Extending the Functionality and Security of Time-Based Primitives

Jodie Knapp

Information Security Group
Royal Holloway, University of London

This dissertation is submitted for the degree of
Doctor of Philosophy

August 2023

Declaration

These doctoral studies were conducted under the supervision of Dr Elizabeth A. Quaglia. The contents of this thesis is the result of original research carried out by myself whilst enrolled in the Information Security Group as a candidate for the degree of Doctor of Philosophy. The results presented are my own and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This work has not been submitted for any other degree or award in any other university or educational establishment.

Jodie Knapp
August 2023

Acknowledgements

To start, I would like to thank my supervisor Dr Elizabeth A. Quaglia for her continued support throughout my four years working with her. Liz is responsible for shaping me as a researcher and persevering with my dry, overly wordy writing to produce meaningful work. I have been influenced by your enthusiasm for research, and your supportive, empathetic mentoring. I look up to you as a strong woman smashing it in academia and wider life.

In equal measure, I want to endlessly thank my parents and sisters Stacey and Laura for their support, putting up with the less pleasant side of me and acting as a much-needed lifeboat in times of stress even if they're oceans away. A special mention, upon her insistence, to Stacey for critiquing my acknowledgements. Without the four of you, I know I wouldn't have completed my undergraduate degree let alone gone on to do all the amazing things I've been fortunate to experience.

The aspect of my PhD that I will treasure the most is the core memories made with my peers. I hope those I met know how much I have valued spending time with them - from deep chats to travelling across continents, living together, card nights, weddings, and generally having fun. The list of good times is endless; however, the process has not been without hard times. I believe with every breakdown comes a breakthrough and I'm glad to have gone on this seemingly never-ending journey with you all.

This leads to my final thank you and that is to the department for accepting my application and providing me with this opportunity. I feel very fortunate to have gained so much work experience, advice, and air miles over the last five years and will never fail to appreciate how much of my future success, both professionally and personally, will be because of my time with the CDT. Thank you.

Abstract

The overarching theme of this Thesis is to explore the role of time in cryptography with an emphasis on protocol design, security modelling, and reducing assumptions of trust placed upon external entities or users.

The first primitive we focus on in-depth is *updatable encryption*. At the heart of updatable encryption is the timely transformation of all encrypted information via outsourced key rotation. To do so requires cryptographic elements known as tokens which are used by untrusted entities to perform the updates such that they learn nothing of the underlying information. Despite the benefits updatable encryption promises, research is concerned with the inferable information an adversary can realise without the need to corrupt cryptographic elements. Thus, the focal point of literature is achieving strong security notions such as ciphertext unlinkability, which intuitively guarantees fresh and updated ciphertexts are indistinguishable. Traditionally, updatable encryption (UE) is designed as a symmetric primitive. In this Thesis, we consider updatable encryption in the context of public-key encryption (PKUE) and present a rigorous model of security to attain ciphertext unlinkability. We build upon our framework to consider the leakage of ciphertext age by capturing the notion we name epoch confidentiality.

We extend our understanding of the PKUE primitive by considering the trust given to entities in the broader public key infrastructure (PKI) to which a PKUE scheme will be applied. In particular, we seek to reduce trust in the key generation centre of a PKI in our work on certificateless public key updatable encryption. Second, we explore reducing trust in the server performing updates for a PKUE scheme in our work to define multi-server PKUE.

The second direction of interest in our work is secret-sharing schemes. In essence, this protocol distributes segments of a secret value to several shareholders to mitigate the risk of adversarial corruption of highly sensitive information. In particular, we examine

the usefulness of a time-delay mechanism to achieve desirable properties related to the fair and sound reconstruction of a secret. To do so, we construct a scheme using homomorphic time-lock puzzles which are assumed to take a threshold amount of time to solve. Not only this, we do not consider honest shareholders and instead assume rational player participation in a secret-sharing game whereby individuals may seek to mislead others for an incentive. Exploring secret sharing in a game theory sense allows us to capture a realistic scenario in which secret sharing may be applied.

Contents

List of figures

List of tables

Chapter 1

Introduction

This Chapter motivates the work presented in this Thesis and outlines the structure.

1.1 Motivation

Time is a fundamental aspect of communication. Depending on the application and security requirements, there may be a delay in time before the information becomes useful, or information may only be available for a limited period, after which it becomes redundant. By way of illustration, consider the need for outsourced storage of encrypted information like medical data, which is of a sensitive nature. Storing data over a long period increases the time an adversary has to attack security, meaning the risk of cryptographic key exposure is high. Cryptographic primitives such as updatable encryption [24, 92] aim to mitigate this risk by dividing the time the encrypted data is stored under a specific key, achieved by a rotation functionality. Thus, the exposure of a key is only useful for a distinct period, after which the corruption is redundant.

An equally important consideration in modern cryptography is the preservation of privacy. To achieve the privacy of sensitive information, protocol security modelling attempts to maintain the secrecy of the data. The omission of this property in a cryptographic scheme can result in a malicious entity learning the underlying information, with potentially grave consequences. One such cryptographic protocol in which privacy is a central motivator is secret sharing [110, 16]. The procedure of secret distribution was introduced to alleviate concerns about a single point of failure if the secret is corrupted. In essence, secret sharing can be viewed as a method to split and distribute

a secret, such as a cryptographic key encrypting important data, across multiple entities who may or may not be trusted.

The interplay between *time* and *privacy* in a cryptosystem can be complex. This Thesis delves into the role that both properties play in cryptographic schemes, with a focus on extending the definitions and security modelling of timely primitives, as well as utilising time delay to attain desirable security properties. In particular, we explore the distinct updatable encryption and secret-sharing primitives mentioned above.

The *first* part of our work explores extensions of the *updatable encryption* primitive. Traditionally, updatable encryption was designed as a symmetric-key encryption primitive and significantly, a large focus of the literature has been on modelling security. Of utmost importance in updatable encryption is the following question:

How can we outsource encrypted data over long periods in an *untrusted* environment, *and* maintain the confidentiality of the information?

We were interested to explore the possible extensions and new security goals achievable when redefining the primitive in the public-key setting.

Contributions in Updatable Encryption In this Thesis, we establish a definition for a public-key updatable encryption primitive. Further, we present a model of security for our primitive and define important security notions relevant to the public-key setting. The idea of *ciphertext unlinkability*, which incorporates the indistinguishability of ciphertexts derived from encryption versus updates, is of paramount importance to the security of an updatable encryption scheme.

We build upon the concept of unlinkability to establish a new security goal coined *epoch confidentiality* which demonstrates the unlinkability of ciphertexts *and* the confidentiality of ciphertext age. In the context of the example we gave at the start of this Chapter regarding medical data, the leakage of ciphertext age can indicate the time an individual has been treated or the individual's age for instance.

Now we have defined public-key updatable encryption (PKUE), we move forward by exploring the primitive in the wider context of a public-key infrastructure (PKI). In practice, applying PKUE in a PKI requires trust in a third party producing the epoch public and secret keys, which is a clear violation of privacy if the key generator behaves maliciously or is corrupted. This issue of keys being held in escrow leads to our second contribution in which our primary concern is to *reduce trust* in the PKI key generator.

Our chosen solution is a new primitive called *certificateless public-key updatable encryption* (CLUE), which is derived from PKUE such that the underlying encryption scheme is certificateless public-key encryption (CL-PKE). Our choice to remove certificates means the key generator does not know the whole epoch secret key, only a partial secret key they extracted, consequently resolving the key escrow problem in the PKUE setting.

At this point in our PKUE research, we have implicitly assumed the server will perform ciphertext updates honestly and so there is a level of trust endowed to this entity. Continuing our thought process of reducing trust in the entities involved in the PKUE primitive, we focus on a real issue that may occur when there is a single point of failure resulting from a server neglecting to update a ciphertext. The consequence in this scenario would essentially mean that the updatable encryption scheme reverts to a standard encryption scheme, ultimately defeating the core purpose of defining the primitive.

Our proposed solution is to define a *multi-server* public-key encryption scheme in which a given committee of servers collaborate to update ciphertexts periodically. It is practical to assume the need for changes in the participating servers over the entirety of the scheme to maintain security given one or more servers are dishonest or corrupted. After careful consideration, we proposed secret-sharing techniques as an ideal foundation to tolerate dynamic multi-server changes in the scheme over time.

Our venture into applying a secret sharing protocol, as a building block for an updatable encryption scheme, ignited our interest to concentrate on the role of time in secret sharing more generally. This leads us to the *second* facet of our work - an exploration of time delay in *secret sharing* protocols. Recollect, secret sharing is a useful tool to ensure privacy. Alas, distributing the secret among several parties does not guarantee this security goal. To see this, once a threshold number of secret shares have been corrupted, an adversary is capable of reconstructing the secret. Note that the adversary may therefore learn vital information before the intended party(s), creating an *unfair* scheme. Secret sharing literature suggests the technique of *time delay* to lessen the chance of an unfair outcome.

Contributions in Secret Sharing Our work on secret sharing incorporates the method of a time delay to satisfy fairness [76] alongside a definition of soundness. We were interested in understanding how fairness and soundness can be achieved

in secret sharing protocols when we model shareholders as *rational* [62, 5]. That is malicious/deviant players in a game but at the very least selfish. To consider rational shareholders, we start by viewing secret sharing through a game theory lens. In doing so, we can gather information about the strategies and outcomes of players in the secret-sharing scheme. At a high level, we employ homomorphic time-lock puzzles [97] on the secret shares, to encrypt the shares into lots of computationally hard puzzles, which means there is a time delay in any entity determining the secret value. As we will discuss, there are challenges in maintaining the security of a secret sharing scheme over a long period, whilst attaining desirable properties and increasing the efficiency of computation.

We now present a summary of the ensuing Chapters proceeded by details of the published work contained within this Thesis.

1.2 Chapter Summary

In Chapter 2 we present preliminaries used throughout this Thesis. Topics covered include general notation, fundamental cryptographic building blocks, specific primitives and hardness assumptions.

In Chapter 3 we formalise *public-key updatable encryption* (PKUE), a primitive so far studied formally only in the symmetric setting. UE outsources the periodic rotation of an encryption key to an untrusted party, transforming a ciphertext encrypted under an old key into a ciphertext encrypted under a new key. Crucially, the party updates the ciphertext without the need to decrypt it, using a tool called an update token. There are two distinct strains of a UE scheme; ciphertext-dependent updates and ciphertext-independent updates. We centre on the latter strain, which essentially generates a single token capable of updating all ciphertexts, and model security notions for the new public-key UE primitive analogous to the symmetric ciphertext-independent UE literature [92, 80].

Defining UE in the public-key setting enables us to establish a new notion of security we call *epoch confidentiality* (EC) which considers the ability of an adversary to distinguish the public keys used in periods known as epochs and in turn reflects the leakage of the time in which a ciphertext was created. Lastly, we propose a public-key UE construction and prove that it satisfies our new notion of security alongside a notion of

ciphertext confidentiality such that efficiency is not affected by moving to the public-key setting.

In Chapter 4 we formalise *certificateless public-key updatable encryption* (CLUE). At a high level, the CLUE primitive functions like the PKUE primitive defined in Chapter 3, except the underlying encryption scheme is certificateless public-key encryption primitive (CL-PKE) [3]. We chose this primitive because by design the key generation centre (KGC) extracts a partial secret key that the data owner uses as an input to generate their epoch public/secret key pairs and update tokens. Important to the security of a PKUE scheme in instances where we do not trust the KGC, our CLUE primitive is designed such that the KGC does not generate the full epoch key pairs and so the aforementioned key escrow issue does not arise.

Furthermore, we provide a security model for CLUE to capture the inferable information at an adversary's disposal due to update token and key corruption, and we propose a concrete construction of CLUE taking inspiration from the certificateless encryption scheme proposed by the authors of [93]. Finally, we prove our construction satisfies the security notion of *ciphertext indistinguishability* that we formalised, following the modular approach given by [80] to reduce security from the updatable to the static setting.

In Chapter 5 we propose a *Dynamic Multi-Server* Updatable Encryption (DMUE) primitive as an extension of standard public-key updatable encryption (PKUE) defined in Chapter 3. Recall, traditional UE aims to have efficient ciphertext updates performed by an untrusted server such that the compromise of several cryptographic keys and *update tokens* must not reduce the standard security of encryption. To mitigate the risk of a single point of failure in single-server UE and thus improve the resilience of the scheme, we formalise a multi-server variant of PKUE to treat the issue of token leakage. We can achieve a distributed update process by providing each server with an update token and requiring a *threshold* of servers to engage honestly. However, servers may act dishonestly or need to be replaced over time, so our primitive must cater to dynamic committee changes in the servers participating across epochs. Inspired by the work of [15] in the context of secret sharing, we propose a *generic* DMUE scheme built from PKUE and dynamic proactive secret sharing primitives. In turn, we prove the ciphertext unlinkability of freshly encrypted versus updated ciphertexts.

In Chapter 6 we turn our attention to secret sharing protocols. In particular, we continue to focus on the impact of time in cryptographic schemes and protocols, utilising time-delay building blocks to satisfy important secret-sharing properties. In other words, achieving the properties of fairness and soundness has proved to be challenging in non-simultaneous rational secret-sharing schemes. To overcome this challenge, a solution suggested in the literature is to employ a time-delay mechanism. We propose a new approach to achieve such delay, namely using homomorphic time-lock puzzles (HTLPs), and constructing a fair and sound rational secret-sharing scheme in the non-simultaneous setting from HTLPs.

HTLPs are used to embed sub-shares of the secret for a predetermined time. This allows for restoring fairness of the secret reconstruction phase, despite players having access to information related to the secret which is required to ensure the soundness of the scheme. Key to our construction is the fact that the time-lock puzzles are homomorphic so that players can compactly evaluate sub-shares. Without this efficiency improvement, players would have to independently solve each puzzle sent from the other players to obtain a share of the secret, which would be computationally inefficient.

In this Chapter we start by defining a fair rational secret sharing scheme (FRSS), in the non-simultaneous setting, built from threshold secret sharing and an HTLP. In doing so, our FRSS scheme can achieve the properties of fairness and soundness. Next, we provide an instantiation of our scheme and demonstrate provable security and efficiency improvements compared to other rational secret-sharing literature incorporating a time delay [96, 40].

To be succinct, we were driven to propose new ideas to research spaces related to time because we believe it is intellectually challenging and highly relevant to modern-day cryptography with the increasing usage of long-term storage of sensitive information and reliance on digital technologies to enhance modern-day life. In particular, we took an interest in designing protocols, proposing new security notions, and re-imagining the context in which time-based primitives are used in the hope that our ideas will inspire and gain traction in the wider cryptographic research community.

1.3 Publications

Chapter 3 is based on the paper "Epoch Confidentiality in Updatable Encryption" which was completed under the supervision of Elizabeth A. Quaglia. This paper [81] was published and presented in ProvSec 2022.

Chapter 4 is based on the paper titled "CLUE: Certificateless Updatable Encryption" which was completed under the supervision of Elizabeth A. Quaglia. This paper [82] is due to be published in ITASEC 2023.

Chapter 5 is based on the paper titled "Dynamic Multi-Server Updatable Encryption". This paper follows on from our contributions in Chapter 3 and it was completed under the supervision of Elizabeth A. Quaglia. The paper is currently under submission.

Chapter 6 is based on the paper "Fair and Sound Secret Sharing from Homomorphic Time-Lock Puzzles" which was completed under the supervision of Elizabeth A. Quaglia. This paper [83] was published and presented in ProvSec 2020.

Chapter 2

Preliminaries

Contents

This Chapter introduces standard notation, fundamental definitions, building blocks and concepts used throughout this Thesis.

2.1 Notation

In this Section, we present the notation that will be used throughout this Thesis.

Sets Let $\mathbb{N}$ denote the set of natural numbers $\{0, 1, 2, \ldots\}$, $\mathbb{Z}$ denotes the set of integers with $\mathbb{Z}^* = \{0\} \cup \mathbb{Z}^+$ defining the set of non-negative integers which itself is the union of zero and the set of positive integers respectively. The set of real numbers is $\mathbb{R}$ and the set of non-negative real numbers is denoted $\mathbb{R}^+ = \{x : x \in \mathbb{R}, x > 0\}$. For some $n \in \mathbb{N}$ let $[n] = \{0, 1, \ldots, n\}$, and $[i, j]$ is the set $\{i, \ldots, j\}$ of consecutive integers $(i, j) \in \mathbb{Z}$ for $i \leq j$.

Variables and Strings The set of all binary bit-strings of length n is denoted $\{0,1\}^n$, $\{0,1\}^*$ denotes the set of all finite bit-strings, and 1^λ defines a bit-string composed of λ many ones, where λ is a security parameter. Moreover, $|x|$ is the length of the bit-string x, if $x \in \mathbb{R}$ then $|x|$ is the absolute value, and $|S|$ is the cardinality of a set S.

Let $x \leftarrow s$ denote the assignment of value s to variable x, $x \leftarrow S$ denotes the uniformly chosen assignment of an element of a finite set S to variable x, and $x \xleftarrow{\$} S$ is the assignment of an element from set S to variable x, which has been uniformly chosen at random.

Algorithms If A is an algorithm, then $y \leftarrow A(x_1, \ldots, x_n)$ is the deterministic evaluation of A on inputs $(x_1, \ldots, x_n)$ with output y. Notation $y \xleftarrow{\$} A(x_1, \ldots, x_n)$ is the probabilistic evaluation of A. The term PPT used concerning algorithms is shorthand for *probabilistic polynomial time.*

Functions Function $f : X \to Y$ maps elements of X to Y, in which we define the image $\mathsf{Im}(f) = \{f(x) : x \in X\}$. An arbitrary polynomial is denoted $\mathsf{poly}(n)$ and function $\mathsf{negl} : \mathbb{N} \to R$ is negligible if, for every positive polynomial p, there exists an N such that for all integers $n > N$, $f(n) < \frac{1}{p(n)}$. Throughout this Thesis $\mathsf{negl}(1^\lambda)$ is used to define a negligible function over the security parameter λ.

Probability Let X denote an event and $\overline{X}$ denote the converse that event X does not occur. Then $\Pr[X] = 1 - \Pr[\overline{X}]$.

Given two separate events X_1 and X_2, the event $(X_1 \wedge X_2)$ is the occurrence of both events, thus, probability $\Pr[X_1 \wedge X_2] \leq \Pr[X_1]$ by definition. The events are *independent* of one another if $\Pr[X_1 \wedge X_2] \geq \Pr[X_1] \cdot \Pr[X_2]$.

Given two separate events X_1 and X_2, the event $(X_1 \vee X_2)$ is the occurrence of *either* event (they are disjoint), thus, probability $\Pr[X_1 \vee X_2] \geq \Pr[X_1]$ by definition. The following upper bound also holds by definition,

$$\Pr[X_1 \vee X_2] \leq \Pr[X_1] + \Pr[X_2].$$

The *conditional* probability of events X_1 *given* X_2 is $\Pr[X_1|X_2] = \frac{\Pr[X_1 \wedge X_2]}{\Pr[X_2]}$, by definition and given $\Pr[X]_2 \neq 0$. The ensuing equation follows,

$$\Pr[X_1 \wedge X_2] = \Pr[X_1|X_2] \cdot \Pr[X_2].$$

Asymptotic Notation For functions $f, g : \mathbb{Z}^* \to \mathbb{R}^+$ the following notation is used:

- $f(n) = O(g(n))$ if $\exists c, n' \in \mathbb{Z}^+$ such that $\forall n > n'$ it holds that $f(n) \leq c \cdot g(n)$.
- $f(n) = \Omega(g(n))$ if $\exists c, n' \in \mathbb{Z}^+$ such that $\forall n > n'$ it holds that $f(n) \geq c \cdot g(n)$.

- $f(n) = \Theta(g(n))$ if $\exists c_1, c_2, n' \in \mathbb{Z}^+$ such that $\forall n > n'$ it holds that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$.
- $f(n) = o(g(n))$ if $lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $f(n) = \omega(g(n))$ if $lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

2.2 Provable Security

Careful consideration is needed to define the security of cryptographic primitives to ensure an accurate analysis of the security of constructed schemes in the *real world*. One approach taken to evaluate the security of cryptographic protocols is to define *security games* in which *adversarial corruption* is modelled.

Security games aim to reflect real-life security as closely as possible, using definitions, cryptographic assumptions based on mathematics, adversarial threat modelling and conditions required to meet the chosen level of security to prevent an adversary from succeeding in winning the game. Informally, a game (Exp) occurs between a *challenger* and a probabilistic-polynomial time *adversary* ($\mathcal{A}$) such that the challenger provides inputs to the adversary and the adversary proceeds to output its challenges under corruption restrictions modelled in the security game. Ultimately, the adversary's final output determines their success in breaking the security notion (X) of the given scheme (Π).

Crucially, for a scheme to remain secure for the chosen level of security, the success or advantage (Adv) of the adversary in winning the game (Exp) must be *negligible* over the security parameter (λ). More formally,

$$\mathsf{Adv}^{\mathsf{Exp}}_{\Pi,\mathcal{A}}(1^\lambda) = \Pr[\mathsf{Exp}^{X}_{\Pi,\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(1^\lambda)$$

for some negligible function $\mathsf{negl}(\cdot)$.

