size $2\lambda + m$. However, the N part of the key can be set as a domain parameter which is common to many keys.

In our construction, the device only needs to encrypt Δ messages per PPRF key. Hence, we can set $m = \Delta$ in the above PPRF, meaning that the FS-secure self-ratcheted scheme has states of size $\lambda + \Delta + \log_2 \Delta$ plus λ bits of common parameter N. Finally, our secure

$\mathsf{PPRF.Setup}(1^\lambda)$:
1: generate an RSA modulus $N = pq$ of length λ using safe primes
2: erase p and q
3: pick $g \in \mathbf{Z}_N$ at random
4: $r \leftarrow (0, 0, \ldots, 0) \in \{0,1\}^m$
5: $k_{\mathsf{PPRF}} \leftarrow (N, g, r)$
6: **return** k_{PPRF}

$\mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, x)$:
7: parse $k_{\mathsf{PPRF}} = (N, g, r)$
8: **if** $r_x = 1$ **then return** $\perp$
9: $P_x \leftarrow \prod_{i=1}^{m} p_i^{r_i \cdot \mathbb{1}_{i \neq x}}$
10: $y \leftarrow H(g^{P_x} \bmod N)$
11: **return** y

$\mathsf{PPRF.Punc}(k_{\mathsf{PPRF}}, x)$:
12: parse $k_{\mathsf{PPRF}} = (N, g, r)$
13: **if** $r_x = 1$ **then return** $\perp$
14: $g \leftarrow g^{p_x} \bmod N$
15: $r_x \leftarrow 1$
16: $k_{\mathsf{PPRF}} \leftarrow (N, g, r)$
17: **return** k_{PPRF}

Figure 5.12 – RSA-based PPRF

self-ratcheted scheme has states of size

$$\ell = \frac{n}{\Delta}\left(\lambda + \Delta + \log_2 \Delta\right) + \log_2 \Delta + \lambda \tag{5.2}$$

We can see that $\frac{\ell\Delta}{n}$ is at least linear in λ, hence super-logarithmic.

Tree-based PPRF. The tree-based constructions is formed with two functions G_0 and G_1 from $\{0,1\}^\lambda$ to itself, which we extend to functions G_x for every binary word x by $G_{xy}(L) = G_y(G_x(L))$. Then, the PPRF defines a binary tree of depth d which is partially labeled. The PPRF key is a set of (x, L) pairs where x is a binary word (hence a node in the binary tree) and L is its label in $\{0,1\}^\lambda$. Initially, the key consists of the label of the root ε. To evaluate on x, one should find a labeled node (y, L) such that y is a prefix of x, write $x = yz$, and return $G_z(L)$. The interface of the PPRF only takes d-bit input x (i.e. leaves), but our evaluation is defined for every node. To puncture a leaf x, one should find this y again and replace (y, L) from the key by the list of (x', L') with $x' = yz_1 \cdots z_{i-1}\bar{z}_i$ and L' being the evaluation on x', where $z_1 \cdots z_{|z|}$ is the binary expansion of z and $\bar{z}_i$ is the bit complement of z_i. Hence, a PPRF key is an anti-chain with no siblings. In the worst case, it could inflate by d pairs at every puncture, but the maximum length is of 2^{d-1} pairs.

Same as the RSA construction, one only needs to evaluate $2^d = \Delta$ leaves. In the worst case, a PPRF key has length $2^{d-1} \times d\lambda$ which is $\frac{1}{2}\lambda\Delta\log_2\Delta$. Hence, the FS-secure self-ratcheted scheme has states of size bounded by $\frac{1}{2}\lambda\Delta\log_2\Delta + \log_2\Delta$. Finally, our secure self-ratcheted scheme has states of size

$$\ell = \frac{n}{\Delta}\left(\frac{1}{2}\lambda\Delta\log_2\Delta + \log_2\Delta\right) + \log_2\Delta$$

which is larger than with the RSA-based method.

5.2.5 FS-Security of FSSR

PPRF. We assume for simplicity that Eval and Punc are deterministic in PPRF and the PPRF correctness imposes that for every k_{PPRF}, x, y such that $x \neq y$, we have

$$\mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, y) = \mathsf{PPRF.Eval}(\mathsf{PPRF.Punc}(k_{\mathsf{PPRF}}, x), y)$$

(This is the case for the RSA-based scheme in Fig. 5.12.) We define the PPRF-security by the indistinguishability of b in the game in Fig. 5.13. The set Q keeps all active challenge queries. They are added by the Challenge oracle. They can be removed by the OPunc oracle. The OEval oracle refuses to evaluate an active challenge query, as it would make a trivial attack. Similarly, Challenge refuses to re-evaluate an active challenge query. The OExp oracle similarly refuses to answer if some challenge queries are still active. The OExp oracle can only be used as a last oracle query, which is enforced by the exposed flag. The adversary is limited to q queries to either Challenge or OEval.

$\underline{\mathsf{IND}^{\mathsf{PPRF}}_{b,q,\lambda}}$:
1: $k_{\mathsf{PPRF}} \leftarrow \mathsf{PPRF.Setup}(1^\lambda)$
2: $Q \leftarrow \emptyset$
3: exposed $\leftarrow$ false
4: $z \leftarrow \mathcal{A}^{\mathsf{Challenge},\mathsf{OEval},\mathsf{OPunc},\mathsf{OExp}}(1^\lambda)$
5: **return** z

$\underline{\mathsf{Challenge}(x)}$:
6: **if** $x \in Q$ or exposed **then return** $\perp$
7: **if** q PPRF.Eval have been done **then return** $\perp$
8: $Q \leftarrow Q \cup \{x\}$
9: $y_0 \xleftarrow{\$} \mathcal{Y}$
10: $y_1 \leftarrow \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, x)$
11: **return** y_b

$\underline{\mathsf{OEval}(x)}$:
12: **if** $x \in Q$ or exposed **then return** $\perp$
13: **if** q PPRF.Eval have been done **then return** $\perp$
14: $y \leftarrow \mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, x)$
15: **return** y

$\underline{\mathsf{OPunc}(x)}$:
16: $k_{\mathsf{PPRF}} \leftarrow \mathsf{PPRF.Punc}(k_{\mathsf{PPRF}}, x)$
17: $Q \leftarrow Q - \{x\}$
18: **return**

$\underline{\mathsf{OExp}()}$:
19: **if** $Q \neq \emptyset$ **then return** $\perp$
20: exposed $\leftarrow$ true
21: **return** k_{PPRF}

Figure 5.13 – PPRF security game

AEAD. We define security of *authenticated encryption with associated data (*AEAD*)* with the game in Fig. 5.14. Actually, this is the security of a symmetric encryption scheme, as we need no authentication in our model. Besides the challenge encryption, there is no chosen plaintext. The adversary can make chosen ciphertext decryption queries. The adversary is limited to q queries in total.