In some instances, a security game is defined as an *indistinguishability experiment*. That is, $\mathcal{A}$ must distinguish which game they are playing according to a randomly selected bit $b \xleftarrow{\$} \{0, 1\}$ chosen by the challenger. Informally, the experiment returns $\mathcal{A}$'s guess bit b' and $\mathcal{A}$ is considered successful if $b' = b$. Formally, $\mathcal{A}$'s advantage in an indistinguishability experiment (X) against protocol Π is defined as

$$\mathsf{Adv}^{\mathsf{Exp},b}_{\Pi,\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{X,0}_{\Pi,\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{X,1}_{\Pi,\mathcal{A}}(1^\lambda) = 1]|.$$

Notably, success is assumed to be negligible for protocol Π to satisfy security notion X. Formally, for some negligible function $\mathsf{negl}(\cdot)$ over security parameter λ,

$$\mathsf{Adv}^{\mathsf{Exp},b}_{\Pi,\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{X,b}_{\Pi,\mathcal{A}}(1^\lambda) = 1] - \tfrac{1}{2}| \leq \mathsf{negl}(1^\lambda).$$

Security Modelling Security can be modelled in the *standard* or *random oracle* setting, which we discuss in more depth below. In the latter, the adversary can make *queries* to *oracles* ($\mathcal{O}$) emulating cryptographic functions to return output to the adversary. Moreover, proofs of security often use a *reduction* technique, that is, reducing the security of the scheme to underlying hardness assumptions. Throughout this Thesis, we will utilise the aforementioned techniques of capturing security games in the standard and random oracle model and proving by reduction.

Hash Functions Hash functions are an essential building block to many cryptographic protocols. Informally, a hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^l$ maps long input strings to output a *compressed* fixed-length string known as a *digest*. Assuming the domain of H is larger than the range $\{0,1\}^l$, two input strings can map to the same digest, therefore, collision resistance is an essential property for secure cryptographic hash functions. More formally,

Definition 1 (Collision Resistance)**.** *A cryptographic hash function H is collision-resistant if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function over the security parameter λ such that,*

$$\Pr[(x,y) \xleftarrow{\$} \mathcal{A}(\cdot) : \mathsf{H}(x) = \mathsf{H}(y) \wedge (x \neq y)] \leq \mathsf{negl}(1^\lambda).$$

Random Oracle Model Proving the security of cryptographic schemes built from hash functions, assuming collision resistance alone, is typically insufficient in the standard model. A solution is to prove security in the random oracle model (ROM) instead, whereby we treat the hash function H as a *truly random* function that is evaluated by *querying* an oracle. The oracle can be thought of as a consistent *black box* in which the output is $\mathsf{H}(x)$ on a queried input of $x \in \{0,1\}^*$.

Practically speaking, a scheme in the ROM is implemented with an appropriately chosen cryptographic hash function H' such that any PPT adversary $\mathcal{A}$ can compute $\mathsf{H}'(x)$ for any input $x \in \{0,1\}^*$. In line with the formal definition presented in [14] the following properties must be satisfied upon adversarial queries:

- **Uniformity** If x has previously been queried, then output $\mathsf{H}(x)$ is always returned. If x has not been queried then H must uniformly choose at random the output $\mathsf{H}(x)$ and store $(x, \mathsf{H}(x))$ for future reference.
- **Extractability** If x is queried to H then a reduction proof of security can see the query and learn the input x.
- **Programmability** The reduction can set the output value $\mathsf{H}(x)$ to a value of its choice, provided the output is distributed properly.

In essence, the core purpose of proving security in the random oracle model is to demonstrate that there are no inherent flaws in the design of a cryptographic protocol as opposed to proving the security of an implemented scheme in the real world. In the context of provable security, it is required that H' is a sufficiently good replica of an oracle, however, this assumption is not rigorous in either a mathematical or heuristic sense. This leads to the general debate within the cryptographic community regarding the usefulness of proving security in the random oracle model.

In more words, the random oracle model is deemed incapable of guaranteeing provable security when instantiated in the so-called real world since there does not exist a truly random hash function H'. Research supporting this opinion includes that from the authors of [27], however, it is important to note that proposed schemes are somewhat orchestrated and more importantly, there have been no successful real-world attacks on schemes whose security has been proven in the random oracle model. Further, supportive evidence includes the recent work from the authors of [26] on *global* random oracles in which they demonstrate the ability to generically prove the existence of practical realisations of a number of essential cryptographic primitives including PKE schemes in the random oracle model.

2.3 Cryptographic Hardness Assumptions

In the following, we present the number theoretic notation, cryptographic hardness assumptions, and definitions related to *cyclic* groups which are used in this Thesis.

Let $\mathcal{G}$ be a PPT group generation function outputting the tuple $(\mathbb{G}, q, g)$ which defines the cyclic group $\mathbb{G}$, of prime order q, generated by element $g \in \mathbb{G}$. For example, group $\mathbb{Z}_q$ is the additive group of integers modulo q defined as $\{0, \ldots, q-1\}$. Group $\mathbb{Z}_q^*$

denotes the multiplicative group of invertible integers modulo q, that is, invertible integers that are relatively prime to q. We denote $\mathbb{G} = \{g^0, \ldots, g^{q-1}\}$ and $\forall h \in \mathbb{G}$, there exists a *unique* $x \in \mathbb{Z}_q$ such that $g^x = h$, such that element x is called the *discrete logarithm* of h with respect to g ($x = \log_g(h)$).

The Discrete Logarithm Problem The discrete logarithm problem is assumed to be hard if the following holds.

Definition 2 (DLP)**.** *For security parameter λ, let $\mathcal{G}(1^\lambda) \to (\mathbb{G}, q, g)$. The discrete logarithm assumption holds for tuple $(\mathbb{G}, q, g)$ if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function over security parameter λ such that*

$$\Pr[h \xleftarrow{\$} \mathbb{G}; x \xleftarrow{\$} \mathcal{A}(\mathbb{G}, q, g, h) : g^x = h] \leq \mathsf{negl}(1^\lambda).$$

The Diffie-Hellman Problem The *computational* Diffie-Hellman (CDH) problem is presented in the following.

Definition 3 (CDH Problem)**.** *The CDH assumption holds over cyclic group $(\mathbb{G}, q, g)$ if, given (g^x, g^y) such that $x, y \xleftarrow{\$} \mathbb{Z}_q$, it is computationally intractable for any PPT adversary $\mathcal{A}$ to compute g^{xy}.*

The *Decisional* Diffie-Hellman (DDH) problem is assumed to be hard if, given elements (g^x, g^y) for $x, y \xleftarrow{\$} \mathbb{Z}_q$, element g^{xy} is *indistinguishable* from a random element $g \in \mathbb{G}$. Formally,

Definition 4 (DDH Problem)**.** *The DDH assumption holds over cyclic group $(\mathbb{G}, q, g)$ if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function over security parameter λ such that*

$$|\Pr[x, y \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}(g^x, g^y, g^{xy}) = 1] - \Pr[x, y, z \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}(g^x, g^y, g^z) = 1]| \leq \mathsf{negl}(1^\lambda).$$

Bilinear Maps Alternatively known as *pairings*, we define bilinear maps using multiplicative notation as follows.

Definition 5 (Bilinear Maps)**.** *Let cyclic groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ have prime order q, such that $\mathbb{G}_1$ is generated by P, and $\mathbb{G}_2$ is generated by Q. A pairing is a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties,*

1. **Bilinearity:** $\forall a, b \in \mathbb{Z}_q^*, \forall P \in \mathbb{G}_1, Q \in \mathbb{G}_2 : \hat{e}(P^a, Q^b) = \hat{e}(P, Q)^{ab}$;
2. **Non-Degeneracy:** $\hat{e} \neq 1$, that is, the mapping is not the identity map;
3. **Computability:** An efficient algorithm exists to determine the output of map $\hat{e}$.

Definition 5 can be classified into three types, in line with [58]:

- **I** : If $\mathbb{G}_1 = \mathbb{G}_2$. This is known as a *symmetric* bilinear map.
- **II** : If $\mathbb{G}_1 \neq \mathbb{G}_2$ and there exists an efficiently computable *homomorphism* $\phi : \mathbb{G}_2 \to \mathbb{G}_1$.
- **III** : If $\mathbb{G}_1 \neq \mathbb{G}_2$ and there does not exist an efficiently computable homomorphism like ϕ.

Next we present a variation of the definition for the CDH problem (Definition 3), following the work of [118], which is in the context of pairings. Namely, the computational co-Diffie-Hellman problem.

Definition 6 (co-CDH Problem)**.** *Let cyclic groups* $(\mathbb{G}_1, \mathbb{G}_2)$ *be defined in Definition 5. Given* $(P, P^a) \in \mathbb{G}_1$ *for random element* a *and* $Q \in \mathbb{G}_2$*, computing* Q^a *is assumed to be computationally infeasible in polynomial time.*

The External Diffie-Hellman Problem Now we introduce the XDH assumption, a computational hardness assumption used in elliptic curve cryptography, to define the symmetric XDH assumption (SXDH) which is required in Chapter 3. The XDH assumption holds if the two distinct groups $(\mathbb{G}_1, \mathbb{G}_2)$ exist with the following properties:

- The discrete logarithm (DLP) problem given in Definition 2, the computational Diffie-Hellman (CDH) problem given in Definition 3, and the computational co-Diffie-Hellman (co-CDH) problem given in Definition 6 are intractable in groups $(\mathbb{G}_1, \mathbb{G}_2)$.
- There exists an efficiently computable bilinear map $\hat{e}$ (Definition 5).
- The decisional Diffie-Hellman problem (DDH) given Definition 4 is intractable in $\mathbb{G}_1$.

The *symmetric* XDH (SXDH) holds if the DDH assumption is also intractable in $\mathbb{G}_2$.

Definition 7 (SXDH Assumption)**.** *Given $\hat{e}$ is defined as in Definition 5 over groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, the SXDH-problem is intractable in polynomial time if groups $(\mathbb{G}_1, \mathbb{G}_2)$ are both DDH-hard groups.*

This means that given $(P_0, P_1, P_2, P_3) \in \mathbb{G}_1^4$ it is *infeasible* to determine if there exists a value x such that $P_1 = P_0^x$ and $P_3 = P_2^x$ simultaneously (similarly for group $\mathbb{G}_2$).

Next we introduce the p-Bilinear Diffie Hellman *Inversion* (p-BDHI) problem, which is used to prove the security of our construction in Chapter 4. The p-BDHI problem is stated as follows:

Definition 8 (p-BDHI Problem)**.** *Given map $\hat{e}$ defined as in Definition 5 over groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ and given $\{P, P^{\alpha}, P^{\alpha^2}, \ldots, P^{\alpha^p}\} \in \mathbb{G}_1^{p+1}$, the p-BDHI problem is considered hard if it is computationally intractable to compute $\hat{e}(P, P)^{1/\alpha} \in \mathbb{G}_T$ in polynomial time.*

The RSA Assumption In the ensuing, we define a strong RSA modulus before stating the sequential squaring assumption needed to prove the security of our construction in Chapter 6. Concretely, our construction utilises a time-delay mechanism introduced in [108], formally called time-lock puzzles (TLPs). At a high level, the TLPs we use in Chapter 6 are computational puzzles based on sequential squaring and the intractability of sequential squaring is essential to assume a TLP takes a threshold amount of time to solve.

First we need to define an RSA integer. Let $N = p \cdot q$ be a *composite* for primes (p, q), such that N is a strong RSA integer.[1] Let $\mathbb{Z}_N^* = \{x \in \mathbb{Z}_N | gcd(x, N) = 1\}$ define the finite group of integers modulo N closed under the multiplication operation $(\otimes)$. For prime p, the *Jacobian* subgroup $\mathbb{J}_p \subseteq \mathbb{Z}_p^*$ is defined as; $\mathbb{J}_p = \{x \in \mathbb{Z}_p^* : \exists y \in \mathbb{Z}_p^* \text{ s.t } y^2 = x \pmod{p}\}$. The same holds for prime q. Thus, the Jacobian subgroup $\mathbb{J}_N \subseteq \mathbb{Z}_N^*$ is formed of elements x such that $\mathbb{J}_N(x) = \mathbb{J}_p(x) \cdot \mathbb{J}_q(x)$.

Definition 9 (Strong RSA Modulus Assumption)**.** *Let λ be the security parameter and N be the product of two λ-bit distinct strong primes p, q, where $p = 2p' + 1$ and*

[1] N is a strong RSA integer if primes $p = 2p' + 1$ and $q = 2q' + 1$, with p', q' also primes.

$q = 2q' + 1$. *Let* e *be a randomly chosen prime such that* $2^\lambda < e < 2^{\lambda+1} - 1$. *Let* $\mathbb{QR}_N$ *be the group of quadratic residues in* $\mathbb{Z}_N^*$ *of order* $p' \cdot q'$. *Given* (N, e) *and a random* $h \in \mathbb{QR}_N$, *it is hard to compute* x *such that* $x^e \equiv h \pmod N$.

Definition 10 (Sequential Squaring Assumption)**.** *Let* N *be a uniform strong RSA integer,* g *a generator of* $\mathbb{J}_N$, *and* $\mathcal{T}(\cdot)$ *be a polynomial. Then there exists some* $0 < \epsilon < 1$ *such that for every polynomial-size adversary* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ *who's depth is bounded from above by* $\mathcal{T}^\epsilon(1^\lambda)$, *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr\left[b \leftarrow \mathcal{A}(N, g, \mathcal{T}(1^\lambda), x, y) : \begin{array}{r} x \xleftarrow{\$} \mathbb{J}_N; b \xleftarrow{\$} \{0,1\} \\ \text{if } b = 0 \text{ then } y \xleftarrow{\$} \mathbb{J}_N \\ \text{if } b = 1 \text{ then } y := x^{2^{\mathcal{T}(1^\lambda)}} \end{array}\right] \leq \tfrac{1}{2} + \mathsf{negl}(1^\lambda).$$

2.4 Fundamental Building Blocks

In this Section, we present the definitions of building blocks essential to cryptographic schemes, and we formalise relevant security notions for each primitive.

Public-Key Encryption The public-key encryption primitive (PKE) is fundamental to many modern cryptosystems [77], applications of which provably satisfy desirable security properties such as confidentiality, integrity, authentication and so on. This primitive is utilised throughout this Thesis and in this Section, we present a standard definition of a PKE scheme and formalise several standard and relevant security notions used in this body of work.

Definition 11 (PKE)**.** *Given the security parameter* λ, *a public-key encryption scheme consists of a tuple of four PPT algorithms* $\Pi_{\mathsf{PKE}} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *which run as follows,*

- $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$: given the input of the security parameter λ, the probabilistic setup algorithm outputs public parameters pp.
- $\mathsf{KG}(pp) \xrightarrow{\$} (pk, sk)$: given the input public parameters pp, the probabilistic key generation algorithm returns a public and private key pair (pk, sk).
- $\mathsf{Enc}(pp, pk, m) \xrightarrow{\$} C$: given input the public parameters pp, public key pk, and the message m, the probabilistic encryption algorithm outputs a ciphertext C.

- $\mathsf{Dec}(pp, sk, C) \to \{m, \perp\}$: given input the public parameters pp, secret key, sk and ciphertext C, the deterministic decryption output outputs the message m on correct execution of the scheme or outputs $\perp$ to indicate an invalid ciphertext.

Correctness Given security parameter 1^λ, Definition 11 is correct if, for any valid message m in message space $\mathcal{MSP}$, there exists a negligible function negl such that the following holds with overwhelming probability.

$$\Pr\left[\begin{array}{l} pp \xleftarrow{\$} \mathsf{Setup}(1^\lambda); (pk, sk) \xleftarrow{\$} \mathsf{KG}(pp); C \xleftarrow{\$} \mathsf{Enc}(pp, pk, m) : \\ \mathsf{Dec}(pp, sk, C) = m \end{array}\right] \geq 1 - \mathsf{negl}(1^\lambda).$$

IND-CPA Security Security against *Chosen Plaintext Attacks* captures adversary $\mathcal{A}$ with an inability to distinguish between the encryption of two chosen messages.

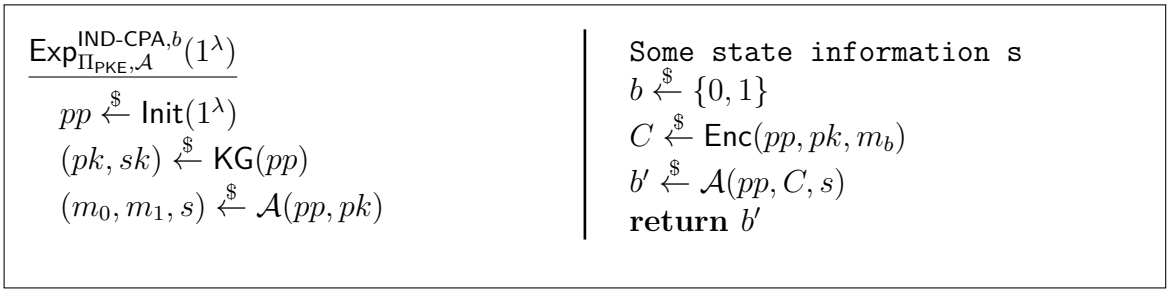

Fig. 2.1 The security experiment for IND-CPA-security of a PKE scheme.

Definition 12 (IND-CPA Security)**.** *A PKE scheme Π_{PKE} is IND-CPA secure if for every efficient PPT adversary $\mathcal{A}$, the advantage the adversary has in the security experiments detailed in Figure 2.1 is negligible as a function of the security parameter 1^λ.*

$$\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CPA},0}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CPA},1}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(1^\lambda).$$

IND-CCA Security In Chapter 5 we model security against *chosen ciphertext attacks*. This indistinguishability notion captures adversary $\mathcal{A}$ with an inability to distinguish between the encryption of two chosen messages, despite oracle access to the decryption of *arbitrary* ciphertexts not equal to the challenge ciphertext.

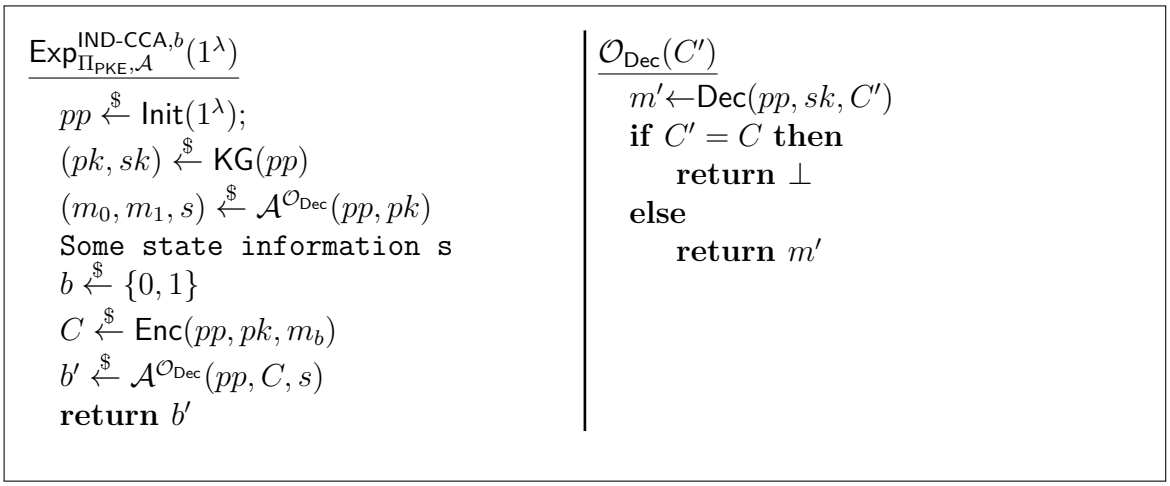

Fig. 2.2 The security experiment for IND-CCA-security of a PKE scheme.

Definition 13 (IND-CCA Security)**.** *A PKE scheme* Π_{PKE} *is IND-CCA secure if for every efficient PPT adversary* $\mathcal{A}$*, the advantage the adversary has in the security experiments detailed in Figure 2.2 is negligible as a function of the security parameter* 1^λ*.*

$$\mathrm{Adv}^{\mathsf{IND\text{-}CCA}}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CCA},0}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CCA},1}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(1^\lambda).$$

IND-RCCA Security In Chapters 3 and 4, we model *replayable* chosen ciphertext (RCCA) security. The authors of [13] prompted a rich area of research into the *tightness* of the security of encryption schemes, and in turn, the level of trust placed on the underlying cryptographic assumptions upon which the security of the scheme relies. However, it remained unclear the level of tangible security a PKE scheme could produce in practice.

This question led to compelling research such as the work of [28] who introduced the notion of replayable CCA security. In essence, the authors highlighted that traditional CCA security is not essential for implementing secure channels - RCCA security suffices and may even permit more *efficient instantiations*.

Effectively, RCCA-security follows the same security modelling as CCA-security with a relaxation of the decryption oracle, intuitively providing an adversary with *inferable* information about a queried ciphertext depending on the oracle's response. That is, an adversary can call the decryption oracle on *arbitrary* ciphertexts, however, the

oracle will respond with test to queries that decrypt to either of the challenge messages (m_0, m_1). This includes queried ciphertexts that differ from the challenge ciphertext C which the adversary obtains from the challenger.

We now present a security game derived from [28] which is used to model this IND-RCCA-security notion for a PKE scheme (Definition 11).