Theorem 16. *Consider the construction* FSSR *in Fig. 5.11. For every* $\mathsf{IND}^{\mathsf{FSSR},\mathsf{FS}}$*-adversary* $\mathcal{A}$ *which is limited to* q_e *calls to either* OEnc *or* Challenge *and* q_d *calls to* ODec*, there exist two* $\mathsf{IND}^{\mathsf{PPRF}}$*-adversaries* $\mathcal{B}_b$*,* $b = 0, 1$*, and* q_e $\mathsf{IND}^{\mathsf{AEAD}}$*-adversaries* $\mathcal{C}_{\mathsf{cnt}}$*,*

$\underline{\mathsf{IND}^{\mathsf{AEAD}}_{b,q,\lambda}}$:
1: $\kappa \leftarrow \mathsf{AEAD.Gen}(1^\lambda)$
2: $\mathsf{Revealed} \leftarrow \emptyset$
3: $z \leftarrow \mathcal{A}^{\mathsf{ODec},\mathsf{Challenge}}(1^\lambda)$
4: **return** z

$\underline{\mathsf{ODec}(\mathsf{ad},\mathsf{ct})}$:
5: **if** q queries have been made **then return** $\perp$
6: **if** $(\mathsf{ad},\mathsf{ct}) \in \mathsf{Revealed}$ **then return** $\perp$
7: $\mathsf{pt} \leftarrow \mathsf{AEAD.Dec}(\kappa,\mathsf{ad},\mathsf{ct})$
8: **return** pt

$\underline{\mathsf{Challenge}(\mathsf{ad},\mathsf{pt}_1)}$:
9: **if** q queries have been made **then return** $\perp$
10: pick pt_0 at random of same length as pt_1
11: $\mathsf{ct} \leftarrow \mathsf{AEAD.Enc}(\kappa,\mathsf{ad},\mathsf{pt}_b)$
12: $\mathsf{Revealed} \leftarrow \mathsf{Revealed} \cup \{(\mathsf{ad},\mathsf{ct})\}$
13: **return** ct

Figure 5.14 – AEAD security game

$\mathsf{cnt} = 1,\ldots,q_e$, *such that*

$$\mathsf{Adv}^{\mathsf{IND}^{\mathsf{FSSR},\mathsf{FS}}}_{n,\Delta,\lambda}(\mathcal{A}) \le \sum_{b=0}^{1} \mathsf{Adv}^{\mathsf{IND}^{\mathsf{PPRF}}}_{q_e+q_d,\lambda}(\mathcal{B}_b) + \sum_{\mathsf{cnt}=1}^{q_e} \mathsf{Adv}^{\mathsf{IND}^{\mathsf{AEAD}}}_{q_d,\lambda}(\mathcal{C}_{\mathsf{cnt}})$$

Proof. Let us consider the $\mathsf{IND}^{\mathsf{FSSR},\mathsf{FS}}_{b,n,\Delta,\lambda}$ game played with an adversary $\mathcal{A}$.

First, we transform $\mathcal{A}$ into an adversary $\mathcal{A}'$ who simulates $\mathcal{A}$ until it makes a successful OExp query. Then, $\mathcal{A}'$ makes this OExp query as well to get the state st. After this query, $\mathcal{A}'$ continues to simulate $\mathcal{A}$ but makes no oracle query any more. Instead, $\mathcal{A}'$ simulates all oracles responding to $\mathcal{A}$. Given st, $\mathcal{A}'$ can impersonate the device. Only the knowledge of b is missing, but this is only used by the Challenge oracle if challenged is false. OExp can only succeed if challenged is true, due to the FS option in $\mathsf{IND}^{\mathsf{FSSR},\mathsf{FS}}$. Hence, there cannot be any Challenge query needing b any more. Consequently, $\mathcal{A}'$ can easily simulate all oracles. The $\mathsf{IND}^{\mathsf{FSSR},\mathsf{FS}}_{b,n,\Delta,\lambda}$ game with $\mathcal{A}'$ has exactly the same advantage:

$$\mathsf{Adv}^{\mathsf{IND}^{\mathsf{FSSR},\mathsf{FS}}}_{n,\Delta,\lambda}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{IND}^{\mathsf{FSSR},\mathsf{FS}}}_{n,\Delta,\lambda}(\mathcal{A}')$$

Then, we construct for each b an adversary $\mathcal{B}_b$ who simulates $\mathcal{A}'$, all oracles in the game in Fig. 5.6 and the algorithms of FSSR in Fig. 5.11, but every PPRF operation. This adversary $\mathcal{B}_b$ plays in an $\mathsf{IND}^{\mathsf{PPRF}}_{1,q_e+q_d,\lambda}$ game.

- The $\mathsf{PPRF.Setup}(1^\lambda)$ execution in $\mathsf{FSSR.Init}(1^\lambda)$ at the beginning of the game is done by the $\mathsf{IND}^{\mathsf{PPRF}}_{1,q_e+q_d,\lambda}$ game.
- The $\mathsf{PPRF.Eval}(k_{\mathsf{PPRF}},x)$ execution from $\mathsf{FSSR.Enc}(\mathsf{st},\mathsf{pt})$ which is in the $\mathsf{Challenge}(\mathsf{pt})$ query in the $\mathsf{IND}^{\mathsf{FSSR},\mathsf{FS}}_{b,n,\Delta,\lambda}$ game is simulated by a $\mathsf{Challenge}(x)$ oracle call to the $\mathsf{IND}^{\mathsf{PPRF}}_{1,q_e+q_d,\lambda}$ game. Thanks to the transformation into $\mathcal{A}'$, the exposed abort condition of Challenge in Fig. 5.13 is never met. As FSSR.Enc increments a counter

$\mathsf{cnt} = x$, Challenge queries are never made on a repeated input x. Hence, the $x \in Q$ abort condition of Challenge in Fig. 5.13 is never met either. In the $\mathsf{IND}^{\mathsf{PPRF}}_{1,q_e+q_d,\lambda}$ game, Challenge and OEval are equivalent. The simulation is perfect.

- The other $\mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, x)$ executions are simulated by $\mathsf{OEval}(x)$ oracle calls, unless $x \in Q$. If $x \in Q$, it means that it was a Challenge query before and that x was not punctured. Hence, the result of $\mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, x)$ should be known and we do not need to query $\mathsf{OEval}(x)$ for that, thanks to the correctness property. Thanks to the transformation into $\mathcal{A}'$, the exposed abort condition of OEval in Fig. 5.13 is never met. The simulation is perfect.
- The $\mathsf{PPRF.Punc}(k_{\mathsf{PPRF}}, x)$ executions from $\mathsf{FSSR.Punc}(\mathsf{st}, \mathsf{ct})$ in the $\mathsf{OPunc}(\mathsf{ct})$ query in Fig. 5.6 are simulated by $\mathsf{OPunc}(x)$ oracle calls in Fig. 5.13. The simulation is perfect.
- The reveal of k_{PPRF} in the OExp query in the end of the game in Fig. 5.6 is done using an OExp oracle call in Fig. 5.13. Clearly, all elements of Q should have been cleared as we must have $\mathsf{Active} - \mathsf{Revealed} = \emptyset$ in the $\mathsf{IND}^{\mathsf{FSSR,FS}}_{b,n,\Delta,\lambda}$ game.

This creates an adversary $\mathcal{B}_b$ for the $\mathsf{IND}^{\mathsf{PPRF}}_{b',q_e+q_d,\lambda}$ game. This perfectly simulates $\mathsf{IND}^{\mathsf{FSSR,FS}}_{b}$ when $b' = 1$:

$$\mathsf{Adv}^{\mathsf{IND}^{\mathsf{FSSR,FS}}}_{n,\Delta,\lambda}(\mathcal{A}') = \left|\Pr\left[\mathsf{IND}^{\mathsf{PPRF}}_{1,q_e+q_d,\lambda}(\mathcal{B}_1) \to 1\right] - \Pr\left[\mathsf{IND}^{\mathsf{PPRF}}_{1,q_e+q_d,\lambda}(\mathcal{B}_0) \to 1\right]\right|$$

Due to the PPRF security, the outcome is indistinguishable from the one obtained with $b' = 0$. The advantage gap is of

$$\mathsf{Adv}^{\mathsf{IND}^{\mathsf{PPRF}}}_{q_e+q_d,\lambda}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{IND}^{\mathsf{PPRF}}}_{q_e+q_d,\lambda}(\mathcal{B}_0)$$

Now, we can consider the $\mathsf{IND}^{\mathsf{PPRF}}_{0,q_e+q_d,\lambda}$ game with $\mathcal{B}_b$ and revert the previous transformation. We obtain that the adversary $\mathcal{A}'$ is unchanged but the game becomes some Γ_b game like $\mathsf{IND}^{\mathsf{FSSR,FS}}$, but with modified oracles as follows:

- for each $\mathsf{Challenge}(\mathsf{pt})$ query, the game looks at the state $\mathsf{st} = (k_{\mathsf{PPRF}}, \mathsf{cnt})$, picks a random κ, uses it instead of the result of $\mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, \mathsf{cnt})$, and keeps in memory that cnt is mapped to κ as long as cnt remains active (i.e. in the set Q);
- for each $\mathsf{ODec}(\mathsf{ct})$ query on some $\mathsf{ct} = (\mathsf{cnt}, \mathsf{ct}_0)$ with a cnt which is remembered in memory, when cnt is still active, the game also bypasses PPRF.Eval to use the same κ directly.

We have

$$\mathsf{Adv}^{\mathsf{IND}^{\mathsf{FSSR,FS}}}_{n,\Delta,\lambda}(\mathcal{A}') - \mathsf{Adv}^{\Gamma}_{\lambda}(\mathcal{A}') \leq \mathsf{Adv}^{\mathsf{IND}^{\mathsf{PPRF}}}_{q_e+q_d,\lambda}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{IND}^{\mathsf{PPRF}}}_{q_e+q_d,\lambda}(\mathcal{B}_0)$$

Finally, for each value of the counter cnt, we define an $\mathsf{IND}^{\mathsf{AEAD}}$ game with an adversary $\mathcal{C}_{\mathsf{cnt}}$. Again, $\mathcal{C}_{\mathsf{cnt}}$ simulates $\mathcal{A}'$, oracles in Γ_b and FSSR algorithms in Fig. 5.11, but some AEAD executions with a target key κ corresponding to cnt.

- For any $\mathsf{Challenge}(\mathsf{pt}_1)$ query by $\mathcal{A}'$ (which induces $\mathcal{B}_b$ making a $\mathsf{Challenge}(x)$ query in Fig. 5.13) such that the state has form $\mathsf{st} = (k_{\mathsf{PPRF}}, \mathsf{cnt})$ for some k_{PPRF}, we have $\mathsf{cnt} = x$ and a random κ is selected to replace $\mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, \mathsf{cnt})$ before computing $\mathsf{AEAD.Enc}(\kappa, \mathsf{pt}_b)$. Instead, $\mathcal{C}_{\mathsf{cnt}}$ does not select κ and rather queries $\mathsf{Challenge}(\mathsf{pt}_1)$ in the $\mathsf{IND}^{\mathsf{AEAD}}$ game. The result ct^* is kept in memory. Note that no OEnc query by $\mathcal{A}'$ needs to run $\mathsf{PPRF.Eval}(k_{\mathsf{PPRF}}, \mathsf{cnt})$ as the counter cnt is incremented. However, some ODec queries by $\mathcal{A}'$ may need it.
- For other $\mathsf{Challenge}(x)$ queries, $\mathcal{C}_{\mathsf{cnt}}$ aborts.
- For any time $\mathcal{A}'$ should remember the hidden value of κ to make a $\mathsf{AEAD.Dec}(\kappa, \mathsf{ad}, \mathsf{ct})$ computation, $\mathcal{C}_{\mathsf{cnt}}$ checks if $(\mathsf{ad}, \mathsf{ct})$ is equal to the value of $(\mathsf{cnt}, \mathsf{ct}^*)$ above (which should not happen as it would mean that $\mathcal{B}_b$ tries to decrypt the challenge ciphertext, hence $\mathsf{ct} \in \mathsf{Active} - \mathsf{Revealed}$ in ODec in Fig. 5.6) and return $\perp$ in that case, otherwise call $\mathsf{ODec}(\mathsf{ad}, \mathsf{ct})$.