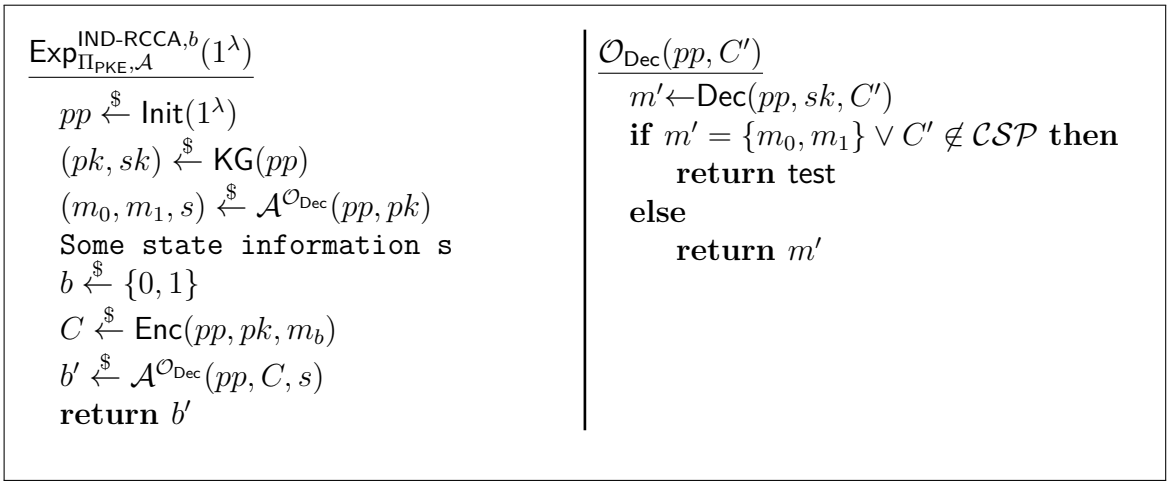

Fig. 2.3 The security experiment for IND-RCCA-security of a PKE scheme.

Definition 14 (IND-RCCA Security)**.** *A PKE scheme Π_{PKE} is IND-RCCA secure if for every efficient PPT adversary $\mathcal{A}$, the advantage the adversary has in the security experiments detailed in Figure 2.3 is negligible as a function of the security parameter 1^λ.*

$$\mathrm{Adv}^{\mathsf{IND\text{-}RCCA}}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}RCCA},0}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{IND\text{-}RCCA},1}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(1^\lambda).$$

Key Privacy In Chapter 3, our analysis of security requires an extended notion of RCCA security that additionally captures the indistinguishability of public keys. This property is formally known as *key privacy* and was introduced by [12]. A PKE scheme secure against key-private, replayable CCA-attacks is modelled in the security experiment for IND-IK-RCCA-security, presented in Figure 2.4, following the design in [12].

Intuitively, modelling key-privacy corresponds to an indistinguishability experiment run by a challenger whereby a polynomial-time adversary with access to a decryption

oracle is given public parameters, including two public keys (pk_0, pk_1), generated by the challenger in the initialisation process. The adversary outputs two chosen messages $(m_0, m_1) \in \mathcal{MSP}$ and the challenger proceeds to encrypt message m_b using public key pk_b for chosen bit $b \in \{0, 1\}$. Security is captured by the adversary's success in the game which corresponds to distinguishing the message and public key used in encryption to produce the challenge ciphertext without knowledge of the corresponding secret keys. More formally,

$\underline{\mathsf{Exp}^{\mathsf{IND\text{-}IK\text{-}RCCA},b}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda)}$
$pp \xleftarrow{\$} \mathsf{Init}(1^\lambda);$
$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KG}(pp);$ `for` $i := \{0, 1\}$
$(m_0, m_1, s) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Dec}}}(pp, pk_0, pk_1)$
if $|m_0| \neq |m_1| \vee \{m_0, m_1\} \notin \mathcal{MSP} \vee (m_0 = m_1)$ **then**
  return $\perp$
else
  $b \xleftarrow{\$} \{0, 1\}$
  $C \xleftarrow{\$} \mathsf{Enc}(pp, pk_b, m_b)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Dec}}}(pp, C, s)$
return b'

$\underline{\mathcal{O}_{\mathsf{Dec}}(C')}$
$m' \leftarrow \mathsf{Dec}(pp, sk_b, C')$
if $m' = \{m_0, m_1\}$ **then**
  return test
else
  return m'

Fig. 2.4 Indistinguishability experiment for security of PKE scheme Π_{PKE} with s as some state information in this figure.

Definition 15 (IND-IK-RCCA Security)**.** *A PKE scheme Π_{PKE} is IND-IK-RCCA secure if for every efficient PPT adversary $\mathcal{A}$, the advantage the adversary has in the security experiments detailed in Figure 2.4 is negligible as a function of the security parameter λ.*

$$\mathrm{Adv}^{\mathsf{IND\text{-}IK\text{-}RCCA}}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}IK\text{-}RCCA},0}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{IND\text{-}IK\text{-}RCCA},1}_{\Pi_{\mathsf{PKE}},\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(1^\lambda).$$

Digital Signature Schemes We use digital signatures as a building block for a concrete instantiation in Chapter 3 to ensure a notion of *integrity* for updatable encryption in a public-key encryption setting. In the following, we formalise the primitive and define correctness and security.

Definition 16 (Digital Signature Scheme)**.** *Given a security parameter λ, a digital signature scheme Π_{Sig} consists of a tuple of four algorithms $\Pi_{SIG} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Sign}, \mathsf{Vrfy})$ such that,*

- $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$: take as input security parameter 1^λ and returns public parameters pp.
- $\mathsf{KG}(pp) \xrightarrow{\$} (pk, sk)$: returns the public and secret key pair comprised of the verification key (pk) and signing key (sk) respectively.
- $\mathsf{Sign}(pp, sk, m) \xrightarrow{\$} \sigma$: returns a signature σ of valid message $m \in \mathcal{MSP}$ with respect to the signing key sk.
- $\mathsf{Vrfy}(pp, pk, m, \sigma) \rightarrow \{\mathsf{accept}, \perp\}$: takes as input the verification key pk and signature σ on message m, returning either accept or rejection $(\perp)$.

Correctness Given security parameter 1^λ, Definition 16 is correct if, for any valid message m in message space $\mathcal{MSP}$, there exists a negligible function negl such that the following holds with overwhelming probability.

$$\Pr\begin{bmatrix} pp \xleftarrow{\$} \mathsf{Setup}(1^\lambda); (pk, sk) \xleftarrow{\$} \mathsf{KG}(pp); \sigma \xleftarrow{\$} \mathsf{Sign}(pp, sk, m) : \\ \mathsf{Vrfy}(pp, pk, m, \sigma) = \mathsf{accept} \end{bmatrix} \geq 1 - \mathsf{negl}(1^\lambda).$$

Security is based on the fact that an adversary should be unable to output a forgery even if it obtains signatures on multiple other chosen messages. This is done by running an experiment where the adversary queries an oracle on chosen messages to obtain signatures, and succeeds in forgery over a message that they have not queried to the oracle.

In more detail, *existentially unforgeable under an adaptive chosen message* attack (EUF-CMA) security is captured in Figure 2.5. This security notion defines the security of a digital signature scheme Π_{Sig} under the leakage of the verification key. That is, an adversary can query a signing oracle $\mathcal{O}_{\mathsf{Sign}}$ on some message $m' \in \mathcal{MSP}$, receiving a valid signature $\sigma' \xleftarrow{\$} \mathsf{Sign}(pp, sk, m')$ output from the oracle. Note, list $\mathcal{M}^*$ is a list maintained by the challenger to tally the messages used for queries in the signing oracle. The adversary proceeds to output a guess signature σ and the challenge returns the message if the signature is valid.

We adapt the notation of [80] in the formal definition of EUF-CMA security as follows,

Definition 17 (EUF-CMA Security)**.** *A digital signature scheme Π_{Sig}, following Definition 16, is EUF-CMA secure if a PPT adversary $\mathcal{A}$ has a negligible advantage in the security experiment defined in Figure 2.5.*

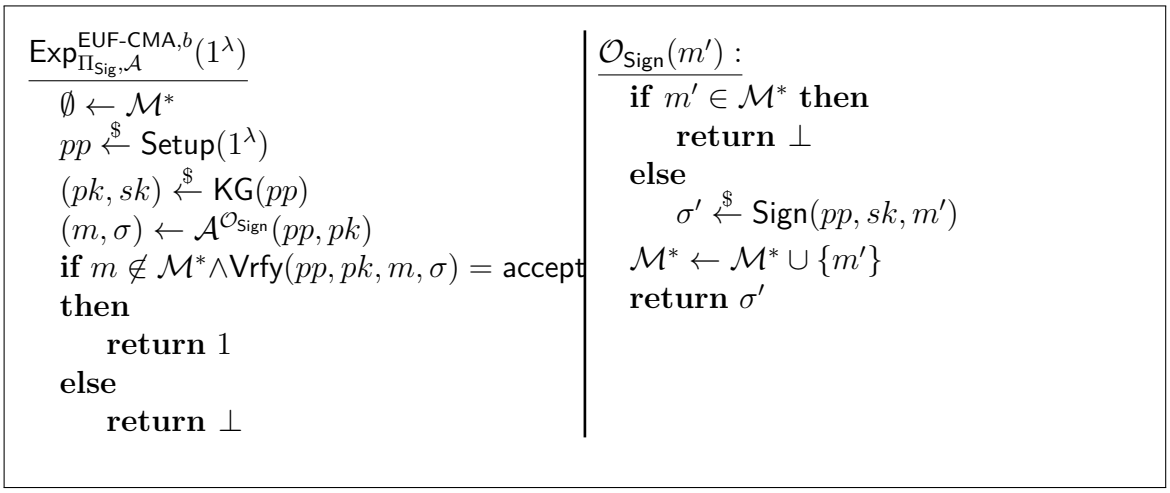

Fig. 2.5 The security game modelling EUF-CMA security for a digital signature scheme Π_{Sig}.

$$\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\Pi_{\mathsf{Sig}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{EUF\text{-}CMA},0}_{\Pi_{\mathsf{Sig}},\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{EUF\text{-}CMA},1}_{\Pi_{\mathsf{Sig}},\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(1^\lambda).$$

In Chapter 3 we make use of a structure-preserving digital signature scheme. Formally, over bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, e)$, following Definition 5, an EUF-CMA secure digital signature scheme satisfies the following definition [107].

Definition 18 (Structure Preserving Digital Signature Scheme)**.** *A digital signature scheme is* structure-preserving *if messages, the verification key and the signature are group elements in $\mathbb{G}_1, \mathbb{G}_2$ and if the verification is a pairing product equation.*

Threshold Secret Sharing We explore extensions and applications of secret sharing schemes in Chapters 5 and 6. Informally, secret sharing is a protocol for *distributing* the *storage* of highly sensitive information. In this Section, we recall the standard definition of a threshold secret sharing scheme as well as an extension, defining important properties for each, and presenting a widely used concrete scheme.

Definition 19 ((t, n) Secret Sharing Protocol)**.** *Given a dealer $\mathcal{D}$, a secret $s \in \mathcal{S}_\lambda$ for security parameter λ, and a set of n authorised players $P = \{P_1, \ldots, P_n\}$, a (t, n) secret sharing scheme (Π_{SS}) is a tuple of three PPT algorithms $\Pi_{SS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Recon})$ defined as follows:*

- **Share Phase**: $\mathcal{D}$ takes as input the secret s and performs the following steps non-interactively:

1. $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$: a probabilistic algorithm that takes as input security parameter 1^λ and outputs public parameters pp, which are broadcast to all players in P.
2. $\mathsf{Share}(pp, s) \xrightarrow{\$} \{s_1, \ldots, s_n\}$: a probabilistic algorithm that takes as input the secret $s \in \mathcal{S}_\lambda$ and outputs n shares s_i, one for each player in P.
3. Distribute s_i to player P_i for every $i \in [n]$ over a secret, authenticated channel.

- **Reconstruction Phase**: Any player in $P = \{P_1, \ldots, P_n\}$ can take part in this phase.

1. Communication:
   (a) Each player P_i sends their share s_i over a secure broadcast channel to all other players in P.
   (b) P_i checks that they have received $(t-1)$ or more shares. If so, they proceed to processing.[2]
2. Processing:
   Once P_i has a set of t' shares labelled S', they independently do the following:
   (a) $\mathsf{Recon}(pp, S') \to \{s, \perp\}$: a deterministic algorithm that takes as input the set S' of t' shares and outputs the secret s if $t' \geq t$ or outputs abort $\perp$ otherwise.

A (t, n) threshold SS scheme needs to satisfy the properties of correctness and secrecy. Informally, correctness means that an honest execution of the scheme results in the true secret being output, except with negligible probability; and secrecy ensures that reconstruction with fewer shares than the threshold (t) results in abort $(\perp)$ being output, except with negligible probability. Formally, Definition 19 needs to satisfy the properties of correctness and secrecy as follows,

Definition 20 (SS Correctness)**.** *A (t, n) secret sharing scheme Π_{SS} is* correct *if $\forall \lambda \in \mathbb{N}$ and for all possible sets of n authorised players $P = \{P_1, \ldots, P_n\}$, given $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$; for all secrets $s \in \mathcal{S}_\lambda$ and any subset of $t' \geq t$ shares S' from $\mathsf{Share}(pp, s) \xrightarrow{\$} \{s_1, \ldots, s_n\}$ communicated by players in P, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

[2]Whilst not explicit in the definition, there is an upper bound on how long players can communicate their shares. Therefore, at the end of their communication, if a player P_i has not obtained a sufficient number of shares, then they output $\perp$ at the end of the reconstruction phase.

$$Pr[\mathsf{Recon}(pp, S') \neq s] \leq \mathsf{negl}(1^\lambda).$$

Definition 21 (SS Secrecy)**.** *A (t, n) secret sharing scheme Π_{SS} is* secret *if $\forall \lambda \in \mathbb{N}$ and for all possible sets of n authorised players $P = \{P_1, \ldots, P_n\}$, given $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$; for all secrets $s \in \mathcal{S}_\lambda$ and any subset of $t' < t$ shares S' from $\mathsf{Share}(pp, s) \xrightarrow{\$} \{s_1, \ldots, s_n\}$ communicated by players in P, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$Pr[\mathsf{Recon}(pp, S') \neq \perp] \leq \mathsf{negl}(1^\lambda).$$

Shamir's Secret Sharing Shamir's threshold secret sharing scheme [110] is a concrete SS scheme, following Definition 19, based on polynomial interpolation. We will informally present this method of secret sharing in the following:

Let $\mathrm{GF}(q)$ be a finite field of prime order $q \geq n$ such that the secret $s \in \mathrm{GF}(q)$. The dealer $\mathcal{D}$ randomly chooses a polynomial $f(x)$ of degree $(t-1)$ such that $f(0) = s$. That is, $f(x) = s + \sum_{i=1}^{t-1} a_i x^i$. Furthermore, $\forall i \in n$, the dealer randomly chooses a unique $\alpha_i \xleftarrow{\$} \mathrm{GF}(q)$ and secretly distributes shares $s_i = f(\alpha_i)$ to the corresponding player in P.

Given subset $P' \subset P$ with $|P'| \geq t$, the players in P' can reconstruct s using polynomial interpolation as follows:

$$f(\alpha_i) = \textstyle\sum_{P_j \in P'} \lambda_{j,i}^{P'} f(\alpha_j) = \sum_{P_j \in P'} \lambda_{j,i}^{P'} s_j,$$

such that the Lagrangian Coefficient $\lambda_{j,i}^{P'} = \Pi_{P_l \in P', l \neq j} \frac{\alpha_i - \alpha_l}{\alpha_j - \alpha_l}$. Note that any subset $P' \subset P$ of players whereby $|P'| < t$ cannot obtain any information about the secret s from the polynomial $f(x)$.

Equipped with these tools, we are now ready to present our results.

Chapter 3

Epoch Confidentiality in Public-Key Updatable Encryption

Contents

This Chapter introduces a new updatable encryption primitive defined in the public-key setting and a novel security notion for updatable encryption capturing the leakage of the age of the ciphertext. We present a security model and propose a construction provably satisfying this notion. Part of this work appears in [81] which is joint work with Elizabeth A. Quaglia, published and presented at ProvSec 2022.

3.1 Introduction

3.1.1 Motivation

In recent years there has been an increased need to outsource the storage of encrypted data to a potentially untrusted host. To protect the privacy of the underlying data and mitigate the security risks of key compromise over a long time, several cryptographic schemes have proposed employing the technique of *key-rotation* which is a mechanism to move existing ciphertexts from encryption under an old to a new cryptographic key.

Trivially, a scheme can satisfy this idea of key rotation by decrypting the ciphertext and then re-encrypting the result with the updated key. However, when the encrypted data has been outsourced to an external (un-)trusted entity, this is an impractical method. There are two options for a data owner if key rotation is performed in the trivial sense. Either the owner downloads, re-encrypts and updates all ciphertexts themself, which is computationally inefficient. Alternatively, the owner could outsource the update by sending the encryption keys to the untrusted host to perform re-encryption, which no longer ensures security.

The authors of [24] introduced the *updatable encryption* (UE) primitive to provide a more elegant, non-trivial solution to the above discussion. Instead of re-encrypting a ciphertext from an old to a new key, the data owner instead generates a *token* that enables the host to convert the ciphertext to encryption under the new key (provided it is trusted to delete old tokens and ciphertexts after an update) *without* the need to decrypt.

Traditionally, the UE primitive is designed in the symmetric-key setting [24, 52, 25, 32, 104, 75] to convert ciphertexts in a periodic manner marked by set time-intervals known as *epochs* and using encryption keys valid only for their associated epoch. We provide a lengthy discussion on symmetric UE literature in the section proceeding the introduction. However, we note here that it is essential to security that the potentially untrusted server should not learn anything about the underlying message of the ciphertext that they are updating, or surplus information arising from the ciphertexts that they are storing.

To date, the focus of UE research has been dominated by the development of rigorous security models to encapsulate notions of ciphertext confidentiality and integrity. As a

consequence, numerous frameworks for security have been proposed and the relationship between security notions has become well-understood [25]. Notwithstanding the importance of strengthening security for the UE primitive, we think it is of equal interest to consider extending the UE primitive to alternative settings and applications.

Precisely, we are interested to explore possible extensions of UE schemes and security modelling in the *public-key* setting. To start, it is considered good key management practice to rotate cryptographic keys when storing encrypted data over a long time. So-called data-at-rest is not limited to the symmetric setting alone, therefore, the update functionality and strong security guarantees from UE literature can be utilised in public-key encryption schemes when applying PKUE as a building block. Thus, formalising a PKUE primitive will be useful in real-world applications requiring public-key encryption, especially in scenarios whereby the data owner does not trust the host storing data, or at the very least the owner seeks to *reduce trust* in the host.

In particular, broadening the scope of UE enables us in this Thesis to design certificateless UE schemes, which is a cryptographic scheme used to mitigate the *key escrow* problem in identity-based encryption [3]. Further, in formalising PKUE we are able to consider identity-based, multi-server updatable encryption schemes. The latter extension is particularly useful for applications such as storing secrets on a public blockchain [15]. These extensions will be explored in Chapter 4 and Chapter 5 respectively.

Another compelling argument for lifting UE to the public-key setting is so we can employ public-key techniques in security modelling. In doing so, we can capture new and interesting security notions that were previously difficult to capture in the traditional, symmetric UE setting. Conventionally, an adversary attacking a UE scheme has greater corruption capabilities in comparison to 'static' encryption, due to the interconnected nature of cryptographic elements such as epoch keys and update tokens. Therefore, finding novel techniques is important since there are unique challenges in UE security modelling arising from the update feature.

In particular, it is important to hide the number of times a ciphertext has been updated in a UE scheme, especially one with authenticated encryption [75], which in practice is the majority of UE schemes. We take the approach of [12] to model a form of *key privacy* for the public keys in PKUE, which is an overlooked technique in the traditional UE security framework. Key privacy essentially ensures no two public epoch keys are

distinguishable in the eyes of an adversary. This property is usefully applied in our PKUE security model to achieve the new notion of confidentiality of an epoch, which in turn implies the confidentiality of the age of a ciphertext.

In the wider sense, the complexities in ensuring security could dominate important factors such as the efficiency, practicality and cost of implementation. We believe it is important to consider all factors to a varying degree, depending on the application to which PKUE is applied. With careful deliberation, we chose to place more emphasis on modelling strong security notions for our PKUE primitive due to the aforementioned corruption capabilities of an adversary. Otherwise, the usefulness of the update feature becomes redundant and one may as well use a standard public-key encryption scheme.

As a consequence, we chose to design a public key UE primitive with *probabilistic updates* to attain desirable levels of privacy. In practice, re-randomising ciphertexts result in a more expensive scheme and the server is burdened with producing 'good' randomness. However, this property allows us to model an adversary with stronger corruption powers than in deterministic schemes. Further, we define PKUE to be a *ciphertext-independent* update primitive since a clear goal in UE is to limit the bandwidth required for the implementation of a scheme. To illustrate, the alternative ciphertext-dependent setting requires individual tokens to be generated per ciphertext and ciphertexts need to be stored for longer periods. We defer to Section 3.2 for an in-depth discussion of the probabilistic, ciphertext-independent setting of UE.

3.1.2 Our Contributions

To begin, we formally define our public-key updatable encryption primitive (PKUE) and formalise the correctness property of PKUE.