We have

$$\Pr[\Gamma_b(\mathcal{A}') \to 1] = \sum_{\mathsf{cnt}} \Pr[\mathsf{IND}_b^{\mathsf{AEAD}}(\mathcal{C}_{\mathsf{cnt}} \to 1]$$

Hence,

$$\mathsf{Adv}_{\lambda}^{\Gamma}(\mathcal{A}') \leq \sum_{\mathsf{cnt}} \mathsf{Adv}_{q_d,\lambda}^{\mathsf{IND}^{\mathsf{AEAD}}}(\mathcal{C}_{\mathsf{cnt}})$$

□

5.2.6 Experimental Results

We instantiate an SR based on FSSR with the RSA-based PPRF. We assumed that the same RSA modulus is used for all PPRF keys, the RSA modulus so is precomputed and given as a parameter to SR. Hence, the cost of setting up the RSA modulus is not covered in our analysis. For H and AEAD, we used SHA-256 and AES-GCM.

Our experiment was done on a machine with the AMD Opteron 8354 processor and 128 GB of RAM by using the SageMath version 8.7. We picked a common 2048-bit RSA modulus.

We tried many values for Δ from $\Delta = 100$ to $\Delta = 10\,000$ by steps of 100. We measured the worst case complexity of an $\mathsf{SR.Enc}$ encryption, which is actually the very first one when nothing is punctured and which includes $\mathsf{FSSR.Init}$, as well as the best case complexity of $\mathsf{SR.Enc}$, which is the very last one after all other values have been punctured. For

accuracy, we did it 1000 times for each Δ and took the average. The results are plotted in Fig. 5.15.

On the plot, we added the total state size divided by the total number n of encryptions as it goes to infinity. This is essentially $\frac{\ell}{n}$ with ℓ given by Eq.(5.2). As we can see, the execution time grows linearly with Δ while $\frac{\ell}{n} - 1$ is inverse proportional to Δ.

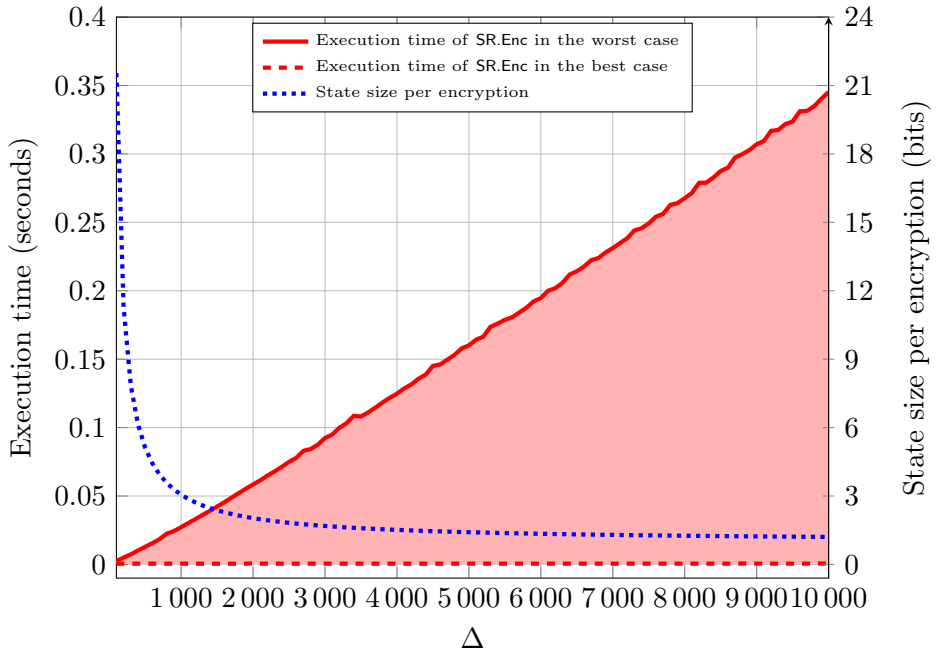


Figure 5.15 – The execution time of SR.Enc in the worst/best case and the state size divided by the number of encryptions with 2048-bit RSA modulus.

5.3 Bipartite Ratcheted Communication

5.3.1 Definitions

We consider a ratcheted scheme $S = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ following the syntax

- $S.\mathsf{Gen}(1^\lambda) \to (\mathsf{st}_A, \mathsf{st}_B)$ (generate a pair of states)
- $S.\mathsf{Enc}(\mathsf{st}) \to (\mathsf{st}', \mathsf{pt}, \mathsf{ct})$ (update the state while producing a pt/ct pair)
- $S.\mathsf{Dec}(\mathsf{st}, \mathsf{ct}) \to (\mathsf{st}', \mathsf{pt})$ (update the state while decrypting ct)

To avoid defining a general correctness and security for ratcheted schemes, which is quite lengthy and complicated, we only adopt a definition matching a particular case

1: $S.\mathsf{Gen}(1^\lambda) \to (\mathsf{st}_0^A, \mathsf{st}_0^B)$
2: **for** $i = 1$ to n **do**
3: $\quad S.\mathsf{Enc}(\mathsf{st}_{i-1}^A) \to (\mathsf{st}_i^A, \mathsf{pt}_i', \mathsf{ct}_i')$
4: $\quad S.\mathsf{Dec}(\mathsf{st}_{i-1}^B, \mathsf{ct}_i') \to (x, \mathsf{pt}_i')$
5: $\quad S.\mathsf{Enc}(x) \to (\mathsf{st}_i^B, \mathsf{pt}_i, \mathsf{ct}_i)$
6: **end for**
7: $x \leftarrow \mathsf{st}_n^A$
8: **for** $i = 1$ to n **do**
9: $\quad S.\mathsf{Dec}(x, \mathsf{ct}_i) \to (x, \mathsf{pt})$
10: $\quad$ **if** $\mathsf{pt} \neq \mathsf{pt}_i$ **then return** 1
11: **end for**
12: **return** 0

Figure 5.16 – Correctness game for a simple ratcheted scheme of level-n

of our interest. This is the case when one participant Alice desperately tries to reach her counterpart Bob by consistently sending messages without receiving any response, while Bob actually acknowledges for the receipt of every message from Alice but his acknowledgments somehow never make it through. (See Fig. 5.17.)

Definition 17. *A **simple ratcheted scheme** is a primitive S defined by $S = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ which is n-**correct** in the sense that the game in Fig. 5.16 never returns 1.*

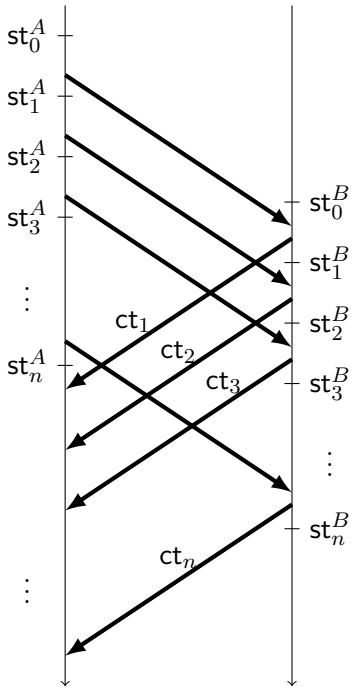


Figure 5.17 – Simulation of the level-n correctness game

In this communication pattern, protocols such as PR [PR18], JS [JS18], JMM [JMM19], and DV [DV19] have growing states. We can clearly see it on the implementation results by Caforio et al. [CDV19]. Protocols such as Signal [Sys17] or ACD [ACD19] keep constant-size states but offer no post-compromise security in our communication pattern. In fact, in ACD, Alice keeps sending messages in the same "epoch" (following the terminology

of ACD [ACD19]) using the forward secret scheme called FS in ACD, while Bob receives those messages from an old epoch (for him) and keeps sending messages in his own epoch, using FS as well. As the FS scheme is deterministic, it offers no post-compromise security. In ACD-PK, there is an extra public-key encryption but the decryption key remains constant within the same epoch. Hence, exposing st_1^A is enough to decrypt all ciphertexts in both ACD and ACD-PK.

Post-compromise security should make impossible to decrypt ct_m which was released after having ratcheted Δ times both participants after the last state exposure which revealed $\mathsf{st}_{m-\Delta}^A$ and $\mathsf{st}_{m-\Delta}^B$. For instance, with $\Delta = 1$ and $m = 2$, it should be impossible on Fig. 5.17 to compute pt_2 from $(\mathsf{st}_1^A, \mathsf{st}_1^B, \mathsf{ct}_1, \mathsf{ct}_2)$. This is formalized by the following definition.

Definition 18. *Let $n(\lambda)$ and $\Delta(\lambda)$ be polynomially bounded positive integer functions of a security parameter λ. For a simple ratcheted scheme S which is n-correct, we define the game in Fig. 5.18 with parameters $m \leq n$ and $\Delta > 0$: We say that S* ***with level n is Δ-secure*** *if for any PPT adversary $\mathcal{A}$, $\lambda \mapsto \max_{1 \leq m \leq n} \Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1]$ is a negligible function.*

$\underline{\mathsf{OW}_{m,\Delta,\lambda}}$:
1: $S.\mathsf{Gen}(1^\lambda) \to (\mathsf{st}_0^A, \mathsf{st}_0^B)$
2: **for** $i = 1$ to m **do**
3: $\quad S.\mathsf{Enc}(\mathsf{st}_{i-1}^A) \to (\mathsf{st}_i^A, \mathsf{pt}_i', \mathsf{ct}_i')$
4: $\quad S.\mathsf{Dec}(\mathsf{st}_{i-1}^B, \mathsf{ct}_i') \to (x, \mathsf{pt}_i')$
5: $\quad S.\mathsf{Enc}(x) \to (\mathsf{st}_i^B, \mathsf{pt}_i, \mathsf{ct}_i)$
6: **end for**
7: $\mathcal{A}(1^\lambda, \mathsf{st}_{m-\Delta}^A, \mathsf{st}_{m-\Delta}^B, \mathsf{ct}_1, \ldots \mathsf{ct}_m) \to x$
8: **return** $1_{x=\mathsf{pt}_m}$

Figure 5.18 – OW game for a simple ratcheted scheme

5.3.2 Impossibility Result

Theorem 17. *For every integer n, ℓ, $\Delta > 0$ and any n-correct simple ratcheted scheme S following Def. 17, and such that $(\mathsf{st}_A, \mathsf{st}_B)$ belongs to a space of size bounded by 2^ℓ, there exist $m \leq n$ and an adversary of low complexity having advantage*

$$\Pr[\mathsf{OW}_{m,\Delta,\lambda}(\mathcal{A}) \to 1] > \frac{1}{4n} 2^{-2\frac{\ell+1}{\lceil n/\Delta \rceil}}$$

in the security game of Def. 18.

Proof. We construct a SEQ protocol P as shown in Fig. 5.19. If S is n-correct (in the sense of Def. 17), then this new scheme P is correct to level n (in the sense of Def. 13).

This comes from a direct translation of definitions. Furthermore, any adversary against P (in the sense of Def. 14) translates into an adversary against S in the sense of Def. 18: given $(\mathsf{st}^A_{m-\Delta}, \mathsf{st}^B_{m-\Delta})$ the adversary decrypts ct_m. Advantages are the same. We conclude by applying Th. 13. □

$\underline{P.\mathsf{Gen}(1^\lambda) \to \mathsf{st}:}$
1: $S.\mathsf{Gen}(1^\lambda) \to (\mathsf{st}_A, \mathsf{st}_B)$
2: $\mathsf{st} \leftarrow (\mathsf{st}_A, \mathsf{st}_B)$
3: **return** st

$\underline{P.\mathsf{Dec}(\mathsf{st}, \mathsf{ct}) \to (\mathsf{st}', \mathsf{pt}):}$
4: parse $\mathsf{st} = (\mathsf{st}_A, \mathsf{st}_B)$
5: $S.\mathsf{Dec}(\mathsf{st}_A, \mathsf{ct}) \to (\mathsf{st}'_A, \mathsf{pt})$
6: $\mathsf{st}' \leftarrow (\mathsf{st}'_A, \mathsf{st}_B)$
7: **return** $(\mathsf{st}', \mathsf{pt})$

$\underline{P.\mathsf{Enc}(\mathsf{st}) \to (\mathsf{st}', \mathsf{pt}, \mathsf{ct}):}$
8: parse $\mathsf{st} = (\mathsf{st}_A, \mathsf{st}_B)$
9: $S.\mathsf{Enc}(\mathsf{st}_A) \to (\mathsf{st}'_A, \mathsf{pt}', \mathsf{ct}')$
10: $S.\mathsf{Dec}(\mathsf{st}_B, \mathsf{ct}') \to (\mathsf{st}'_B, \mathsf{pt}'')$
11: $S.\mathsf{Enc}(\mathsf{st}'_B) \to (\mathsf{st}''_B, \mathsf{pt}, \mathsf{ct})$
12: $\mathsf{st}' \leftarrow (\mathsf{st}'_A, \mathsf{st}''_B)$
13: **return** $(\mathsf{st}', \mathsf{pt}, \mathsf{ct})$

Figure 5.19 – Simple ratchet S to SEQ

5.4 Conclusion of Chapter

We defined a self-encryption mechanism involving a device which encrypts a secret message for herself to use in the future. We are interested in security when the state of a device in such settings leaks causing the leakage of the secret message. We started giving some instances where self-ratcheting finds applications in cloud storage, when a client encrypts files to be stored, and in 0-RTT session resumption, when a server encrypts a resumption key to be kept by the client. Unlike previous works which focused on forward secrecy and resistance to replay attacks, we studied how to add post-compromise security, as well.