Next we comprehensively detail a public-key UE security model to capture two essential UE security notions: post-compromise security and ciphertext unlinkability. Both are necessary because an adversary can *infer* information independently due to the interwoven relationship between cryptographic elements. Informally, post-compromise security is a property necessary in updatable schemes to prevent an adversary from gaining an advantage in decrypting ciphertexts generated in a later epoch than one in which a secret key has been compromised. Post-compromise security must be attained under the assumption that old epoch keys and tokens have not been deleted. Else, the security of a UE scheme is reduced to the security of a non-updatable encryption scheme

since old epoch keys are no longer useless and tokens incorporate old keys as well as new ones [92]. The latter property of ciphertext unlinkability is an indistinguishability notion (IND-UPD [92]) desirable in the UE framework since an adversary can corrupt two types of ciphertext, namely, updated ciphertexts and newly encrypted ciphertexts. If ciphertexts are linkable then an adversary is capable of tracing an updated ciphertext back to the original version and possibly inferring information about update tokens or epoch keys in the process.

More formally, we encapsulate post-compromise security and ciphertext unlinkability in a replayable chosen-ciphertext security game (UP-IND-RCCA) in which an adversary queries updates of *arbitrary* ciphertexts. We defer the reader to Section 2.4 and Section 3.2 for a discussion about the replay-ability of ciphertexts in security experiments but we note here for clarity that the probabilistic nature of ciphertext updates in the design of our PKUE primitive prevents the strictly stronger notion of security against chosen ciphertext attacks from being achieved.

Following the above framework for standard security notions, we explore a previously overlooked issue related to the leakage of ciphertext age which is important considering applications of UE in which leakage of this information could violate the privacy of an individual. Concretely, we introduce a new notion of security called *epoch confidentiality* tackling the above problem which we achieve by using public-key techniques. In particular, we apply key privacy [12] methods (see Section 2.4) in our security model to satisfy epoch confidentiality for our PKUE primitive. We believe key privacy is especially important in PKUE schemes for the same reason that post-compromise security is important. Namely, epoch keys have more functions than in standard PKE schemes since epoch keys are required in update token generation and they directly relate to the corresponding epoch in which they are used.

Lastly, we present a concrete public-key UE scheme which is an adaptation of an existing symmetric UE construction [80] explicitly using *updatable* public-key building blocks. We conclude by considering the efficiency of the scheme and demonstrating it is provably secure for the epoch confidentiality notion.

3.1.3 Existing Work

The authors of [49] established definitions for updatable *public-key* encryption (UPKE) using an alternative update procedure. One can view the UPKE primitive [49, 45] as

dual to the UE primitive in the sense of key-policy versus ciphertext-policy attribute-based encryption respectively. Specifically, the authors of [49] look at the separate primitive of key-updatable encryption scheme which is for message layer security (MLS) like environments where a key pair can be partially updated by any party in the group to provide post-compromise security. The token mechanism used in a UPKE scheme updates the public key, rather than the ciphertext. As a consequence, the designs and security modelling for the UPKE scheme [49] differ from the UE literature [52, 24, 92, 80]. Therefore, our formal definition of public-key UE presented below can be considered distinct from the definition of [49].

Closely related to our definition of public-key updatable encryption is the work on *Self-Updatable Encryption* (SUE) schemes in [91, 90]. An SUE scheme is a public-key encryption primitive enabling ciphertext updates which are used to provide timed access control in applications with cloud storage. The basic idea [90] is to associate independent time parameters to ciphertexts and keys, such that public ciphertext updates (to a future time) can be performed using only an individual's public key and decryption of ciphertexts is possible with a user's secret key from some past time. Additionally, neither decryption nor updates of ciphertexts should be possible after a user has their access revoked.

The authors of [91, 90] defined an extension of the SUE primitive enabling *time-specific* [105] encryption (TI-SUE). Essentially, the TI-SUE primitive limits the decryption and updates of ciphertexts to distinct windows of time. Despite similarities between the (TI-)SUE and UE primitives, there are rudimentary distinctions between the two. Categorically, the UE primitive does not enable *public ciphertext updates*, a basic feature of SUE. Conversely, the update mechanism in a UE scheme requires an element known as a token and the process is outsourced to a *specific* server for storage. Moreover, SUE traditionally updates only a *fraction* of the ciphertext and so notions featured in UE security modelling, such as ciphertext unlinkability, cannot be satisfied.

Recently the authors of [120] proposed a re-randomisable encryption scheme in the identity-based setting, to achieve the unlinkability of identities in an IBE scheme. Despite UE schemes currently being defined for individuals, the parallels of [120] to our work exemplify the usefulness of defining public-key UE schemes.

One may view our PKUE primitive as an alternative version of a *multi-hop* PRE (MH-PRE) primitive [38], in the sense of key updates and the security level attained.

In particular, MH-PRE allows delegation of decryption by updating ciphertexts from encryption under the key of one recipient to the next recipient (which can be repeated many times). Conversely, PKUE is designed to be used in public-key primitives in which one user outsources the update of their secret information by rotating keys from one *epoch* to the next over distinct periods. In general, PRE schemes do not necessarily require the originally re-encrypted ciphertext to be of the same form as the originally generated ciphertext. In this case, the notion of ciphertext unlinkability that is fundamental to a UE scheme is not satisfied in PRE. Ultimately, the distinction between any extended UE and PRE scheme can be seen in the fundamental differences of the underlying primitives. We highlight the similarities and differences between UE and PRE primitives in Section 3.2, Table 3.1.

The authors of [6] introduce a notion of anonymous re-encryption keys for PRE schemes which are achieved using a similar technique to our technique for modelling epoch confidentiality. Whilst the key-privacy techniques used in [6] are similar to our own, we model in tandem the *indistinguishability* of public keys used in encryption and ciphertext updates *and* ciphertext confidentiality.

Further UE research centring on a notion similar to epoch confidentiality can be seen in [19, 25] who simultaneously observed that prior schemes such as [52] do not prevent the leakage of the age of a ciphertext, be that from fresh encryption or ciphertext updates. In other words, their work focuses on strengthening existing ciphertext confidentiality security notions. Our results, however, are distinguished from this line of work as we detail further in Section 3.5.

3.2 Chapter Preliminaries

In this Section, we begin by presenting the syntax related to traditional updatable encryption. Next, we formally introduce symmetric UE, followed by a definition of correctness. Furthermore, we discuss the state of the art in UE literature, providing a discussion regarding differing standard security notions and assumptions dependent on the type of UE scheme being used. Lastly, we make comparisons between the closely aligned UE and proxy re-encryption primitives. We note that the UE primitive is featured in Chapters 3, 4, and 5.

Syntax We present the syntax used throughout this Chapter and the ensuing Chapters related to updatable encryption. This primitive is defined by epochs of time e_i from

the range of time $i = \{0, \ldots, \mathsf{max}\}$. We denote the current epoch e or use the subscript notation e_i for $i \in \mathbb{N}$ if we define multiple epochs at once and in security games the challenge epoch is represented by $\tilde{e}$. To signify epoch keys the notation k_e, k_{e+1} and k^{old}, k^{new} are used interchangeably in this Thesis, depending on whether we require explicit epoch notation or we only need to define consecutive epoch keys (similarly for update tokens Δ).

Introduced by [24], updatable encryption (UE) is an important building block to applications in which sensitive information is outsourced over long periods. Traditionally, UE is a symmetric primitive designed to enable the *periodic rotation* of the encryption key such that a ciphertext is updated to encryption under a new key. The update is achieved using an *update token* generated by the data owner and the update itself is processed by an untrusted, outsourced host. The formal definition of symmetric updatable encryption is as follows.

Definition 22 (UE)**.** *An updatable encryption scheme Π_{UE} for message space $\mathcal{MSP}$ consists of a set of polynomial-time algorithms $\Pi_{\mathsf{UE}} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{TG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Upd})$ which are defined as follows:*

1. $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$: The *data owner* runs the *probabilistic* setup algorithm on input security parameter λ, outputting public parameters pp.

2. $\mathsf{KG}(pp, e) \xrightarrow{\$} (k_{e+1})$: The *data owner* runs the *probabilistic* key generation algorithm on input public parameters pp for the current epoch e. The output is a new secret key k_{e+1} for the next epoch.

3. $\mathsf{TG}(pp, k_e, e{+}1) \rightarrow \Delta_{e+1}$: The *data owner* runs the deterministic token generation algorithm using the current epoch secret key k_e as input and outputs an update token Δ_{e+1} for the next epoch.

4. $\mathsf{Enc}(pp, k_e, m) \xrightarrow{\$} C_e$: The *data owner* runs the *probabilistic* encryption algorithm on input a message $m \in \mathcal{MSP}$ and secret key k_e of some epoch e, outputting a ciphertext C_e.

5. $\mathsf{Dec}(pp, k_e, C_e) \rightarrow \{m, \perp\}$: The *data owner* runs the *deterministic* decryption algorithm on input a ciphertext C_e and secret key k_e for some epoch e, returning either the message m or abort $\perp$.

6. $\mathsf{Upd}(pp, \Delta_{e+1}, C_e) \rightarrow C_{e+1}$: The *host* runs *either* the deterministic *or* probabilistic ciphertext update algorithm. This is run on input ciphertext C_e for epoch e, update token Δ_{e+1} for the *next* epoch $(e+1)$, and returns as output the updated ciphertext C_{e+1}.

Correctness Given security parameter λ, Definition 22 is correct if, for any message $m \in \mathcal{MSP}$ and for any $j \in \{1, \ldots, \mathsf{max}\}$, $i \in \{0, \ldots, \mathsf{max}\}$ with $\mathsf{max} > i$, there exists a negligible function negl such that the following holds with overwhelming probability.

$$\Pr\left[\begin{array}{l} pp \xleftarrow{\$} \mathsf{Setup}(1^\lambda); k_{e_j} \xleftarrow{\$} \mathsf{KG}(pp, e_j); \\ \Delta_{e_j} \leftarrow \mathsf{TG}(pp, k_{e_{j-1}}, e_j); C_{e_i} \xleftarrow{\$} \mathsf{Enc}(pp, k_{e_i}, m); \\ \{C_{e_j} \leftarrow \mathsf{Upd}(pp, \Delta_{e_j}, C_{e_{j-1}}) : j \in \{i+1, \cdots, \mathsf{max}\}\}; \\ \mathsf{Dec}(pp, k_{e_{\mathsf{max}}}, C_{e_{\mathsf{max}}}) = m \end{array}\right] \geq 1 - \mathsf{negl}(1^\lambda).$$

State of the Art in Updatable Encryption There are two distinct types of UE schemes in the literature, both with their own merits. *Ciphertext-dependent* UE schemes [24, 52, 19, 32] require the host to send a header for each ciphertext followed by the data owner creating and sending tokens for each header. As such, communication grows linearly with the number of stored messages [53] and so efficiency and cost are affected. Less obvious is the fact that epoch keys will be retained for a long time to enable the data owner to generate individual tokens, increasing the power of the adversary and potentially reducing security as the adversary can compromise keys for a longer period.

Ciphertext-independent UE schemes [92, 80, 25, 75, 104] require the data owner to generate a *single* update-token. Typically, the server can *sequentially* update ciphertexts using a token derived purely from the old and new epoch keys. A consequence of this is that post-compromise security [92] must be satisfied to capture the security of epochs after the corruption of keys or tokens since one cannot rely on the secure deletion of old keys in this updated setting [92]. The broader functionality of the update token escalates the issues in providing security, however, from an efficiency point-of-view generating a single *compact* token that can update *any* ciphertext to the next epoch is desirable [75, 104]. Moving forwards, we will focus on ciphertext-independent updatable encryption.

Standard security notions for a UE scheme include ciphertext confidentiality and plaintext/ciphertext integrity which are two established security notions that are considered fundamental to the security of a UE scheme. Ciphertext confidentiality [92, 80, 25] models the indistinguishability of ciphertexts that have either been updated or freshly generated, even with an adversary possessing past ciphertexts. Integrity notions model an adversary attempting to produce valid forgeries of a plaintext/ciphertext [80, 25, 19], in which a successful ciphertext forgery informally means an adversary is capable of generating a ciphertext that decrypts to a valid message ($m \neq \perp$).

Current literature shows that attaining such levels of security can be affected by whether the UE scheme is designed for *deterministic* or *probabilistic* ciphertext updates, and there are merits to both designs depending on the application of the scheme. Indeed, prior UE schemes have demonstrated that probabilistic updates are more expensive and one cannot achieve ciphertext-integrity (CTXT) nor CCA security [92, 80, 25]. However, randomised updates enable a meaningful definition of ciphertext unlinkability [53, 112], namely, updated ciphertexts appear indistinguishable from freshly encrypted ciphertexts, even in possession of previous ciphertexts and tokens.

Conversely, without provisions in the security modelling, an adversary in a deterministic scheme is capable of corrupting an update token in an epoch and trivially distinguishing between an update of a known ciphertext and other ciphertexts in the next epoch [25]. As a consequence, some deterministic schemes such as [52] can only allow adversarial ciphertext update queries on honestly generated ciphertexts, preventing such schemes from achieving the ciphertext unlinkability notion unless stronger assumptions on the adversary's corruption capabilities are made (see [80]).

Notably, modelling security against *passive re-encryption* attacks is not appropriate for probabilistic UE schemes which allow re-encryption queries on arbitrary ciphertexts and thus assume an adversary is *active*. The approach taken by [80], who design UE with probabilistic re-encryption of ciphertexts, is to model *replayable*-CCA security of a UE scheme. Recall from Section 2.4 a discussion on the RCCA-security [28] of a standard PKE scheme. The authors of [65] first revealed that RCCA-secure PKE schemes can have re-randomisable ciphertexts such that, given an input ciphertext, the re-randomisation mechanism produces a *fresh* and *unlinkable* ciphertext decrypting to the same message. Therefore, such PKE schemes can be used for applications requiring secure communication guarantees such as confidentiality and anonymity [54].

Consequently, we concur with [80] that modelling an RCCA-secure UE scheme with probabilistic updates is a suitably chosen level of security since ciphertext unlinkability is an important security notion in the context of updatable encryption.

In the context of update direction, prior works including [92, 24, 52, 25] have defined schemes in which updates are *bi-directional*, meaning a scheme supports the downgrading of epoch keys and ciphertexts to a previous epoch. This is less desirable from a security modelling standpoint as an adversary has more indirect information available, alas, the challenges of achieving one-way updates to the future had prevented any uni-directional schemes from being proposed. More recently, however, [75] demonstrated that the direction of ciphertext updates does not matter much and they proved that security notions in uni- versus bi-directional schemes are *equivalent*.

Moreover, [104] introduced a new definition called *backwards-leak uni-directional* key updates and demonstrated that prior bi-directional UE schemes did not satisfy this strictly stronger security notion.[1] Further, [112] showed that there is currently no suitable approach to provide (replayable-) CCA security for uni-directional UE schemes. In addition, the authors of [75, 112] investigated a strictly stronger level of security, compared to bi/uni-directional key update setting, if one designs update tokens to be *no-directional*, meaning that no keys in two successive epoch keys can be derived from the other.

Probabilistic Versus Deterministic Updates Recall from the above discussion on UE literature, the primitive can be categorised depending on whether the primitive is defined for probabilistic or deterministic ciphertext updates. This distinction is necessary for security modelling, as a UE challenger must record more information than standard security modelling. In particular, the challenger needs to track the epochs in which honestly generated or challenge ciphertexts have been updated, however, if ciphertexts are re-randomised upon updates then they are harder to trace.

In more detail, the level of security achieved for probabilistic and deterministic UE differs. Concretely, the strongest notion of ciphertext confidentiality in probabilistic UE schemes is security against *replayable* chosen ciphertext attacks (RCCA) [28], whereas deterministic UE schemes can attain the stronger notion of chosen ciphertext attacks (CCA). See Definition 14 and Definition 13 respectively, from Section 2.4.

[1] Observe that the proposed scheme in [75] is distinguished by the authors of [104] to be *forward*-leak uni-directional.

Recollect, the model of RCCA differs in an adversary's corruption capabilities; they can query updates on *arbitrary ciphertexts* including a version of the challenge ciphertext. Security against CCA attacks cannot be satisfied with probabilistic updates since a challenger cannot track whether a queried ciphertext derives from the challenge ciphertext. This trivial attack occurs if an adversary has corrupted an update token since they can manually re-encrypt ciphertexts. Note that this attack does not apply to the deterministic setting as updates of ciphertexts are not re-randomised.

In UE security modelling, to prevent the decryption of an updated challenge ciphertext, irrespective of whether the UE scheme is probabilistic or deterministic, a useful predicate defined in [80] can be utilised in the running of decryption and update oracles.[2] Informally, the $\mathsf{isChallenge}(k_{e_i}, C)$ predicate detects any queries to the decryption and update oracles on challenge ciphertexts ($\tilde{C}$), or versions (i.e. updated) of the challenge ciphertext. Formally,

Definition 23 ($\mathsf{isChallenge}$ Predicate)**.** *Given challenge epoch $\tilde{e}$ and challenge ciphertext $\tilde{C}$, the $\mathsf{isChallenge}$ predicate, on inputs of the current epoch key k_{e_i} and queried ciphertext C_{e_i}, responds in one of three ways:*

1. If $(e_i = \tilde{e}) \wedge (C_{e_i} = \tilde{C})$, return true;
2. If $(e_i > \tilde{e}) \wedge (\tilde{C} \neq \perp)$, return true if $\tilde{C_{e_i}} = C_{e_i}$ in which $\tilde{C_{e_i}}$ is computed iteratively by running $\mathsf{Upd}(pp, \Delta_{e_{l+1}}, \tilde{C_{e_l}})$ for $e_l = \{\tilde{e}, \ldots, e_i\}$;
3. Otherwise, return false.

Moreover, depending on whether a ciphertext is re-randomised in the update process affects the underlying assumptions placed on the scheme, necessary to ensure security. We state these assumptions in the proceeding content as we will propose concrete public key UE schemes designed for both forms of update in Chapters 3 and 4 of this Thesis.

Probabilistic Update Assumptions: The property of *perfect re-encryption* is assumed to hold for a UE scheme with probabilistic ciphertext updates when following the generic approach to proving security (by reduction) first defined by [80]. We assume this property holds in Chapter 3 to prove the security of a concrete public-key UE scheme with re-randomised ciphertext updates.

[2] A predicate is a statement or mathematical assertion that contains variables. The outcome of the predicate may be true or false depending on the input values.

Definition 24 (Perfect Re-Encryption)**.** *Given a UE scheme where the algorithm* Upd *is probabilistic, the update of a ciphertext is* ***perfect*** *if for all* $pp \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*, for all old and new epoch key pairs* $k_e \xleftarrow{\$} \mathsf{KG}(pp, e)$ *and* $k_{e+1} \xleftarrow{\$} \mathsf{KG}(pp, e+1)$*, for all ciphertexts* $C \in \mathcal{CSP}$ *and for all tokens* $\Delta_{e+1} \leftarrow \mathsf{TG}(pp, k_e, e+1)$*, we have the following:*

$$\mathsf{Enc}(pp, k_{e+1}, \mathsf{Dec}(pp, sk_e, C)) \overset{Dist}{=} \mathsf{Upd}(pp, \Delta_{e+1}, C).$$

Equality $\overset{\text{Dist}}{=}$ denotes the equal distribution of the left and right-hand sides of the equation. For any epoch e, $\mathsf{Enc}(pp, pk_e, \perp) = \perp$ by definition.

Deterministic Update Assumptions: To prove the security of a UE scheme with *deterministic* ciphertext updates, we assume the properties of randomness-preserving re-encryption and tidy encryption in addition to the aforementioned simulatable token generation assumption. We assume this property holds in Chapter 4 to prove the security of a concrete public-key UE scheme with deterministic updates.

Definition 25 (Randomness Preserving Re-Encryption)**.** *Given a UE scheme (*Π_{UE}*) designed with* deterministic *updates, an updated ciphertext is* randomness-preserving *assuming* Π_{UE} *encrypts with uniformly chosen randomness (the outputs of* $\mathsf{Enc}(pp, m, k_e)$ *and* $\mathsf{Enc}(pp, m, k_e; r)$ *for uniformly chosen* r *are identically distributed). If for all* $pp \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*; for all old and new epoch keys* (k_e, k_{e+1}) *generated from running the* KG *in epochs* e *and* $(e+1)$ *respectively; for all valid ciphertexts* $C \in \mathcal{CSP}$ *and for all tokens* $\Delta_{e+1} \leftarrow \mathsf{TG}(pp, k_e, e+1)$*, we then have the following:*

$$\mathsf{Enc}(pp, k_{e+1}, \mathsf{Dec}(pp, k_e, C_e)) = \mathsf{Upd}(pp, C_e, \Delta_{e+1}).$$

Definition 26 (Tidy Encryption)**.** *A UE scheme is* randomness-recoverable *if there is an associated efficient deterministic algorithm* $\mathsf{RDec}(pp, C_e, k_e)$ *for epoch* e *such that* $\forall k_e, m, r : \mathsf{RDec}(pp, k_e, \mathsf{Enc}(pp, k_e, m; r)) = (m, r)$*. A randomness-recoverable UE scheme* tidy *if* $\forall (k_e, C_e) : \mathsf{RDec}(pp, C_e, k_e) = (m, r) \implies \mathsf{Enc}(pp, k_e, m; r) = C_e$*.*

Informally, this means that the public key encryption scheme is tidy if encryption and decryption algorithms are *bijections*(one-to-one correspondence for a fixed key) between the *message-randomness pairs* and *valid ciphertexts.*

Lastly, *simulatable token generation* is an important property, required in this Thesis for proving security, irrespective of the manner in which ciphertexts are updated. That is, we assume that simulatable token generation is feasible to prove the security of concrete constructions in Chapters 3 and 4. Informally, this property means it is possible to simulate perfectly indistinguishable update tokens. More formally,

Definition 27 (Simulatable Token Generation)**.** *The UE scheme defined in Definition 22 has simulatable token generation if the following properties hold:*

1. *There exists a PPT algorithm denoted* $\mathsf{Sim.TG}(pp)$ *which samples a pair of update tokens* (Δ, Δ') *of the token and reverse token respectively.*

2. *For arbitrary (fixed) epoch key* $k^{old} \leftarrow \mathsf{KG}(pp, old)$*, the following token* (Δ) *distributions are the same:*

   - *Distribution induced by running* $(\Delta, \cdot) \xleftarrow{\$} \mathsf{Sim.TG}(pp)$*;*
   - *Distribution induced by running* $(\Delta, \cdot) \xleftarrow{\$} \mathsf{TG}(pp, k^{old}, new))$*, for epoch key* $k^{new} \xleftarrow{\$} \mathsf{KG}(pp, new)$*.*

UE vs. PRE Standards and frameworks for *cloud computing* have garnered much interest in the cryptographic research community. In particular, cloud computing affects the risks to the security and privacy of cryptographic schemes implemented in this setting, thus, new solutions are necessary to maintain security.