We first proved that post-compromise security implies a super-linear state size in terms of the number of ciphertexts which can still be decrypted by the state. We then give formal definitions of self-ratchet. We finally showed how to design a secure scheme satisfying our bound on the state size.

Furthermore, we showed that our results on the growth of state size matches with existing secure bidirectional secure messaging applications. Given the fact that the messaging applications provide different level of PCS, we observed that there exist some protocols such as ACD without growing state size. It is due to the fact that the protocol is secure with a weaker notion of PCS which could allow constant-size states. It would be interesting to investigate weaker PCS notions in self-encryption applications such as cloud storage or 0-RTT.

6 Conclusion

In this dissertation, we have studied several problems in cryptography which are related to sending secrets to the future.

In the first part, we proposed a timed-release encryption scheme with master time bound key. By using the master time bound key, a ciphertext could be decrypted even though the release time is unknown. This can be an advantage in the case when it is more important not to lose the data. We also proved that our construction is secure under the bilinear Diffie-Hellman hardness assumption. In our experimental result, we studied the change of execution time of our construction by security level. In the real world application, we can make the computations faster by parallelizing some computations or with precomputations.

In the second part, we defined the notion of hidden group with hashing, and built a witness key encapsulation model on top of it. We have proven that the underlying hard problem of our witness key encapsulation model is NP-complete, so that we can generate a ciphertext from any NP instance. Moreover, we have analyzed the timed-release encryption from witness encryption and the Bitcoin blockchain. In this case, there is no trusted server in the system and Bitcoin miners replace the role of trusted server. We however have shown that this instantiation is costly due to the structure of SHA-256, and proposed a way to improve it by replacing SHA-256 by another hash function.

In the third part, we studied post-compromise security in self-encryption. We have proven that we need a state which super-linearly increases in terms of the number of active ciphertexts, i.e. ciphertexts which can be correctly decrypted with the current state, in order to achieve post-compromise security and forward secrecy at the same time in self-encryption. We then applied this result to several cases.

Future Work

In cryptography, there still exist plenty of problems which are related to time. One of common problem in cryptography, also in any field of computer science, is to improve the *efficiency* of a protocol or a system, and the execution time is usually a part of *efficiency*. We therefore can say that we always have some time-related problems to solve in cryptography.

Conversely, there also exist some algorithms whose goal is to slow down the system. For example, there exists a time-lock puzzle [RSW96, BGJ+16, MT19] which wants to control the time to solve a puzzle and a verifiable delay function [BBBF18, Wes19, DFMPS19, Pie18] which wants to control the time to generate a proof. For both of them, one of interesting research direction is to find a construction which is post-quantum secure.

This is also the same for our research. The timed-release encryption scheme and the witness key encapsulation model that we proposed in this dissertation are based on hardness assumptions which can be broken by a quantum computer. Therefore, in the post-quantum era, both of them will be insecure. Some researches might be needed to investigate if we can build a construction which is post-quantum secure.

For the timed-release encryption scheme with master time bound key, we needed a master time bound key to decrypt any ciphertext as it is hard to extract the associated release time from a ciphertext. The introduction of the master time bound key introduced a new problem as the trusted server needs to decide in which case it will accept the decryption with the master time bound key and in which case it will reject. Ideally, the trusted server should accept if the release time of ciphertext is already passed, but there is no efficient way to check it from the ciphertext. We will therefore require a way to efficiently verify it.

A Source Codes

A.1 TRE Benchmark Source Code

```
1  import hashlib
2  import time
3
4  def setup(lbd, lbd2):
5      q = random_prime(2^lbd, lbound=2^(lbd-1))
6      p = 6*randint(2^(lbd2-lbd-1), 2^(lbd2-lbd))*q-1
7      while not p.is_pseudoprime():
8          q = random_prime(2^lbd, lbound=2^(lbd-1))
9          p = 6*randint(2^(lbd2-lbd-1), 2^(lbd2-lbd))*q-1
10     K = GF(p)
11     R.<y> = K[]
12     K = K.extension(R.irreducible_element(2), 'x')
13     E = EllipticCurve(K, [0, 1])
14     return (K, E, q, R)
15
16 def keygen_TS(parms):
17     K = parms[0]
18     E = parms[1]
19     q = parms[2]
20     R = parms[3]
21
22     P = None
23     while P is None:
24         try:
25             a = K(R.random_element(1))
26             P = E([a, sqrt(a^3 + 1)])
27         except:
```

```
28              pass
29      P = ((K.characteristic()+1)//q)*P
30      while P == E(0):
31          P = None
32          while P is None:
33              try:
34                  a = K(R.random_element(1))
35                  P = E([a, sqrt(a^3 + 1)])
36              except:
37                  pass
38          P = ((K.characteristic()+1)//q)*P
39
40      Q = randint(1,q-1)*E([P.xy()[0]^K.characteristic(), P.xy
          ()[1]^K.characteristic()]) + randint(0,q-1)*P
41      zq = Integers(q)
42      a = zq.random_element()
43      b = zq.random_element()
44      c = zq.random_element()
45      d = zq.random_element()
46      while a == 0 or b == 0 or c == 0 or d == 0 or a*b == 1 or
          c*a == d or b*d == c:
47          a = zq.random_element()
48          b = zq.random_element()
49          c = zq.random_element()
50          d = zq.random_element()
51
52      a = a.lift()
53      b = b.lift()
54      c = c.lift()
55      d = d.lift()
56      pk1 = b*P+Q
57      return (a,b,c,d,P,Q), ((P+a*Q).weil_pairing(pk1, q), pk1,
          c*P+d*Q)
58
59  def broadcast(sk, t, parms):
60      K = parms[0]
61      E = parms[1]
62      q = parms[2]
63      R = parms[3]
64
65      a = sk[0]
66      b = sk[1]
```

```
67      c = sk[2]
68      d = sk[3]
69      P = sk[4]
70      Q = sk[5]
71      zq = Integers(q)
72
73      s = zq.random_element()
74      while s == 0:
75          s = zq.random_element()
76      s = s.lift()
77
78      if t == zq(-d):
79          print "-d"
80          return s*P + (a*b - 1)*power_mod(c+b*t, -1, q)*Q
81      elif t == zq(-c*power_mod(b,-1,q)):
82          print "-cb^-1"
83          return (1-a*b)*power_mod(d+t,-1,q)*P + s*Q
84      else:
85          return s*power_mod(d+t,-1,q)*P + (s+a*b-1)*power_mod(
              c+b*t,-1,q)*Q
86
87  def enc(pk1, pk2, m, t, parms):
88      K = parms[0]
89      E = parms[1]
90      q = parms[2]
91      R = parms[3]
92
93      m = K(m)
94
95      r1 = Integers(q).random_element()
96      while r1 == 0:
97          r1 = Integers(q).random_element()
98      r1 = r1.lift()
99
100     r2 = K(R.random_element(1))
101
102     wp = pk1[0]^r1
103
104     ct0 = m * r2
105     ct1 = r1*(t*pk1[1] + pk1[2])
106     ct2 = r2 + wp
107     return (ct0, ct1, ct2), r1, r2
```

```
108
109 def dec(sk2, tau, ct, parms):
110     K = parms[0]
111     E = parms[1]
112     q = parms[2]
113
114     wp = tau.weil_pairing(ct[1], q)
115     return ct[0]/(ct[2] - wp)
116
117 lbd = 160   # log_2 q
118 lbd2 = 512  # log_2 p
119
120 t_setup = []
121 t_keygen = []
122 t_broadcast = []
123 t_enc = []
124 t_dec = []
125
126 it = 1000
127
128
129 for i in xrange(it):
130     print 'Iteration %03d'% i
131     start = time.time()
132     parms = setup(lbd, lbd2)
133     end = time.time()
134     t_setup.append(end - start)
135
136     m = parms[0](parms[3].random_element(1))
137     t = Integers(parms[2]).random_element().lift()
138
139     start = time.time()
140     sk, pk = keygen_TS(parms)
141     end = time.time()
142     t_keygen.append(end - start)
143
144     start = time.time()
145     tau = broadcast(sk, t, parms)
146     end = time.time()
147     t_broadcast.append(end - start)
148
149     start = time.time()
```

```
150     ct, r1, r2 = enc(pk, 0, m, t, parms)
151     end = time.time()
152     t_enc.append(end - start)
153 
154     start = time.time()
155     pt = dec(0, tau, ct, parms)
156     end = time.time()
157     t_dec.append(end - start)
158 
159     if pt != m:
160         print "decryption error"
161 
162     print "    Setup = %.06f"% mean(t_setup)
163     print "   Keygen = %.06f"% mean(t_keygen)
164     print "Broadcast = %.06f"% mean(t_broadcast)
165     print "  Encrypt = %.06f"% mean(t_enc)
166     print "  Decrypt = %.06f"% mean(t_dec)
167     print "="*30
```