Two important solutions to cloud computing are updatable encryption and *proxy re-encryption* (PRE) [17], which are closely aligned cryptographic primitives in terms of motivation and design. The PRE primitive is used for *decryption delegation* in the cloud and it works as follows. At a high level, a delegating party (A) outsources the re-encryption of their ciphertext via a trusted third party (*proxy*) such that the output is a ciphertext encrypted under the cryptographic key of the intended receiving party (B). Consequently, party B can decrypt party A's ciphertext and learn the underlying message. Instantly, one can see the synchronicity between UE and PRE such as the outsourcing of re-encryption, however, UE re-encryption or so-called updates are related to *time keys* not the delegation of decryption.

In Table 3.1 we explicitly compare key features and properties of UE and PRE to highlight that there are clear distinctions between the two primitives. We note that in Chapters 3, 4, and 5 we provide a discussion on extensions of PRE relevant to each Chapter in their respective related work Sections. We highlight that despite some of the similarities, any extended PRE scheme remains distinct from our UE contributions due to the fundamental differences between standard UE and PRE definitions. The contrast between the two primitives is further supported by the works of [39, 92, 80].

UE vs. PRE		
Properties	**Updatable Encryption**	**Proxy Re-Encryption**
Applications	Timely updates from epoch e_i to e_{i+1}.	Ciphertext decryption delegation from user A to user B.
Type of cryptosystem	Symmetric but can be asymmetric.	Asymmetric
The direction of updates	Bi-directional or no-directional.	Uni-directional or bi-directional.
Token generation	Sequential and deterministic.	Probabilistic
Security goals	Full re-randomisation of updated ciphertexts (ciphertext unlinkability).	Full re-randomisation is not necessary.

Table 3.1 Key distinctions between the traditional properties of UE and PRE primitives.

3.3 Public-Key Updatable Encryption

In this Section, we give an intuition for public-key updatable encryption (PKUE), we present a formal definition of a PKUE primitive and define correctness.

3.3.1 Formal Definition of PKUE

Recall from Section 3.2 that a symmetric UE scheme has an owner create a ciphertext (encryption of sensitive information) that will be outsourced over a long time. Time in a UE scheme is formally divided into equal periods known as epochs in which epochs are associated with distinct keys. A ciphertext is updated (re-encrypted) by a potentially untrusted host to the next epoch to provide stronger security by rotating the key used for encryption. Crucially, this update is performed by the host using an update token derived by the data owner, which is formed from the current and preceding epoch keys, such that the host is incapable of learning anything about the encrypted information.

Following the discussion in the Introduction of this Chapter, we are motivated to formalise a public-key UE scheme (PKUE) which will be provided below. The key idea is to lift the definition of UE to the public-key setting by generating an epoch key consisting of a public key and a secret key component, and the update token is derived from the past epoch secret key and the current (full) epoch key.

Definition 28 (PKUE)**.** *A public-key updatable encryption scheme for message space* $\mathcal{MSP}$ *consists of a set of polynomial-time algorithms* $\Pi_{\mathsf{PKUE}} = (\mathsf{UE.Setup}, \mathsf{UE.KG}, \mathsf{UE.TG}, \mathsf{UE.Enc}, \mathsf{UE.Dec}, \mathsf{UE.Upd})$*, defined as follows:*

- $\mathsf{UE.Setup}(1^\lambda) \xrightarrow{\$} pp$: The *owner* runs the *probabilistic* setup algorithm on input security parameter λ, outputting public parameters pp. Whilst not made explicit, assume throughout that the security parameter (λ) is input into the algorithms of the scheme.

- $\mathsf{UE.KG}(pp, e) \xrightarrow{\$} k_e := (pk_e, sk_e)$: The owner runs the probabilistic key-generation algorithm $\mathsf{UE.KG}$ for epoch e on input of the public parameters. The output is an epoch key $k_e := (pk_e, sk_e)$ composed of the public key (pk_e) and secret key (sk_e) elements.

- $\mathsf{UE.TG}(pp, sk_e, k_{e+1}) \rightarrow \Delta_{e+1}$: The owner generates the update token by running the *deterministic* algorithm $\mathsf{UE.TG}$ on input the public parameters pp, the secret key sk_e of epoch key pair k_e, and epoch key pair k_{e+1} for the proceeding epoch.

- $\mathsf{UE.Enc}(pp, pk_e, m) \xrightarrow{\$} C_e$: The owner runs the probabilistic encryption algorithm on input the public parameters pp, a message $m \in \mathcal{MSP}$, and public key pk_e for epoch e, outputting a ciphertext C_e.

- $\mathsf{UE.Dec}(pp, sk_e, C) \rightarrow \{m, \perp\}$: The owner runs the deterministic algorithm $\mathsf{UE.Dec}$ on input the public parameters pp, a ciphertext C, and secret key sk_e for some epoch e, returning either the message m or abort $\perp$.

- $\mathsf{UE.Upd}(pp, \Delta_{e+1}, C_e) \xrightarrow{\$} C_{e+1}$: The *host* runs the probabilistic update algorithm on inputs the public parameters pp, ciphertext C_e for epoch e, and update token Δ_{e+1} for the *next* epoch $(e+1)$. The resulting updated ciphertext C_{e+1} is output.

Informally, the correctness property ensures that fresh encryptions and updated ciphertexts should decrypt to the underlying plaintext, given the appropriate epoch key [92, 80, 25].

Definition 29 (Correctness)**.** *Given security parameter* λ*, updatable encryption scheme* Π_{PKUE} *formalised in Definition 28 is correct if, for any message* $m \in \mathcal{MSP}$ *and for any* $j \in \{1, \ldots, \mathsf{max}\}$*,* $i \in \{0, \ldots, \mathsf{max}\}$ *with* $\mathsf{max} > i$*, there exists a negligible function* negl *such that the following holds with overwhelming probability.*

$$\Pr\left[\begin{array}{l} pp \xleftarrow{\$} \mathsf{UE.Setup}(1^\lambda); k_{e_j} \xleftarrow{\$} \mathsf{UE.KG}(pp, e_j); \\ \Delta_{e_j} \leftarrow \mathsf{UE.TG}(pp, sk_{e_{j-1}}, k_{e_j}); C_{e_i} \xleftarrow{\$} \mathsf{UE.Enc}(pp, pk_{e_i}, m); \\ \{C_{e_j} \xleftarrow{\$} \mathsf{UE.Upd}(pp, \Delta_{e_j}, C_{e_{j-1}}) : j \in \{i+1, \cdots, \mathsf{max}\}\}; \\ \mathsf{UE.Dec}(pp, sk_{e_{\mathsf{max}}}, C_{e_{\mathsf{max}}}) = m \end{array}\right] \geq 1 - \mathsf{negl}(1^\lambda).$$

3.4 Security Modelling

In this Section, we formally model security for our PKUE primitive and define an essential privacy property in ciphertext-independent UE literature.

We concur with the literature [92, 25] that *ciphertext confidentiality* is an essential security requirement to be satisfied for any updatable encryption scheme. In particular, we focus on capturing indistinguishability security notions for ciphertext unlinkability, defining an equivalent notion of replayable chosen-ciphertext (RCCA)-security [28, 80] for a public-key UE scheme.

As we mentioned in Section 3.1 and Section 3.2, RCCA-security is the strongest level of confidentiality that can be attained in a UE scheme with *probabilistic* updates. Recollect, the stronger notion of CCA security cannot be obtained for the following reason: recording ciphertext updates is vital to ensure the adversary cannot trivially succeed in breaking security. However, from the moment an adversary corrupts an update token, the challenger can no longer keep track of the updates on challenge ciphertexts as they have been re-randomised in the update procedure.

Formally, to prevent an adversary from exploiting the probabilistic update mechanism, we model replayable chosen-ciphertext security UP-IND-RCCA (Definition 31) for a public-key UE scheme (Definition 28). This security notion is based on the security game presented in Figure 3.2. In line with the UE literature [92, 80, 25], we produce a security framework using indistinguishability experiments such that an adversary can access some oracles and a challenger can continuously update lists to record every call to an oracle, as presented in Figure 3.1. Lists are especially important to a challenger for keeping records of adversary's *corruptions* and the *indirect knowledge* an adversary can infer in specific epochs.

3.4.1 Lists

To initialise the UP-IND-RCCA security experiment, the challenger runs $\mathsf{Init}(1^\lambda)$ which outputs the *global state* (GS) oracles have access to throughout. At the start, $\mathsf{GS} := (pp, k_0, \Delta_0, \mathbf{L}, 0)$ contains the public parameters pp generated by the setup algorithm; epoch key $k_0 := (pk_0, sk_0)$; initial update token $\perp \rightarrow \Delta_0$; set $\mathbf{L} := \{\mathcal{L}, \mathcal{M}^*, \mathcal{T}, \mathcal{K}, \mathcal{C}^*\}$ containing initially empty lists that the challenger is required to maintain throughout the experiment and current epoch $0 \rightarrow e$.

List $\mathcal{L}$ is maintained to keep a log of updated versions of *honestly-generated* ciphertexts *and* the corresponding epoch that the adversary learns through calls to the relevant oracles. List $\mathcal{M}^*$ tracks the challenge messages the adversary sends to the challenger. Further, lists $\mathcal{T}$ and $\mathcal{K}$ keep track of the epoch(s) in which the adversary has obtained an update token or an epoch secret key respectively.

We define a list $\mathcal{C}$ that tracks the epochs in which an adversary obtains an *updated version* of the *challenge-ciphertext* via queries to the ciphertext update oracle. The probabilistic nature of ciphertext updates in PKUE means we have to extend list $\mathcal{C}$ to capture additional information, else an adversary can trivially win in the security experiment for Definition 31 (see Section 3.2).

To satisfy RCCA-security, we extend $\mathcal{C}$ to a list labelled $\mathcal{C}^*$ which encapsulates all of the *challenge-equal epochs* in which the adversary knows a *version* of the challenge ciphertext. Versions of the challenge ciphertext can be obtained via oracle queries *or* manually since there are epochs in which the adversary can infer information using corrupted tokens and/or updated ciphertexts (recall Table 3.2). To see this, if an adversary knows a ciphertext $\tilde{C}_e$ from challenge epoch e and update token Δ_{e+1}, then the adversary can manually update the ciphertext to the epoch $(e+1)$ and realise challenge ciphertext $\tilde{C}_{e+1}$. Without making a record of challenge-equal epochs $\{e, e+1\}$, the challenger has no means to track this ciphertext update due to the re-randomisation of updates, which is problematic if the adversary proceeds to corrupt additional tokens or updated ciphertexts. We highlight that the challenge-equal predicate plays a crucial role in calls made to the *update* oracle *and* the winning conditions of the RCCA security experiment, both of which will be described below. We proceed to the formal definition of the challenge-equal predicate.

Definition 30 (Challenge-Equal Predicate)**.** *The recursive predicate* challenge-equal *defines epochs connected to challenge ciphertexts as follows,*

$$\mathcal{C}^* \leftarrow \{e \in \{0, \ldots, e_{\max}\} | \mathsf{challenge\text{-}equal}(e) = \mathsf{true}\}$$
$$\text{and } \mathsf{true} \leftarrow \mathsf{challenge\text{-}equal}(e) \text{ iff :}$$
$$(e \in \mathcal{C}) \vee (\mathsf{challenge\text{-}equal}(e-1) \wedge e \in \mathcal{T})$$
$$\vee(\mathsf{challenge\text{-}equal}(e+1) \wedge (e+1) \in \mathcal{T}).$$

3.4.2 Oracles

Recollect from Section 3.2, our discussion on trivial attacks on probabilistic UE schemes. Within the discussion, we present a predicate dubbed isChallenge (Definition 23). This predicate is used in modelling to prevent the decryption of updated versions of the challenge ciphertext and is an essential component of the decryption and update oracles.

Informally, a ciphertext can be queried to decryption oracle $\mathcal{O}_{\mathsf{Dec}}$ provided one of two conditions does not hold. First, the ciphertext decrypts to one of the adversary's challenge messages $\{m_0, m_1\} \in \mathcal{M}^*$. Second, if the predicate isChallenge, from Definition 23 in Section 3.2, returns true. If either case holds then test is returned. Otherwise, the decryption of a valid ciphertext under the current epoch secret key is returned to the adversary.

The adversary can update *arbitrary* ciphertexts via calls to $\mathcal{O}_{\mathsf{Upd}}$. In return, $\mathcal{A}$ receives a version of the queried ciphertext updated to the current epoch such that the new epoch is added to the list $\mathcal{C}$. In CCA-secure UE schemes, the oracle restricts calls to non-challenge ciphertexts. Conversely, RCCA-secure UE schemes allow any ciphertext to be updated from the prior epoch e_i to the current epoch e. Ciphertext C_e is computed by the update oracle running UE.Upd from epoch e_i to epoch $\{e_{i+1}, \ldots, e\}$ iteratively.

Whilst the update oracle returns a re-encryption of arbitrary ciphertexts to the current epoch, the challenger records the inferable information leading to trivial wins to use as part of the winning conditions in the security experiment. Specifically, if (e_i, C_{e_i}) is an honestly generated ciphertext (that is, in list $\mathcal{L}$) then the current epoch and updated ciphertext (e, C_e) are added to $\mathcal{L}$. Moreover, if the queried ciphertext C decrypts to one of the challenge messages $\{m_0, m_1\} \in \mathcal{M}^*$ or the isChallenge predicate returns true, then the challenger updates list $\mathcal{C}^*$ to contain the challenge-equal epoch e that the queried ciphertext has been updated to.

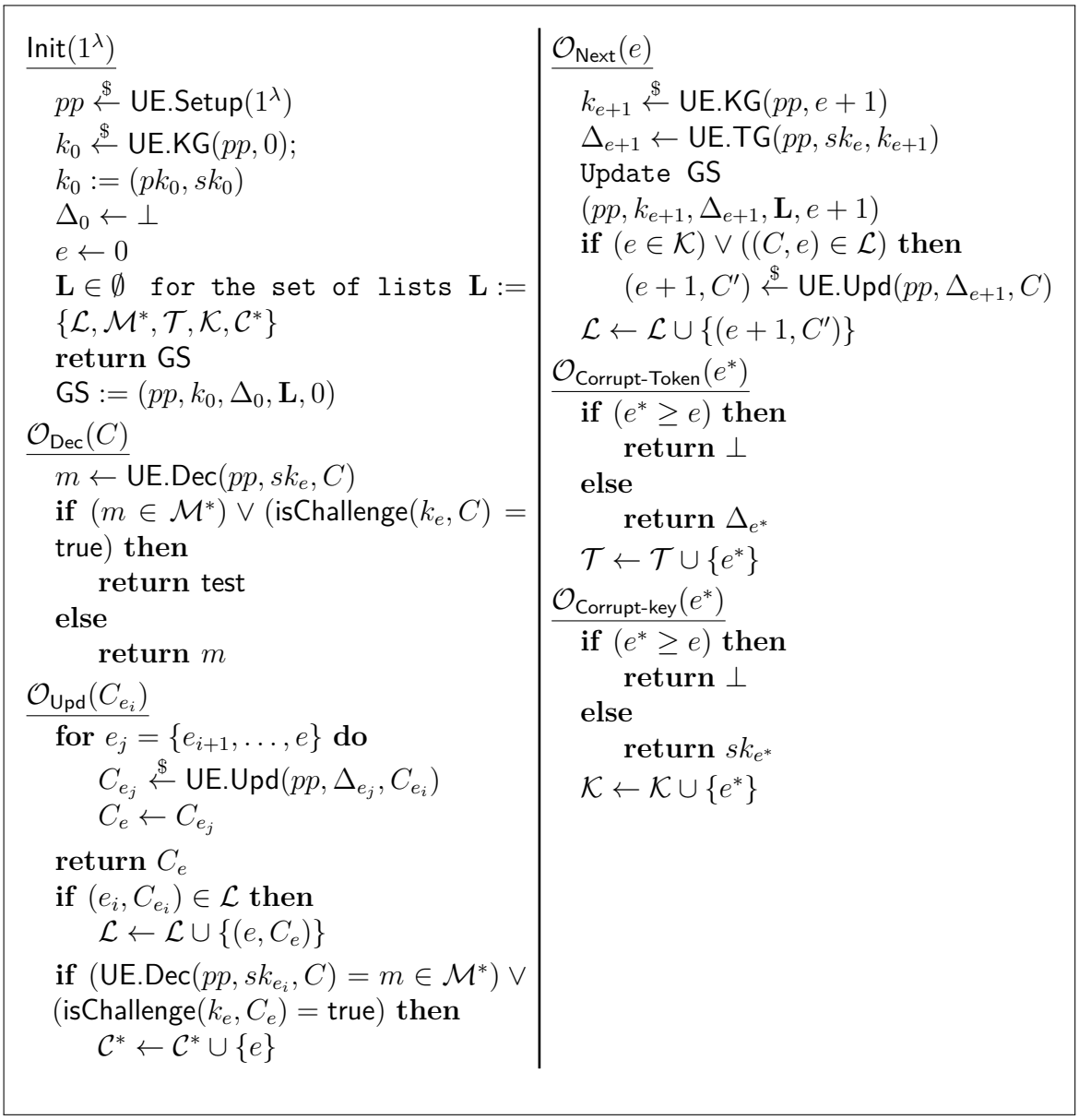

Fig. 3.1 Details of the initialisation phase run by the challenger and the oracles adversary $\mathcal{A}$ calls in epoch e during the security experiment of Definition 31.

Lastly, we highlight the technique used of so-called 'skipping' epoch keys in the update oracle queries. This method applies to a PKUE security model, as it captures the behaviour of an adversary more realistically. The reason is that skipping epochs enables an adversary to corrupt keys in between the old epoch e_i and updated epoch e [80].

Querying oracle $\mathcal{O}_{\mathsf{Next}}$ in challenge epoch e results in the PKUE key-generation algorithm updating the epoch key to k_{e+1} and the token generation algorithm then computes

update token Δ_{e+1}. In turn, the global state must be updated such that the current epoch is $(e+1)$. Observe the following: if the query is in an epoch such that the adversary has corrupted the epoch key *or* the epoch belongs to list $\mathcal{L}$, then the challenge ciphertext must be updated to the next epoch using the generated update token Δ_{e+1} and the new ciphertext is added to the list of honestly updated ciphertexts ($\mathcal{L}$).

Lastly, queries made to $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}$ and $\mathcal{O}_{\mathsf{Corrupt\text{-}Key}}$ are for the corruption of an update token and epoch secret key respectively. The restriction for both oracles is that the adversary's query must be from an epoch preceding the current epoch e. The oracle returns the token (resp. the secret key) for the queried epoch.

Note in the case of key corruption only the epoch secret key is revealed to the adversary, but the update token is not revealed. A dedicated query to the corrupt-token oracle must be made if the adversary wants to learn the token for that epoch as well [92]. We intentionally separate oracle access to tokens and secret epoch key corruption for a given epoch, unlike previous works [92, 25, 80], to provide a granular level of corruption modelling.

3.4.3 The UP-IND-RCCA Security Game

In the security experiment of Figure 3.2, the initialisation process is first run by the challenger. Equipped with a challenge public key, the adversary proceeds to query the detailed oracles in Figure 3.1 with their relevant restrictions, outputting two challenge messages $(m_0, m_1) \in \mathcal{M}^*$ alongside some state information s. The challenger must check that the given messages are of the same length and belong to the message space $\mathcal{MSP}$ of the scheme before proceeding, else the challenger aborts the game and returns $\perp$.

Moving forward the challenger runs the PKUE encryption algorithm on one of the messages m_b for chosen bit $b \in \{0,1\}$, outputting a challenge ciphertext C. The challenger sends C to $\mathcal{A}$ and updates the current epoch to the challenge epoch $\tilde{e}$ as well as the lists in $\mathbf{L}$. Equipped with a challenge ciphertext and state information, the adversary can query the oracles again before outputting a guess bit $b' \in \{0,1\}$. The adversary succeeds in the security experiment if they satisfy the winning conditions and successfully guess the correct bit $(b' = b)$.

$\underline{\mathsf{Exp}^{\mathsf{UP\text{-}IND\text{-}RCCA},b}_{\Pi_{\mathsf{PKUE}},\mathcal{A}}(1^\lambda)}$

$\mathsf{GS} \xleftarrow{\$} \mathsf{Init}(1^\lambda)$ `the initial global state` $\mathsf{GS} = (pp, k_0, \Delta_0, \mathbf{L}, 0)$
$k_e \xleftarrow{\$} \mathsf{UE.KG}(pp, e)$; $k_e := (pk_e, sk_e)$
$(m_0, m_1, s) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(pp, pk_e)$
`Some state information s`
if $|m_0| \neq |m_1| \vee \{m_0, m_1\} \notin \mathcal{MSP}$ **then**
  return $\perp$
else
  $C \xleftarrow{\$} \mathsf{UE.Enc}(pp, pk_e, m_b)$
  $\mathcal{M}^* \leftarrow \mathcal{M}^* \cup (m_0, m_1)$; $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup \{e\}$;$\tilde{e} \leftarrow \{e\}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(pp, C, s)$
if $(\mathcal{K} \cap \mathcal{C}^* = \emptyset)$ **then**
  return b'
`Else abort.`

Fig. 3.2 The security experiment for UP-IND-RCCA-security of a PKUE scheme. Let $\mathcal{O} = \{\mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Next}}, \mathcal{O}_{\mathsf{Upd}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Token}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}\}$ denote the set of oracles that adversary $\mathcal{A}$ calls during the experiment.

Definition 31 (UP-IND-RCCA-Security)**.** *A public-key updatable encryption scheme* Π_{PKUE} *as in Definition 28 is UP-IND-RCCA secure if for any PPT adversary* $\mathcal{A}$ *the following advantage is negligible over security parameter* λ*:*

$$\mathsf{Adv}^{\mathsf{UP\text{-}IND\text{-}RCCA},b}_{\Pi_{\mathsf{PKUE}},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{UP\text{-}IND\text{-}RCCA},0}_{\Pi_{\mathsf{PKUE}},\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{UP\text{-}IND\text{-}RCCA},1}_{\Pi_{\mathsf{PKUE}},\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(1^\lambda)$$

where the indistinguishability experiment is defined in Figure 3.2.

Security Framework For clarity, we note that Figure 3.2 captures a challenger who returns a challenge-ciphertext (C) generated by the encryption algorithm only. At first glance, this model does not appear to capture the indistinguishability of freshly generated ciphertexts versus updated ciphertexts, as the challenge ciphertext is never generated from the update algorithm in the experiment.

However, to demonstrate the satisfaction of ciphertext unlinkability in this Chapter, we do not require an amendment to our security game when following the modular proof technique suggested in [80]. Namely, a generic transformation demonstrating that it is sufficient to consider the underlying encryption and key-rotation capabilities of a scheme

(almost) separately to imply ciphertext unlinkability. In other words, it is only required that the underlying public-key encryption scheme ($\mathsf{UE.Setup}, \mathsf{UE.KG}, \mathsf{UE.Enc}, \mathsf{UE.Dec}$) scheme satisfies standard security to imply unlinkability.

Crucially, for the generic transformation to work we make assumptions on the update feature. Firstly, that our scheme satisfies a notion of perfect re-encryption and secondly, we assume that simulatable reversible update tokens exist in which a reversible token (Δ_e^{-1}) can *downgrade* a ciphertext from epoch $(e+1)$ to epoch e. We stress that a reversible update token is not a feature of a PKUE scheme, rather, it is used as part of the formal security analysis. Both assumptions will be defined and proven to hold in Section 3.8.

Preventing Trivial Wins and Ciphertext Updates To reflect the *adaptive* or *retroactive* corruptions of any update token or epoch secret key in the security experiment, Definition 28 must capture the following:

- **Forward Security** An adversary compromising the epoch secret key (sk_{e^*}) in some epoch e^* does not gain an advantage in decrypting ciphertexts in a prior epoch e (condition $e < e^*$).
- **Post-Compromise Security** An adversary compromising an epoch secret key in some epoch e^* does not gain an advantage in decrypting ciphertexts in an epoch e after the compromise (condition $e > e^*$).

Satisfying both forms of security means the adversary does not further their advantage in decrypting ciphertexts from any epoch, including the challenge epoch. In particular, post-compromise security is important when dealing with sequentially derived update tokens, in which both old and new epoch keys are used.

To illustrate, if $\mathcal{A}$ compromises a secret key sk_{e_i} or $sk_{e_{i+1}}$ and updates token $\Delta_{e_{i+1}}$, one would expect the confidentiality of ciphertexts encrypted for epoch e_{i+1} to remain confidential, however, this is not the case. In the forward security case, the adversary can determine sk_{e_i} having corrupted $(sk_{e_{i+1}}, \Delta_{e_{i+1}})$ and decrypt past ciphertexts C_{e_i}. Conversely, in the post-compromise scenario, the adversary can determine $sk_{e_{i+1}}$ having corrupted $(sk_{e_i}, \Delta_{e_{i+1}})$ and decrypt $C_{e_{i+1}}$which is a future ciphertext in the eyes of the adversary.

Inferable Information			
Epoch	Key-Pairs	Update Tokens	Challenge
e_{i-2}	$k_{e_{i-2}} := (pk_{e_{i-2}}, sk_{e_{i-2}})$	$\Delta_{e_{i-1}}$	$\tilde{C}_{e_{i-2}}$
e_{i-1}	$k_{e_{i-1}} := (pk_{e_{i-1}}, sk_{e_{i-1}})$	Δ_{e_i}	$\tilde{C}_{e_{i-1}}$
e_i	$k_{e_i} := (pk_{e_i}, sk_{e_i})$	$\Delta_{e_{i+1}}$	$\tilde{C}_{e_i}$
e_{i+1}	$k_{e_{i+1}} := (pk_{e_{i+1}}, sk_{e_{i+1}})$	$\Delta_{e_{i+2}}$	$\tilde{C}_{e_{i+1}}$
e_{i+2}	$k_{e_{i+2}} := (pk_{e_{i+2}}, sk_{e_{i+2}})$	$\Delta_{e_{i+3}}$	$\tilde{C}_{e_{i+2}}$

Table 3.2 The leakage profile due to *inferable* information at adversary $\mathcal{A}$'s disposal across multiple epochs, for any $i \in \mathbb{N}$. The blue, red, and purple boxes demonstrate how the elements in the respective coloured boxes are used to infer the cryptographic element highlighted in the same colour.

Crucially queries made to the token and key corruption oracles ($\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}$ and $\mathcal{O}_{\mathsf{Corrupt\text{-}Key}}$ respectively) in epoch e^* are met with a response in which the challenger allows for corruption of tokens and secret keys in a prior epoch e^* to the current epoch e. The *post-compromise security* notion is satisfied due to the challenger updating lists $\mathcal{T} \in \mathcal{C}^*$ and $\mathcal{K}$ respectively with epoch e^*, both of which are lists featured in the winning condition of Figure 3.2, namely, the requirement stating that the intersection of lists $\mathcal{K}$ and $\mathcal{C}^*$ must be empty.

Moreover, to satisfy *forward security* the winning condition ($\mathcal{K} \cap \mathcal{C}^* = \emptyset$) is of importance to prevent the adversary from trivially winning in the security game as follows. The challenge epoch (e) of the experiment cannot belong to the set of epochs in which an update token has been learned or inferred, nor can there exist a single epoch where the adversary knows both the epoch key (public and secret key components) and the (updated) challenge-ciphertext [92]. To see this, if the adversary $\mathcal{A}$ corrupts token Δ_{e+1} in an epoch after which $\mathcal{A}$ has obtained the challenge ciphertext $\tilde{C}$ during epoch e, either by inference or via an update, then the adversary is capable of updating the ciphertext into the next epoch $(e+1)$ [80]. It is easy to see that the forward security condition $e < e^*$ where $(e^* = e + 1)$ in this example is satisfied. We supply Table 3.2 above to further assist in illustrating the information an adversary can infer upon corrupting different elements of a PKUE scheme via oracle queries.

3.5 Epoch Confidentiality

In this Section, we introduce the notion of an epoch confidential public-key UE primitive. We extend Definition 31 to capture both epoch and ciphertext confidentiality simultaneously, formalised by security notion UP-IND-EC-RCCA (Definition 32).

Recollect from the Introduction of this Chapter, a motivation in defining PKUE was so we can utilise public-key techniques in security modelling to formalise novel notions previously overlooked or unattainable in the traditional, symmetric setting. In this Section, we examine how our security framework from Section 3.4 can be supplemented to prevent an adversary from inferring the number of times a ciphertext has been updated. That is, the epoch associated with the ciphertext.

In particular, the exposure of personally identifiable information such as the age of a ciphertext, which we deem to be an undesirable outcome from a security perspective as privacy is no longer ensured since *ciphertext age* directly translates to the number of key updates on the encrypted file. To tackle this problem, we define epoch confidentiality, a notion capturing the leakage of the so-called *ciphertext age* in the ciphertext-independent updatable encryption setting. To re-emphasise, we believe epoch confidentiality is an attractive security property for any UE scheme to guarantee privacy.

Our belief is supported by the literature. In particular, the authors of [19, 25] independently highlighted ciphertext-age leakage to be problematic in real-world scenarios. For instance, [19] considers the setting of dating apps where the number of updates in a UE scheme would reveal how long the person has been a customer which is sensitive information. An alternative application is the storage of medical data, and leakage of ciphertext age could reveal anything from how long an individual has been registered with an organisation, to the age of the individual.

In detail, issues occur in schemes such as that of [52], in which ciphertexts expand as time progresses, producing ciphertext length variance. The authors of [19] demonstrated how *ciphertext length* can be used to trivially infer ciphertext age in a UE scheme. They proposed a tentative solution to require the length of fresh and updated ciphertexts to be equal, a notion known as *compactness*. We note that satisfying compactness alone does not guarantee there will be no leakage of ciphertext age without additionally satisfying *ciphertext unlinkability*.

To illustrate, ciphertext compactness is a property that, on its own, does not prevent patterns from occurring between ciphertexts. Therefore, an adversary remains capable of using these patterns to distinguish ciphertexts generated from fresh encryption versus updates. Indeed, in [19] a simple example is given in which the last bit of the respective ciphertexts differ, and an adversary can determine whether the ciphertext was derived from an update of a pre-existing ciphertext *or* fresh encryption simply by comparing the last bits of the ciphertexts, thus leaking age information.

Note that the UE framework in [19, 25] is for symmetric encryption. In the context of this Chapter, more thought needs to be given to security modelling as we are in the *public-key setting*, so an adversary remains capable of inferring the epoch and consequently the age of a ciphertext by distinguishing the public component of the epoch key used in encryption, token generation and ciphertext updates.

Consequently, our contribution in this Section extends the above by additionally modelling the *indistinguishability of epoch public keys*. This is because ciphertext-independent UE schemes are designed such that the update token can be derived from the current and proceeding epoch keys. Without specific conditions in security modelling, the corruption of epoch keys and update tokens in challenge-equal epochs enables an adversary to infer information about a version of the challenge ciphertext. To be clear, we necessitate the computational indistinguishability of the *epoch public keys* in addition to *ciphertext unlinkability* to provide epoch confidentiality in a given UE scheme.

3.5.1 Security Modelling

To achieve epoch confidentiality for a public-key updatable encryption scheme Π_{PKUE} as in Definition 28, an adversary should be unable to distinguish the public-key component of the epoch key under which a ciphertext has been generated. Thus, possession of distinct public keys and a challenge ciphertext should not give an adversary an advantage in determining which public key and therefore which epoch the ciphertext was encrypted under. This approach to modelling security is inspired by and similar in manner to key privacy [12], which is used to define anonymity in public-key encryption schemes.

To formalise Definition 32, we use the security experiment $(\mathsf{Exp}^{\mathsf{UP\text{-}IND\text{-}EC\text{-}RCCA},b}_{\Pi_{\mathsf{PKUE}},\mathcal{A}}(1^\lambda))$ given in Figure 3.3 . The intuition is to model an indistinguishability game between

the challenger and an adversary $\mathcal{A}$. Initially, the adversary has two challenge public keys (pk_{e_0}, pk_{e_1}) and $\mathcal{A}$ proceeds to query the oracles detailed in Figures 3.1.

To define epoch confidentiality, we introduce a new list of epochs satisfying certain requirements. As a consequence, the oracles of Figure 3.1 must be adapted. Informally, the challenger must check before running every oracle whether the current epoch is contained in our new list and if so, they abort the interaction. In opposition to Definition 31, $\mathcal{A}$ must distinguish not only the underlying message that has been encrypted but also the public key used for encryption, given only access to the relevant oracles and a challenge ciphertext.

The aforementioned extra list maintained by the challenger will be denoted $\tilde{\mathcal{K}}$. Informally, list $\tilde{\mathcal{K}}$ captures the epochs in which adversary $\mathcal{A}$ receives *challenge public keys* and this list must be checked before responding to oracle queries, to prevent trivial wins. The initialisation process runs as normal (see Figure 3.1), however, there is a nuanced difference in the remaining oracles.

The adapted oracles run as described in Section 3.4, Figure 3.1 with the following prerequisites:

- List $\tilde{\mathcal{K}}$ is initialised, at time 0, by running $\mathsf{Init}(1^\lambda)$ and the set of lists contained within the global state now becomes $\mathbf{L} = \{\mathcal{L}, \mathcal{M}^*, \mathcal{T}, \mathcal{K}, \tilde{\mathcal{K}}, \mathcal{C}^*\}$ such that $\mathbf{L} \in \emptyset$.
- $\mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Next}}$: if the current epoch $e \in \tilde{\mathcal{K}}$ then the corresponding oracle that has been queried by $\mathcal{A}$ does not proceed with running the oracle nor responding to an invalid query. Otherwise, the oracle runs as normal.
- $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}$: if the queried epoch e^* belongs to the set $\tilde{\mathcal{K}}$ then the corresponding oracle that has been queried by $\mathcal{A}$ does not proceed with running nor responding to an invalid query. Otherwise, the oracle runs as normal.
- $\mathcal{O}_{\mathsf{Upd}}$: if the queried epoch $e_i \in \tilde{\mathcal{K}}$ *or* current epoch $e \in \tilde{\mathcal{K}}$ then the oracle ceases running on an invalid query and aborts ($\perp$). Otherwise, the oracle runs as normal.

In other words, the game in Figure 3.3 starts by initialising the global state GS as previously explained. Next, the key-generation algorithm is run twice to generate epoch keys $k_{e_0} = (pk_{e_0}, sk_{e_0})$ and $k_{e_1} = (pk_{e_1}, sk_{e_1})$ for distinct epochs of time e_0, e_1.

$\mathsf{Exp}^{\mathsf{UP\text{-}IND\text{-}EC\text{-}RCCA},b}_{\Pi_{\mathsf{PKUE}},\mathcal{A}}(1^\lambda)$

$\mathsf{GS} \xleftarrow{\$} \mathsf{Init}(1^\lambda)$ `for initial global state` $\mathsf{GS} = (pp, k_0, \Delta_0, \mathbf{L}, 0)$
$k_{e_0} \xleftarrow{\$} \mathsf{UE.KG}(pp, e_0)$, $k_{e_1} \xleftarrow{\$} \mathsf{UE.KG}(pp, e_1)$ `such that` $k_{e_0} \neq k_{e_1}$, $(e_0, e_1) \notin \mathcal{K}$
$k_{e_0} := (pk_{e_0}, sk_{e_0}), k_{e_1} := (pk_{e_1}, sk_{e_1})$
$\tilde{\mathcal{K}} \leftarrow \{(e_0, e_1)\} \cap \tilde{\mathcal{K}}$
$(m_0, m_1, s) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(pp, pk_{e_0}, pk_{e_1})$
`Some state information` s
if $|m_0| \neq |m_1| \vee \{m_0, m_1\} \notin \mathcal{MSP} \vee (m_0 = m_1)$ **then**
  return $\perp$
else
  $C \xleftarrow{\$} \mathsf{UE.Enc}(pp, pk_{e_b}, m_b)$
  $\mathcal{M}^* \leftarrow \mathcal{M}^* \cup \{(m_0, m_1)\}$; $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup \{e_b\}$; $\tilde{e} \leftarrow \{e_b\}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(pp, C, s)$
if $(\mathcal{K} \cap \mathcal{C}^* = \emptyset)$ **then**
  return b'
`Else abort.`

Fig. 3.3 The security game for a PKUE scheme satisfying UP-IND-EC-RCCA-security, where set $\mathbf{L} = \{\mathcal{L}, \mathcal{M}^*, \mathcal{T}, \mathcal{K}, \tilde{\mathcal{K}}, \mathcal{C}^*\}$ is initially empty, $\mathcal{O}$ is the set of oracles an adversary $\mathcal{A}$ calls, and s defines some *state* information output by the adversary.

Given the challenge public keys (pk_{e_0}, pk_{e_1}), the adversary proceeds to query oracles $\mathcal{O} = \{\mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Upd}}, \mathcal{O}_{\mathsf{Next}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Token}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}\}$ to output valid challenge messages $(m_0, m_1) \in \mathcal{MSP}$ required to be of the same length, alongside some state information s. Subsequently, the challenger encrypts m_b using public key pk_{e_b}, for a pre-determined bit $b \in \{0, 1\}$, sending the challenge ciphertext C to the adversary. Using this challenge ciphertext alongside the state information s and further access to previously detailed oracles, $\mathcal{A}$ guesses the bit b' and succeeds in the game if their guess corresponds to the bit b chosen before the experiment began. More formally,

Definition 32 (UP-IND-EC-RCCA-Security)**.** *A public-key updatable encryption scheme Π_{PKUE} formalised in Definition 28, satisfies UP-IND-EC-RCCA security if for any PPT adversary $\mathcal{A}$ there exists a negligible function negl such that*

$$\mathrm{Adv}^{\mathsf{UP\text{-}IND\text{-}EC\text{-}RCCA},b}_{\Pi_{\mathsf{PKUE}},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{UP\text{-}IND\text{-}EC\text{-}RCCA},0}_{\Pi_{\mathsf{PKUE}},\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{UP\text{-}IND\text{-}EC\text{-}RCCA},1}_{\Pi_{\mathsf{PKUE}},\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(1^\lambda).$$

Remark 1. Recall in Section 3.4 that we discussed the *generic* transformation first proposed in [80] in which the security of a standard encryption scheme, coupled with the assumption of perfect re-encryption for the update feature, directly implies ciphertext unlinkability of a PKUE scheme. We make explicit that the same transformation is utilised in the security analysis of our construction (Definition 39) to illustrate that epoch confidentiality is achieved.

To distinguish our contribution we compare it to closely related work of [19]. The authors define a notion of confidentiality in the ciphertext-dependent UE setting that captures the idea of hiding the age of the ciphertext, namely the number of times that the ciphertext was re-encrypted since it was initially created. Their notion of confidentiality is considered stronger than prior notions of confidentiality from works such as [24, 52] since [19] require re-encrypted ciphertexts to be computationally indistinguishable from freshly generated ciphertexts, no matter how many times re-encryption occurs. This requirement is in addition to message confidentiality and re-encryption indistinguishability security notions defined in [24, 52].

Whilst we are motivated by the same reasoning as [19], our notion of epoch confidentiality is designed for *ciphertext-independent PKUE* schemes, which we established as a distinct strain of updatable encryption. Secondly, our security model captures the indistinguishability of epoch public keys, not ciphertexts. Definition 32 does cover ciphertext confidentiality with the addition of epoch confidentiality which means that an adversary cannot use epoch public keys to distinguish the epoch public key used to generate a ciphertext, be that from encryption or ciphertext updates. As a consequence, an adversary cannot determine the epoch a ciphertext has derived from.

3.6 Construction Preliminaries

In this Section, we present the building blocks and assumptions required to construct our concrete public-key updatable encryption scheme Π_{PKUE}. We use an ElGamal-based public-key UE scheme; a message-independent signature scheme and a linear malleable proof system, detailed in this order.

First, our construction Π_{PKUE} is a modified version of the NYUE scheme [80], itself based upon RISE [92] which we convert to the public-key setting.