Bibliography

[AB09] Shweta Agrawal and Xavier Boyen. Identity-based encryption from lattices in the standard model. *Manuscript, July*, 2009.

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 553–572. Springer, 2010.

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol. In *Advances in Cryptology – EUROCRYPT 2019*, LNCS. Springer, 2019.

[AF07] Masayuki Abe and Serge Fehr. Perfect NIZK with adaptive soundness. In *Theory of Cryptography Conference*, pages 118–136. Springer, 2007.

[AGJ19] Nimrod Aviram, Kai Gellert, and Tibor Jager. Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 117–150, Cham, 2019. Springer International Publishing.

[BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *Annual International Cryptology Conference*, pages 443–459. Springer, 2004.

[BBB+12] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management part 1: General (revision 3). *NIST special publication*, 800(57):1–147, 2012.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual International Cryptology Conference*, pages 757–788. Springer, 2018.

[BC04] Ian F. Blake and Aldar C.-F. Chan. Scalable, server-passive, user-anonymous timed release public key encryption from bilinear pairing. *IACR Cryptology ePrint Archive*, 2004:211, 2004.

[BDJR97] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 394–403. IEEE, 1997.

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 345–356, 2016.

[bit] Bitcoin developer reference. https://bitcoin.org/en/developer-reference. Accessed 28 January 2020.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key Homomorphic PRFs and Their Applications . In *Advances in Cryptology – CRYPTO 2013*, LNCS. Springer, 2013.

[CDV19] Andrea Caforio, F. Betül Durak, and Serge Vaudenay. On-Demand Ratcheting with Security Awareness. Cryptology ePrint Archive, Report 2019/965, 2019. https://eprint.iacr.org/2019/965.

[CGCG16] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On post-compromise security. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 164–178, June 2016.

[CHKO06] Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Timed-release and key-insulated public key encryption. In *International Conference on Financial Cryptography and Data Security*, pages 191–205. Springer, 2006.

[CHS07] Konstantinos Chalkias, Dimitrios Hristu-Varsakelis, and George Stephanides. Improved anonymous timed-release encryption. In *European Symposium on Research in Computer Security*, pages 311–326. Springer, 2007.

[CJK19] Peter Chvojka, Tibor Jager, and Saqib A Kakvi. Offline witness encryption with semi-adaptive security. 2019.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *Advances in Cryptology - CRYPTO 2002*, LNCS. Springer, 2002.

[CLQ05] Julien Cathalo, Benoît Libert, and Jean-Jacques Quisquater. Efficient and non-interactive timed-release encryption. In *International Conference on Information and Communications Security*, pages 291–303. Springer, 2005.

[COR99] Giovanni Di Crescenzo, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 74–89. Springer, 1999.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Annual International Cryptology Conference*, pages 13–25. Springer, 1998.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 45–64. Springer, 2002.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[Dam91] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *Annual International Cryptology Conference*, pages 445–456. Springer, 1991.

[Den06] Alexander W Dent. The hardness of the DHK problem in the generic group model. *IACR Cryptology ePrint Archive*, 2006:156, 2006.

[DFMPS19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 248–277. Springer, 2019.

[DG17] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In *Annual International Cryptology Conference*, pages 537–569. Springer, 2017.

[DJSS18] David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 425–455, Cham, 2018. Springer International Publishing.

[DT07] Alexander W. Dent and Qiang Tang. Revisiting the security model for timed-release encryption with pre-open capability. In *International Conference on Information Security*, pages 158–174. Springer, 2007.

[DV19] F. Betül Durak and Serge Vaudenay. Bidirectional Asynchronous Ratcheted Key Agreement with Linear Complexity. In *Advances in Information and Computer Security – IWSEC 2019*, LNCS. Springer, 2019.