Definition 33 (Rise)**.** *Given security parameter λ, the RISE PKUE scheme is a tuple of six PPT algorithms $\Pi_{\mathsf{RISE}} = (\mathsf{Rise.Setup}, \mathsf{Rise.KG}, \mathsf{Rise.TG}, \mathsf{Rise.Enc}, \mathsf{Rise.Dec}, \mathsf{Rise.Upd})$ defined as follows:*

- $\mathsf{Rise.Setup}(1^\lambda) \xrightarrow{\$} pp$: the data owner runs the setup algorithm on input the security parameter λ and outputs public parameters $pp := (\mathbb{G}_1, g, q)$ for $\mathbb{G}_1$ a cyclic group with generator g, of prime order $q > 2^\lambda$.
- $\mathsf{Rise.KG}(pp, e) \xrightarrow{\$} k_e$: given inputs the public parameters pp and epoch e, the key-generation algorithm outputs the epoch key-pair $k_e := (pk_e, sk_e)$, where the secret key sk_e is a randomly chosen value $x \xleftarrow{\$} \mathbb{Z}_q^*$, and the public key pk_e is computed as $g^x \in \mathbb{G}_1$.
- $\mathsf{Rise.TG}(pp, sk_e, k_{e+1}) \rightarrow \Delta_{e+1}$: the owner parses $k_{e+1} = (pk_{e+1}, sk_{e+1})$ the epoch key and computes the update token as $\Delta_{e+1} \leftarrow (sk_{e+1}/sk_e, pk_{e+1})$, returning Δ_{e+1}. Note that the token space is defined over $\mathbb{Z}_q^* \times \mathbb{G}_1$.
- $\mathsf{Rise.Enc}(pp, pk_e, m) \xrightarrow{\$} C_e$: for $m \in \mathbb{G}_1$ the data owner draws $r \xleftarrow{\$} \mathbb{Z}_q$, and returns $C_e = (pk_e^r, g^r m) \in (\mathbb{G}_1)^2$.
- $\mathsf{Rise.Dec}(pp, sk_e, C_e) \rightarrow \{m, \perp\}$: the data owner parses $C_e = (C_1, C_2)$ and returns $m \leftarrow C_2 \cdot {C_1}^{-1/sk_e}$. Else, they abort and output $\perp$.
- $\mathsf{Rise.Upd}(pp, \Delta_{e+1}, C_e) \xrightarrow{\$} C_{e+1}$: the data host parses $\Delta_{e+1} = (\Delta, pk_{e+1})$ and $C_e = (C_1, C_2)$, draws $r' \xleftarrow{\$} \mathbb{Z}_q$, computes $C_1' \leftarrow C_1^\Delta \cdot pk_{e+1}^{r'}$, $C_2' \leftarrow C_2 \cdot g^{r'}$, and returns $C_{e+1} = (C_1', C_2')$.

Given honestly generated consecutive epoch keys $k_e := (pk_e, sk_e) = (g^x, x)$ and $k_{e+1} := (pk_{e+1}, sk_{e+1}) = (g^{x'}, x')$, token $\Delta_{e+1} = (sk_{e+1}/sk_e, pk_{e+1})$ such that $\Delta := sk_{e+1}/sk_e$, and ciphertext $C_e := (C_1, C_2) = (pk_e^r, g^r m)$, correctness of updates is demonstrated as follows:

- $C_1' = C_1^\Delta \cdot pk_{e+1}^{r'} = (pk_e^r)^{sk_{e+1}/sk_e} \cdot pk_{e+1}^{r'} = (g^{xr})^{(x'/x)} \cdot (g^{x'})^{r'} = g^{x'(r+r')} = pk_{e+1}^{(r+r')}$;
- $C_2' = C_2 \cdot g^{r'} = g^r m \cdot g^{r'} = g^{r+r'} m$.

Since randomness $(r + r') \in \mathbb{Z}_q$, updated ciphertext $C_{e+1} := (C_1', C_2')$ is of the correct form.

If we remove the update feature of Π_{Rise}, from Definition 33, then we are left with the public-key encryption scheme $\Pi_{\mathsf{PKE}} = (\mathsf{Rise.Setup}, \mathsf{Rise.KG}, \mathsf{Rise.Enc}, \mathsf{Rise.Dec})$. To prove the security of our construction (Section 3.8), we move from the updatable to the standard setting by the method of reduction. Thus, we reduce the security of our construction to that of Π_{PKE} and use the ensuing remark to support our analysis.

Remark 2. The authors of [12] demonstrated that an El-Gamal-based PKE scheme such as Π_{PKE} satisfies key privacy under chosen-plaintext attacks (IND-IK-CPA), assuming the hardness of the DDH in $\mathbb{G}_1$. We discuss this in more detail in Section 3.8.

Recall Figure 2.4 used in Definition 15 (Section 2.4), which formalises the strictly stronger key-private *replayable* security notion (IND-IK-RCCA) for a PKE scheme. A *key-private* CPA-secure PKE scheme models an adversary's capability of distinguishing the public key used to encrypt a chosen message from the challenge ciphertext.

Message Independent Updatable Signature Scheme The authors of [34] were the first to introduce the explicit definition of updatable signature (US) schemes, present concrete US schemes from key-homomorphic signature, and formally prove security properties in the updatable setting using the key-insulation technique of [80]. In particular, they modelled security against updatable-signature and unlinkable-signature existentially-unforgeable chosen message attacks.

Previously, there were notions of signatures supporting key rotation in addition to guaranteeing unforgeability when allowing queries under (adversarially) updated keys. However, [34] highlighted that none of the prior literature rigorously defines security when signatures were updated between different keys.

In the context of our work, we use a *message-independent* updatable signature (MI-US) scheme formalised in [34] to build a concrete PKUE scheme. Message independence is desirable since we want signature updates without the input of the underlying message, given the outsourced environment in which UE schemes are applied. Moreover, this property is important in practical applications as access to the message is not required, meaning that signature verification can be stored and updates can occur at a later time, in batches. To achieve message independence the authors of [34] adapt signature schemes such as [22, 107] such that updates to signatures are multiplicative. In the following definition, we present the concrete MI-US scheme proposed by [34] which is based on the efficient, short signature scheme given in [107].

Definition 34 (MI-US)**.** *Given security parameter $\lambda \in \mathbb{N}$, let the message-independent updatable signature scheme be defined by a tuple of six PPT algorithms $\Pi_{\mathsf{US}} = (\mathsf{US.Setup}, \mathsf{US.KG}, \mathsf{US.Next}, \mathsf{US.Sign}, \mathsf{US.Upd}, \mathsf{US.Ver})$ as follows,*

- $\mathsf{US.Setup}(1^\lambda) \xrightarrow{\$} pp$: given security parameter λ, run the bilinear group generator algorithm $\mathsf{BG.Gen}(1^\lambda) \rightarrow pp := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g}, \hat{e}, q)$ where g is the generator of $\mathbb{G}_1$ and $\tilde{g} \xleftarrow{\$} \mathbb{G}_2$ for groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order q and asymmetric bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Output public parameters pp.
- $\mathsf{US.KG}(pp) \xrightarrow{\$} (sig_e, vk_e)$: randomly choose values $x \xleftarrow{\$} \mathbb{Z}_q, y \xleftarrow{\$} \mathbb{Z}_q^*$; set $sig_e := (x, y)$ and $vk_e := (\tilde{X}, \tilde{Y}) = (\tilde{g}^x, \tilde{g}^y)$.
- $\mathsf{US.Next}(pp, sig_e, vk_e) \xrightarrow{\$} (\Delta_{e+1}, vk_{e+1})$: define $\Delta_{e+1} := (\Delta_{e+1,1}, \Delta_{e+1,2})$ such that $\Delta_{e+1,1} \xleftarrow{\$} \mathbb{Z}_q^*$; set $\Delta_{e+1,2} \xleftarrow{\$} \mathbb{Z}_q$. Compute new keys $sig_{e+1} := (x', y') = (x \cdot \Delta_{e+1,1} + \Delta_{e+1,2}, y \cdot \Delta_{e+1,1})$ and $vk_{e+1} := (\tilde{X}', \tilde{Y}') = (\tilde{X}^{\Delta_{e+1,1}} \cdot \tilde{g}^{\Delta_{e+1,2}}, \tilde{Y}^{\Delta_{e+1,1}})$.
- $\mathsf{US.Sign}(pp, sig_e, m) \xrightarrow{\$} \sigma_e$ for message $m \in \mathcal{MSP} := \mathbb{Z}_q$, randomly choose $h \xleftarrow{\$} \mathbb{G}_1^*$ and set signature $\sigma_e := (\sigma_{e,1}, \sigma_{e,2}) = (h, h^{(x+y \cdot m)})$.
- $\mathsf{US.Upd}(pp, \Delta_{e+1}, \sigma_e) \xrightarrow{\$} \sigma_{e+1}$: choose randomness value $r \leftarrow \mathbb{Z}_q^*$ and compute the updated signature $\sigma_{e+1} := (\sigma_{e+1,1}, \sigma_{e+1,2}) = (\sigma_{e,1}^r, \sigma_{e,2}^{r \cdot \Delta_{e+1,1}} \cdot \sigma_{e,1}^{r \cdot \Delta_{e+1,2}})$.
- $\mathsf{US.Ver}(pp, vk_e, m, \sigma_e) \rightarrow b \in \{0, 1\}$: parse $\sigma_e := (\sigma_{e,1}, \sigma_{e,2})$ and first check that $\sigma_1 \neq 1_{\mathbb{G}_1}$. If so, check $\hat{e}(\sigma_{e,1}, \tilde{X} \cdot \tilde{Y}^m) \stackrel{?}{=} \hat{e}(\sigma_{e,2}, \tilde{g})$. If the equality holds, output 1 and output 0 otherwise.

Correctness of verification and updates is demonstrated respectively as follows:

1. If $\sigma_e := (\sigma_{e,1} = h, \sigma_{e,2} = h^{(x+y \cdot m)})$ then $\hat{e}(\sigma_{e,1}, \tilde{X} \cdot \tilde{Y}^m) = \hat{e}(h, \tilde{g})^{(x+y \cdot m)} = \hat{e}(h^{(x+y \cdot m)}, \tilde{g}) = \hat{e}(\sigma_{e,2}, \tilde{g})$.
2. Given $\sigma_{e+1} := (\sigma_{e,1}^r, \sigma_{e,2}^{r \cdot \Delta_{e+1,1}} \cdot \sigma_{e,1}^{r \cdot \Delta_{e+1,2}}) = (h^r, h^{(x+y \cdot m) \cdot r \cdot \Delta_{e+1,1}} \cdot h^{r \cdot \Delta_{e+1,2}}) = (h^r, h^{r(\cdot (x+y \cdot m) \cdot \Delta_{e+1,1} + \Delta_{e+1,2})}) = (h^r, h^{r(x'+y' \cdot m)})$, the randomisation property of the scheme allows for a valid updated signature by replacing $h \in \mathbb{G}_1^*$ with $h' := h^r \in \mathbb{G}_1^*$.

Security of Definition 34 follows from the assumptions and proofs given in [107, 34]. For clarity, we state the relevant definitions and security assumptions relevant to our work in the following, and the proofs follow from [34]. Firstly, all US schemes proposed in

[34], including Definition 34, are based on additive or multiplicative *key-homomorphic* (KH) signature schemes ($\Sigma = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Adapt}, \mathsf{Vrfy})$) whereby Σ has *perfect adaptation*. Formally,

Definition 35 (Perfect Adaptation)**.** *Key-homomorphic signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Adapt}, \mathsf{Vrfy})$ *satisfies the* perfect adaptation *property. Using a secret-key to public-key homomorphism* $\mu : \mathbb{H} \to \mathbb{E}$ *with* $\mathbb{H}$ *being the secret key space of* Σ*, the following holds,*

$$\forall sk, sk' \in \mathbb{H} : \mu(sk + sk') = \mu(sk) \cdot \mu(sk') \text{ and } \forall (sk, pk) \leftarrow \mathsf{Gen}(1^\lambda) \text{ it holds that } pk = \mu(sk).$$

then there exists a PPT algorithm $\mathsf{Adapt}(pp, pk, m, \sigma, \Delta) \to (pk', \sigma'), \forall \Delta \in \mathbb{H}$.

Assumption 1. *Let* $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, \hat{e}, g, \tilde{g})$ *be a bilinear group setting with* $g, \tilde{g}$ *generators groups* $\mathbb{G}_1, \mathbb{G}_2$ *respectively. For* $(\tilde{X} = \tilde{g}^x, \tilde{Y} = \tilde{g}^y)$ *for* $x \xleftarrow{\$} \mathbb{Z}_q, y \xleftarrow{\$} \mathbb{Z}_q^*$*, we define the oracle* $\mathcal{O}(m)$ *for message* $m \in \mathbb{Z}_q$ *which chooses* $h \xleftarrow{\$} \mathbb{G}_1^*$ *and outputs the pair* $P = (h, h^{(x+y \cdot m)})$*. Given* $(\tilde{X}, \tilde{Y})$ *and unlimited access to the defined oracle, no adversary can efficiently generate a pairing like* P *with randomness* $h \neq 1_{\mathbb{G}_1}$*, for a new message* m^* *that was not queried to the oracle.*

The next two definitions formalise US-EUF-CMA and US-UU-CMA security in Figures 3.4, 3.5 respectively. In essence, the former notion encapsulates a standard signature scheme security notion translated to the updatable setting and the latter security notion captures the indistinguishability of freshly generated versus updated signatures against chosen message attacks.

Informally, the set of lists $\mathbf{L} = (\mathcal{I}, \mathcal{K}, \mathcal{T}, \mathcal{S})$ track all the keys and tokens corrupted; epochs in which the key was corrupted; epochs in which the token was corrupted; and the tuple of the epoch, message, and signature in which an adversary obtained the signature via fresh signing or an update. We note that oracles $\mathcal{O}' := \{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Next}}, \mathcal{O}_{\mathsf{Upd}}, \mathcal{O}_{\mathsf{Ver}}, \mathcal{O}_{\mathsf{Corrupt}}\}$ are the signature equivalent of our UE security oracles presented in Section 3.4, except for $\mathcal{O}_{\mathsf{Ver}}$ which on an input message and signature pair $(m, \sigma_{e'})$ returns the verification bit b using public verification key $vk_{e'}$. Furthermore, algorithm UpdCH in Figure 3.5 is the repeated application of the US update algorithm from epoch e' to challenge epoch e^*. We defer the reader to [34] for a full treatment of the security of an updatable signature scheme. More formally,

Definition 36 (US-EUF-CMA Security)**.** *The updatable signature scheme* Π_{US} *given in Definition 34 is* US-EUF-CMA *secure iff a PPT adversary* $\mathcal{A}$ *has a negligible advantage in the security experiment defined in Figure 3.4. Namely,*

$$Adv^{\text{US-EUF-CMA}}_{\Pi_{US},\mathcal{A}}(1^\lambda) := Pr[Exp^{\text{US-EUF-CMA}}_{\Pi_{US},\mathcal{A}}(1^\lambda) = 1] \leq negl(1^\lambda).$$

$\underline{\mathsf{Exp}^{\text{US-EUF-CMA}}_{\Pi_{\mathsf{US}},\mathcal{A}}(1^\lambda)}$

$pp \xleftarrow{\$} \mathsf{US.Setup}(1^\lambda)$
$(sig_1, vk_1) \xleftarrow{\$} \mathsf{US.KG}(pp)$
$\mathbf{L} = (\mathcal{I}, \mathcal{K}, \mathcal{T}, \mathcal{S})$ `for` $\mathcal{I} := \{(sig_1, vk_1), \perp\}, \mathcal{K} := \mathcal{T} := \mathcal{S} := \emptyset$
$(m^*, \sigma^*_{e^*}) \leftarrow \mathcal{A}^{\mathcal{O}'}(1^\lambda)$
if $\mathsf{US.Vrfy}(pp, vk_{e^*}, m^*, \sigma^*_{e^*}) = \mathsf{accept}$ **then**
  return 1
`Else abort.`

Fig. 3.4 The security game modelling updatable-signature existentially-unforgeable chosen-message security of the updatable signature scheme Π_{US}.

Definition 37 (US-UU-CMA Security)**.** *The updatable signature scheme* Π_{US} *given in Definition 34 is* US-UU-CMA *secure iff a PPT adversary* $\mathcal{A}$ *has a negligible advantage in the security experiment defined in Figure 3.5. Namely,*

$$Adv^{\text{US-UU-CMA}}_{\Pi_{US},\mathcal{A}}(1^\lambda) := |Pr[Exp^{\text{US-UU-CMA},1}_{\Pi_{US},\mathcal{A}}(1^\lambda) = 1] - Pr[Exp^{\text{US-UU-CMA},0}_{\Pi_{US},\mathcal{A}}(1^\lambda) = 1]| \leq negl(1^\lambda).$$

Lemma 1. *The message-independent updatable signature scheme* Π_{US} *of Definition 34 is secure against updatable signature existentially unforgeable chosen message attacks* (US-EUF-CMA) and *unlinkable signature chosen message* (US-UU-CMA) *attacks given that the underlying signature scheme is* EUF-CMA *secure and satisfies* perfect adaptation *([34]).*

Linear Malleable Proofs We require a (linear) malleable proof system to support the re-randomisation of commitments and proofs in the *updatable signature scheme* used to build a concrete updatable encryption scheme. Linear malleability of proofs essentially ensures that commitments are *homomorphic*.

$\underline{\mathsf{Exp}^{\text{US-UU-CMA},b}_{\Pi_{\mathsf{US}},\mathcal{A}}(1^\lambda)}$

$pp \xleftarrow{\$} \mathsf{US.Setup}(1^\lambda)$
$(sig_1, vk_1) \xleftarrow{\$} \mathsf{US.KG}(pp)$
$\mathbf{L} = (\mathcal{I}, \mathcal{K}, \mathcal{T}, \mathcal{S})$ `for` $\mathcal{I} := \{(sig_1, vk_1), \perp\}, \mathcal{K} := \mathcal{T} := \mathcal{S} := \emptyset$
$(m^*, e^*) \leftarrow \mathcal{A}^{\mathcal{O}'}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$\sigma^{(0)} \leftarrow \mathsf{US.Sign}(pp, sig_{e^*}, m^*), \sigma^{(1)} \leftarrow \mathsf{UpdCH}(m^*)$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}'}(\sigma^{(b)})$
if $(e', m^*, \cdot) \in \mathcal{S} \wedge (e' < e^*) \wedge (b = b^*)$ **then**
  return b^*
`Else abort.`

Fig. 3.5 The game modelling security against updatable-signature unlinkable-updates under chosen-message attacks of the updatable signature scheme Π_{US}.

The concrete proof system used to support our construction is a Groth-Sahai proof system [67, 50], needed to ensure the *randomisability* and *malleability* of our updateable encryption scheme. Commonly, GS-proofs are *commit-and-prove* systems for quadratic equations. One commits to a witness and then proves the committed witness is a solution for the given system of equations. In other words, it is a system that proves the satisfiability of sets of linear and quadratic pairing equations.

Note that Definition 38 presented below can be viewed as *dual-mode* proof systems in the sense that there are two setup algorithms: $\mathsf{GS.SetupH}(gp), \mathsf{GS.SetupB}(gp)$ generates hiding and binding common reference strings (crs_H, crs_B) respectively. As a consequence, the resulting proof π is *perfectly hiding* and *perfectly sound* respectively. Furthermore, binding commitments to groups are *extractable* [80] which is a property implying structure-preserving cryptographic schemes. For notational purposes, the first two algorithms have been combined into one setup algorithm ($\mathsf{GS.Setup}$). We provide an explicit definition as follows.

Definition 38 (GS-Proof System)**.** *Let $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q)$ be a pairing group. Define a GS-proof system as a tuple of four PPT algorithms $\Pi_{GS} = (\mathsf{GS.Setup}, \mathsf{GS.Com}, \mathsf{GS.Prove}, \mathsf{GS.VRFY})$ that works in the following way,*

- $\mathsf{GS.Setup}(gp) \xrightarrow{\$} crs$: global parameters are input into the probabilistic setup algorithm that outputs a common reference string $crs := (crs_H, crs_B)$ which is a

pair of *hiding* and *binding* common reference strings. Let the output (crs) be implicitly input into the following algorithms.

- $\mathsf{GS.Com}_{\mathbf{t}}(x; r) \xrightarrow{\$} c$: the probabilistic commitment algorithm takes a variable of type $\mathbf{t}$ and outputs a commitment (c) for input x, using randomness r. Note that the type $\mathbf{t}$ determines what group the input x is in, for example. Further, the implicit input of crs means that the commitment is perfectly hiding and binding respectively.

- $\mathsf{GS.Prove}(\epsilon, (w_i)_i, (r_i)_i) \xrightarrow{\$} \pi$: Define input ϵ to be an equation which can involve variables, constants over $\mathbb{G}_1, \mathbb{G}_2$ or scalar $\mathbb{F}_q$ (where $\mathbb{F}_q$ is a finite field of prime order q). Let input $(w_i)_i$ be a solution to ϵ[3]. Let input $(r_i)_i$ be commitment randomness for w_i and $(C_i)_i = (\mathsf{Com}_{\mathbf{t}_i}(w_i; r_i))_i$ be the set of commitments. Thus, running the algorithm $\mathsf{GS.Prove}(\epsilon, (w_i)_i, (r_i)_i) \xrightarrow{\$} \pi$ produces a proof.

- $\mathsf{GS.VRFY}(\epsilon, (c_i)_i, \pi) \rightarrow \{\mathsf{accept}, \mathsf{reject}\}$: Given an equation ϵ, commitments $(C_i)_i$ and a proof π, the deterministic verification algorithm either accepts or rejects the proof.

Let a statement/equations $(\epsilon_j)_j$ be defined by $(\Gamma_{i,j}, [a_j]_1, [b_i]_2)$ and the solution/witness defined by $([x_i]_1, [y_j]_2)$. It suffices to prove and verify each equation ϵ_j for the *same* committed solution $(C_i)_i$. The proof π and commitments are used implicitly to prove the satisfiability of the system of equations of the form:

$$\sum_{i,j} \Gamma_{i,j} \hat{e}([x_i]_1, [y_j]_2) + \sum_{i} \Gamma_{i,j} \hat{e}([x_i]_1, [b_i]_2) + \sum_{j} \Gamma_j \hat{e}([a_j]_1, [y_j]_2) = [t]_T,$$

where $\Gamma_{i,j} \in \mathbb{F}_q$, $([x_i]_i, [a_j]_1) \in \mathbb{G}_1$, $([b_i]_2, [y_j]_2) \in \mathbb{G}_2$, and $[t]_T \in \mathbb{G}_T$ and pairing $\hat{e}$ is some bilinear map between $\mathbb{F}_q, \mathbb{G}_1$, and $\mathbb{G}_2$.