[EPRS17] Adam Everspaugh, Kenneth Paterson, Thomas Ristenpart, and Sam Scott. Key Rotation for Authenticated Encryption. In *Advances in Cryptology – CRYPTO 2017*, LNCS. Springer, 2017.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 467–476. ACM, 2013.

[GHJL17] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT key exchange with full forward secrecy. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 519–548, Cham, 2017. Springer International Publishing.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run Turing machines on encrypted data. In *Annual Cryptology Conference*, pages 536–553. Springer, 2013.

[HARV19] Ralph Holz, Johanna Amann, Abbas Razaghpanah, and Narseo Vallina-Rodriguez. The Era of TLS 1.3: Measuring Deployment and Use with Active and Passive Methods. *CoRR*, 2019.

[HYL05] Yong Ho Hwang, Dae Hyun Yum, and Pil Joong Lee. Timed-release encryption with pre-open capability and its application to certified e-mail system. In *International Conference on Information Security*, pages 344–358. Springer, 2005.

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient Ratcheting: Almost-Optimal Guarantees for Secure Messaging. In *Advances in Cryptology – EUROCRYPT 2019*, LNCS. Springer, 2019.

[JPV00] Marc Joye, Pascal Paillier, and Serge Vaudenay. Efficient generation of prime numbers. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 340–354. Springer, 2000.

[JS18] Joseph Jaeger and Igors Stepanovs. Optimal Channel Security Against Fine-Grained State Compromise: The Safety of Messaging . In *Advances in Cryptology – CRYPTO 2018*, LNCS. Springer, 2018.

[Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[KME+16] Kohei Kasamatsu, Takahiro Matsuda, Keita Emura, Nuttapong Attrapadung, Goichiro Hanaoka, and Hideki Imai. Time-specific encryption from forward-secure encryption: generic and direct constructions. *International Journal of Information Security*, 15(5):549–571, 2016.

[LJKW18] Jia Liu, Tibor Jager, Saqib A Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 86(11):2549–2586, 2018.

[LP19] Chao Li and Balaji Palanisamy. Silentdelivery: Practical timed-delivery of private information using smart contracts. *arXiv preprint arXiv:1912.07824*, 2019.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23. Springer Berlin Heidelberg, 2010.

[M+86] Victor Miller et al. Short programs for functions on curves. *Unpublished manuscript*, 97(101-102):44, 1986.

[May93] Timothy C May. Timed-release crypto. http://www.hks.net.cpunks/cpunks-0/1460.html, 1993.

[Mer78] Ralph C Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.

[Mer87] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology – CRYPTO 1987*, LNCS. Springer, 1987.

[MNM10] Takahiro Matsuda, Yasumasa Nakai, and Kanta Matsuura. Efficient generic constructions of timed-release encryption with pre-open capability. In *International Conference on Pairing-Based Cryptography*, pages 225–245. Springer, 2010.

[MOV93] Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Information Theory*, 39(5):1639–1646, 1993.

[MT19] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In *Annual International Cryptology Conference*, pages 620–649. Springer, 2019.

[Nak19] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *Annual International Cryptology Conference*, pages 96–109. Springer, 2003.

[NDHC18] Jianting Ning, Hung Dang, Ruomu Hou, and Ee-Chien Chang. Keeping time-release secrets through smart contracts. *IACR Cryptology ePrint Archive*, 2018:1166, 2018.

[NMKM09] Yasumasa Nakai, Takahiro Matsuda, Wataru Kitada, and Kanta Matsuura. A generic construction of timed-release encryption with pre-open capability. In *International Workshop on Security*, pages 53–70. Springer, 2009.

[Pie18] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[PQ10] Kenneth G Paterson and Elizabeth A Quaglia. Time-specific encryption. In *International Conference on Security and Cryptography for Networks*, pages 1–16. Springer, 2010.

[PR06] Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *Theory of Cryptography*, pages 145–166. Springer Berlin Heidelberg, 2006.

[PR18] Bertram Poettering and Paul Rösler. Towards Bidirectional Ratcheted Key Exchange . In *Advances in Cryptology – CRYPTO 2018*, LNCS. Springer, 2018.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.

[Sil09] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer Science & Business Media, 2009.

[Sys17] Open Whisper Systems. Signal protocol library for Java/Android. GitHub repository https://github.com/WhisperSystems/libsignal-protocol-java, 2017.

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 379–407. Springer, 2019.

[WS07] Jiang Wu and Douglas R Stinson. An efficient identification protocol and the knowledge-of-exponent assumption. *IACR Cryptology ePrint Archive*, 2007:479, 2007.

Curriculum Vitae

Gwangbae Choi

Email: gwangbae.choi@hotmail.com
Nationality: South Korea
Date of Birth: 30th of May 1990

Education

2016-2020	Ph.D in **Computer and Communication Sciences** Area: **Cryptography** Laboratory of Security and Cryptography **Ecole Polytechnique Fédérale de Lausanne**
2014-2016	Master in **Communication System** Specialization: **Information Security** **Ecole Polytechnique Fédérale de Lausanne**
2010-2014	Bachelor in **Communication System** **Ecole Polytechnique Fédérale de Lausanne**
2009-2010	Cours de Mathématiques Spéciales (CMS) **Ecole Polytechnique Fédérale de Lausanne**

Publications

Choi, G. and Vaudenay, S., 2019. *Timed-Release Encryption With Master Time Bound Key (Extended)*. J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl., 10(4), pp.88-108.

Choi, G. and Vaudenay, S., 2019, August. *Timed-Release Encryption With Master Time Bound Key*. In International Workshop on Information Security Applications (pp. 167-179). Springer, Cham.

Raisaro, J.L., Choi, G., Pradervand, S., Colsenet, R., Jacquemont, N., Rosat, N., Mooser, V. and Hubaux, J.P., 2018. *Protecting Privacy and Security of Genomic Data in i2b2 with Homomorphic Encryption and Differential Privacy.* IEEE/ACM transactions on computational biology and bioinformatics, 15(5), pp.1413-1426.

Choi, G., Raisaro, J.L., Pradervand, S., Colsenet, R., Jacquemont, N., Rosat, N. and Hubaux, J.P., 2016. *Privacy-preserving exploration of genetic cohorts with i2b2 at lausanne university hospital*. In Proceedings of the 3rd International Workshop on Genome Privacy and Security (GenoPri'16) (Vol. 27).

Programming Languages

Java, Scala, C, Matlab, Python, Assembly, VHDL, Perl, SQL

Languages

Korean	Mother tongue
English	B2-C1 (Upper intermediate – Advanced)
French	B2-C1 (Upper intermediate – Advanced)
Japanese	B2 (Upper intermediate)