The technique of proving in our construction will use an OR-compilation of statements (equation) (S^0, S^1) to enumerate the constants and variables contained within the two equations such that the witness is $\mathsf{OR}(S^0, S^1)$. This is a commit-and-prove method, such that the commitment is necessarily linearly malleable, to suit the update feature of our construction.

We assume commitments are additively homomorphic, that is, $C_i(a; r) + C_i(b; s) = C_i(a + b; r + s)$. Further, we assume perfectly re-randomisable commitments and proofs

[3]That is, is a suitable variable assignment with variable w_i of type $\mathbf{t}_i$.

$((C_i)_i, \pi_j)$ respectively in the sense that re-randomised commitments and proofs are identically distributed to the freshly generated ones. We highlight that re-randomisability is the property required for unlinkability in probabilistic updatable encryption schemes.

The high-level idea to prove one of the two statements (S^0, S^1) is true is as follows. Given bit $b \in \{0, 1\}$ and solution $(x_i)_i$ to equation S^b, the constants and witness x_i of valid equation S^b are copied to solution $x_{i,b}$. The unsatisfied statement S^{1-b} and witness are copied to 0, to ensure equalities trivially hold. Consequently, witness $(x_i)_i$ is easy to reconstruct from a witness for $\mathsf{OR}(S^0, S^1)$.

3.7 An Epoch Confidential Construction

In this Section, we present a *concrete* construction of a public-key UE scheme which we later prove to satisfy UP-IND-EC-RCCA security in Section 3.8. We chose to define a concrete construction to demonstrate the existence of a PKUE scheme satisfying epoch confidentiality in practice and to compare the efficiency of our construction versus concrete symmetric UE schemes.[4]

High Level Idea Our construction (Π_{PKUE}), presented in Definition 39, generates two epoch key pairs $(k_e^1 = (pk_e^1, sk_e^1), k_e^2 = (pk_e^2, sk_e^2))$ for the *same* epoch. This is achieved following the public-key RISE key generation algorithm presented in Definition 33 of Section 3.6, which is an El-Gamal-based PKUE scheme. Then, we encrypt message m twice using the encryption algorithm in Definition 33 with the two generated epoch public keys (pk_e^1, pk_e^2) and specified randomness (r_1, r_2 respectively). In addition, the ciphertext contains a NIZK proof (π) of validity, using a dual-mode GS-proof system presented in Definition 38, Section 3.6. We note that a message-independent updatable signature scheme Π_{US} from Definition 34, Section 3.6, is used in running the proof system. We chose to use this scheme since Π_{US} is an updatable building block satisfying a notion of signature unlinkability, and crucially, it is compatible with the GS proof system.[5]

[4] Concrete schemes provide quantitative information for the computational security of a cryptographic scheme by explicitly bounding the maximum success probability of any PPT adversary [76]. Quantitative security is useful in practice when care is taken with the conclusions made from security analysis.

[5] In the sense that it is a *structure-preserving* signature scheme (Definition 18, Section 3.6).

To decrypt, we first require the verification of the proof contained within the ciphertext and, if accepted, the decryption algorithm follows the decryption process given in Definition 33.

The token generation procedure follows Definition 33 in producing a ciphertext update token and additionally runs the updatable signature token generation algorithm.

Lastly, ciphertext updates follow a four-step process of 1) verification; 2) key rotation through the use of the generated update tokens and update procedure of ciphertext and signatures respectively, together with a new NIZK proof; 3) re-randomisation of the new ciphertexts, followed by 4) re-randomisation of the new signature and NIZK proof.

Formal Description To build construction Π_{PKUE} we use a modified Naor-Yung transform [102] applied to the public-key UE scheme Π_{Rise} , a *linear malleable* NIZK GS-proof system Π_{GS} [67], and a message-independent updatable signature scheme Π_{US} [34] that is compatible with Π_{GS}.

These building blocks assume the intractability of the SXDH-problem in $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ [8, 67], presented in Definition 7, Section 2.3. Informally, this assumption implies the existence of cyclic groups $(\mathbb{G}_1, \mathbb{G}_2)$ in which the DDH problem is hard in both groups and with the property that there exists an efficiently computable bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ (Definition 5, Section 2.3). The intractability of the SXDH problem is useful in building Π_{PKUE} since the DDH assumption holds within at least one of a pair of XDH groups $(\mathbb{G}_1, \mathbb{G}_2)$, which can be used to construct a pairing-based protocol for the ElGamal-style encryption. Namely, the PKUE scheme Π_{Rise} detailed in Definition 33, Section 3.6.

The NIZK GS-proof system Π_{GS} used is a simulation-sound commit-and-prove system admitting malleable and re-randomisable proofs to facilitate the public verification of ciphertext consistency. The system works by demonstrating one of two statements is true in our construction Π_{PKUE}: either two ciphertexts $c_1 = \mathsf{UE.Enc}(pp, pk_1, m_1)$, $c_2 = \mathsf{UE.Enc}(pp, pk_2, m_2)$ are encryptions of the *same* message (meaning $m_1 = m_2$), *or*, possibly distinct messages (m_1, m_2) are signed under a signature verification key derived from the message-independent updatable signature scheme Π_{US}.

We label the aforementioned statements as S_{NY} and S_{US} respectively, and to prove that the equations in one of the statements hold, we use an OR-compilation technique to

compile these equations into a single equation which can then be proven to be satisfied, demonstrating the truth of the chosen statement. Further details on OR-compilations can be found in the works of [66, 80]. The *probabilistic message-independent* updatable signature scheme (Π_{US}) used in proving one of the two statements is based on the efficient, short signature scheme in [107] that satisfies updatable security properties such as *signature unlinkability* (see Definition 36, Section 3.6). We now present the formal definition of our concrete construction.

Definition 39 (PKUE Scheme)**.** *Given the security parameter $\lambda \in \mathbb{N}$, let the public-key updatable encryption scheme be defined by a tuple of six algorithms $\Pi_{PKUE} =$ (UE.Setup, UE.KG, UE.TG, UE.Enc, UE.Dec, UE.Upd) as follows,*

- $\mathsf{UE.Setup}(1^\lambda) \xrightarrow{\$} pp$: given the security parameter λ, the setup algorithm runs:
  - $\mathsf{RISE.Setup}(1^\lambda) \xrightarrow{\$} pp_{\mathsf{UE}} := (\mathbb{G}_1, g, q)$ for some cyclic group $\mathbb{G}_1$ with generator g of prime order q.
  - $\mathsf{GS.Setup}(gp) \xrightarrow{\$} crs_{\mathsf{GS}}$: outputs a common reference string;
  - $\mathsf{US.Setup}(1^\lambda) \xrightarrow{\$} pp_{\mathsf{US}}$: given security parameter λ, run the bilinear group generator algorithm $\mathsf{BG.Gen}(1^\lambda) \rightarrow pp_{\mathsf{US}} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g}, \hat{e}, q)$ where g is the generator of $\mathbb{G}_1$ and $\tilde{g} \xleftarrow{\$} \mathbb{G}_2$ for groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order q of asymmetric bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.
  - $\mathsf{US.KG}(pp_{\mathsf{US}}) \xrightarrow{\$} (sig_e, vk_e)$: randomly choose values $x \xleftarrow{\$} \mathbb{Z}_q, y \xleftarrow{\$} \mathbb{Z}_q^*$; set $sig_e := (x, y)$ and $vk_e := (\tilde{X}, \tilde{Y}) = (\tilde{g}^x, \tilde{g}^y)$. Only return vk_e in the system parameters. [6]

  Return public parameters $pp := (pp_{\mathsf{UE}}, crs_{\mathsf{GS}}, (pp_{\mathsf{US}}, vk_e))$.
- $\mathsf{UE.KG}(pp, e) \xrightarrow{\$} k_e$: given the public parameters, return the key pairs $k_e^i := (pk_e^i, sk_e^i)$ generated from running $\mathsf{RISE.KG}(pp_{\mathsf{UE}}, e) \xrightarrow{\$} k_e^i$ twice for $i = \{1, 2\}$. Parse $k_e^i = (pk_e^i, sk_e^i)$, let $sk_e := (sk_e^1, sk_e^2)$, $pk_e := (pk_e^1, pk_e^2)$.
- $\mathsf{UE.TG}(pp, (sk_e, k_{e+1}), (sig_e, vk_e)) \rightarrow (\Delta_{e+1}, (\delta_{e+1}, vk_{e+1}))$: compute update token $\mathsf{RISE.TG}(pp_{\mathsf{UE}}, sk_e^i, k_{e+1}^i) \rightarrow \Delta_{e+1}^i$ such that $\Delta_{e+1}^i := (sk_{e+1}^i / sk_e^i, pk_{e+1}^i)$ for $i = \{1, 2\}$ and set $\Delta_{e+1} := (\Delta_{e+1}^1, \Delta_{e+1}^2)$. Run $\mathsf{US.Next}(pp_{\mathsf{US}}, sig_e, vk_e) \rightarrow (\delta_{e+1}, vk_{e+1})$ where we use the notation δ for the signature scheme tokens.

[6] Note that the signature signing key sig_e should be different to the epoch secret key sk_e. That is, if $sk_e = x' \in \mathbb{Z}_q^*$ then we require $x' \neq \{x, y\}$.

- $\mathsf{UE.Enc}(pp, pk_e, m; r_1, r_2) \xrightarrow{\$} C_e$: compute $\mathsf{RISE.Enc}(pp_{\mathsf{UE}}, pk_e^i, m; r_i) \xrightarrow{\$} c_e^i$ for $i = \{1, 2\}$, proving one of the following two statements is true by running $\mathsf{GS.Prove}(\mathsf{OR}(S_{\mathsf{NY}}, S_{\mathsf{US}})) \xrightarrow{\$} \pi$, with common input $(pp, pk_e^1, pk_e^2, c_e^1, c_e^2)$ such that
  - $S_{\mathsf{NY}} : \exists \hat{m}, \hat{r_1}, \hat{r_2} : \mathsf{RISE.Enc}(pp_{\mathsf{UE}}, pk_e^1, \hat{m}; \hat{r_1}) = c_e^1 \wedge \mathsf{RISE.Enc}(pp_{\mathsf{UE}}, pk_e^2, \hat{m}; \hat{r_2}) = c_e^2$;
  - $S_{\mathsf{US}} : \exists \hat{m_1}, \hat{m_2}, \hat{r_1}, \hat{r_2}, \hat{\sigma} : \mathsf{RISE.Enc}(pp_{\mathsf{UE}}, pk_e^1, \hat{m_1}; \hat{r_1}) = c_e^1 \wedge \mathsf{RISE.Enc}(pp_{\mathsf{UE}}, pk_e^2, \hat{m_2}; \hat{r_2}) = c_e^2 \wedge \mathsf{US.Ver}(pp_{\mathsf{US}}, vk_e, (\hat{m_1}, \hat{m_2}), \hat{\sigma_e}) = 1$.[7]

  Output ciphertext $C_e := (c_e^1, c_e^2, \pi_e)$.

- $\mathsf{UE.Dec}(pp, sk_e, C_e) \rightarrow \{m, \perp\}$: parse $k_e := (pk_e, sk_e)$ and $C_e = (c_e^1, c_e^2, \pi)$ for some epoch e, run $\mathsf{GS.Vrfy}(vk_e, C_e) \rightarrow \{\mathsf{accept}, \mathsf{reject}\}$ w.r.t epoch public key $pk_e := (pk_e^1, pk_e^2)$. If the GS-verification algorithm outputs accept then proof π is accepted and the message $\mathsf{RISE.Dec}(pp_{\mathsf{UE}}, sk_e^1, c_e^1) = m$ is returned. Rejection $(\perp)$ is output otherwise.

- $\mathsf{UE.Upd}(pp, (\Delta_{e+1}, (\delta_{e+1}, vk_{e+1})), C_e) \xrightarrow{\$} C_{e+1}$: parse the ciphertext $C_e = (c_e^1, c_e^2, \pi)$ and update tokens $\Delta_{e+1} = (\Delta_{e+1}^1, \Delta_{e+1}^2)$, $\delta_{e+1} := (\delta_{e+1,1}, \delta_{e+1,2})$ for ciphertexts and signatures respectively. Proceed as follows,

  1. **Verification of Ciphertext:** $\mathsf{GS.Vrfy}(vk_e, C_e) \rightarrow \{\mathsf{accept}, \mathsf{reject}\}$ given the verification key the update process aborts on reject and proceeds otherwise.
  2. **Key Rotation:** ciphertext parts c_e^1, c_e^2 are updated by running $\mathsf{Rise.Upd}(pp_{\mathsf{UE}}, \Delta_{e+1}^1, c_e^1) \xrightarrow{\$} c_{e+1}^1$ and $\mathsf{Rise.Upd}(pp_{\mathsf{UE}}, \Delta_{e+1}^2, c_e^2) \xrightarrow{\$} c_{e+1}^2$. Next, the signature is updated by running $\mathsf{US.Upd}(pp_{\mathsf{US}}, \delta_{e+1}, \sigma_e) \xrightarrow{\$} \sigma_{e+1}$. Given pk_{e+1}^i, which is a component of $\Delta_{e+1}^i := (sk_{e+1}^i / sk_e^i, pk_{e+1}^i)$ for $i = \{0, 1\}$, proof π_e can be updated as a consistency proof of the new statement $\pi_{e+1} \xleftarrow{\$} \mathsf{GS.Prove}(sp, pk_{e+1}, c_{e+1}^1, c_{e+1}^2)$. The final step is due to the malleability of GS-proofs (Section 3.6).
  3. **Re-Randomisation Ciphertexts:** ciphertext parts c_{e+1}^1, c_{e+1}^2 are re-randomised. Linear malleability of GS proofs ensures that the updated proof π_{new} is valid for the re-randomised ciphertexts and signature.

[7] Note that $\hat{\sigma_e} \xleftarrow{\$} \mathsf{US.Sign}(pp_{\mathsf{US}}, sig_e, \hat{M})$ for $\hat{M} := (\hat{m_1}, \hat{m_2}) \in \mathbb{G}_1^2$ effectively produces two signatures $\hat{\sigma_e} := (\sigma_e^1, \sigma_e^2)$ following Definition 34 in Section 3.6 in which a signature on a single message m is as follows $\sigma_e := (\sigma_{e,1}, \sigma_{e,2}) = (h, h^{(x+y \cdot m)})$ for some randomly chosen $h \in \mathbb{G}_1^*$ and recall $sig_e = (x, y)$.

4. **Re-Randomise Proof:** lastly we re-randomise the proof π_{new}. Note that signature σ_{e+1} does not need to be re-randomised as it is message independent.

Return updated ciphertext $C_{e+1} := (c^1_{e+1}, c^2_{e+1}, \pi_{\mathsf{new}})$.

Correctness The construction Π_{PKUE} presented in Definition 39 satisfies correctness (Definition 29) due to the correctness of the underlying building blocks. Formally,

Theorem 1 (Correctness of PKUE Construction)**.** *Π_{PKUE} is correct assuming Π_{Rise}, Π_{GS}, and Π_{US} are correct updatable encryption, proof system, and updatable signature schemes respectively.*

Proof. First, we observe that Π_{Rise} and Π_{US} have been shown to be correct updatable encryption and signature schemes respectively (see Section 3.6). Second, by definition of the correctness of a PKUE scheme, Π_{PKUE} is correct with overwhelming probability if it outputs message m upon running $\mathsf{UE.Dec}(pp, sk_{e_{\mathsf{max}}}, C_{e_{\mathsf{max}}})$, for final epoch e_{max}, such that $C_{e_{\mathsf{max}}}$ is an updated ciphertext from $\mathsf{UE.Upd}$.

This translates to the following requirements:

- $\mathsf{GS.Vrfy}(vk_{e_{\mathsf{max}}}, C_e)$ outputs accept with respect to epoch public key $pk_{e_{\mathsf{max}}} = (pk^1_{e_{\mathsf{max}}}, pk^2_{e_{\mathsf{max}}})$. Note that the updatable signature $\sigma_{e_{\mathsf{max}}} \xleftarrow{\$} \mathsf{US.Sign}(pp, sig_{e_{\mathsf{max}}}, vk_{e_{\mathsf{max}}})$ from Π_{US} must be correct for this to hold.
- $\mathsf{RISE.Dec}(pp, sk_{e_{\mathsf{max}}}, C_{e_{\mathsf{max}}})$ outputs m given ciphertext $C_{e_{\mathsf{max}}}$ derived from $\mathsf{Rise.Upd}$ on honestly generated inputs.

Let us assume instead that $\mathsf{Rise.Dec}$ outputs $\perp$ and/or $\mathsf{GS.Vrfy}$ outputs reject. Either outcome contradicts the assumed correctness of Π_{Rise} and Π_{GS}. □

Remark 3. Construction Π_{PKUE} is designed for bi-directional key updates, however, when analysing the security of their scheme, the authors of [59] recently combined the work of [75, 104] to demonstrate that a strictly stronger notion of security for key updates is possible in the uni-directional setting. The authors of [112, 31] recently proved an equivalency of this key update notion in the no-directional key update setting. Regardless, we place precedence on modelling bi-directional key updates, despite the added complexity, to enable the modelling of stronger ciphertext confidentiality notions not yet possible in the other two cases.

Comparisons We remark the NYUE construction proposed in [80] already uses public-key techniques, however, it is still a symmetric UE scheme. We view the scheme as a suitable candidate to inspire our public-key UE construction, since the Naor-Yung paradigm is a double encryption technique that transforms a CPA-secure PKE scheme (such as the El-Gamal-based PKE in RISE) into a CCA-secure PKE scheme, and in our case, the transform can be modified to satisfy the RCCA-security of a PKUE scheme. In other words, adapting the NYUE scheme to suit the definition and needs of our PKUE framework allows us to attain the level of security we necessitated in Section 3.4 and Section 3.5.

Another key difference to the scheme in [80] compared to our construction is our explicit use of *updatable building blocks*. In particular, we use an updatable *message-independent* signature scheme, detailed in Section 3.6. We note that the authors of [80] implicitly used a modified *deterministic* updatable version of a one-time signature scheme from [79], however, they do not treat the signature security in the updatable setting. Conversely, our chosen signature scheme Π_{US} is formally proven secure in the *updatable* setting, that is, signatures from consecutive epochs are unlinkable in the same sense as ciphertext unlinkability. Moreover, Π_{US} is perfectly re-randomisable and the message-independent update property is desirable in the UE setting since updates are outsourced.

3.8 Security Analysis

In this Section, we formally state and prove that our construction Π_{PKUE} satisfies epoch confidentiality. To demonstrate provable security, we must first prove that several underlying assumptions are satisfied. Moreover, due to the length of the proof of our main theorem and the general complexity of PKUE notation and security modelling, we provide a sketch proof before the two-part proof by reduction at the end of this Section.

3.8.1 Assumptions

PKUE Building Block Recollect, in Section 3.2 we present symmetric UE definitions and assumptions that we now adapt to the public-key setting and prove in the ensuing.

First and foremost, our formal security statement Theorem 2 requires Π_{PKUE} to satisfy perfect re-encryption, simulatable reversible tokens and simulatable update-token generation in the public-key setting. We follow the notation in [80], using terms such as supp(x) which denotes the set of outcomes of positive probabilities. Observe that the following definitions do not need encryption randomness, nor are they restricted by invalid ciphertexts. To begin, Definition 40 formalises the indistinguishability between encrypted versus updated ciphertexts from the *same* epoch.

Definition 40 (Perfect Re-Encryption [80])**.** *Given a PKUE scheme where the algorithm* $\mathsf{UE.Upd}$ *is probabilistic, the update of a ciphertext is* ***perfect*** *if for all* $pp \xleftarrow{\$} \mathsf{UE.Setup}(1^\lambda)$*, for all old and new epoch key pairs* $k_e \xleftarrow{\$} \mathsf{UE.KG}(pp, e)$ *and* $k_{e+1} \xleftarrow{\$} \mathsf{UE.KG}(pp, e{+}1)$*, for all ciphertexts* $C \in \mathcal{CSP}$ *and for all tokens* $\Delta_{e+1} \leftarrow \mathsf{UE.TG}(pp, sk_e, k_{e+1})$*, we have the following:*

$$\mathsf{UE.Enc}(pp, pk_{e+1}, \mathsf{UE.Dec}(pp, sk_e, C)) \overset{Dist}{=} \mathsf{UE.Upd}(pp, \Delta_{e+1}, C).$$

Equality $\overset{\text{Dist}}{=}$ denotes the equal distribution of the left and right-hand sides of the equation. For any epoch e, $\mathsf{UE.Enc}(pp, pk_e, \perp) = \perp$ by definition.

Lemma 2. *The public-key UE scheme* Π_{PKUE} *satisfies* perfect re-encryption.

Proof. Recall, the ciphertext in Π_{PKUE} for epoch e is denoted $C_e = (c_e^1, c_e^2, \pi_e)$, and proof $\pi_e \in \mathbb{G}_T$. Further, we have an honestly generated update token for the following epoch which we defined as $\Delta_{e+1} := (\Delta, pk_{e+1}) \in (\mathbb{Z}_q^* \times \mathbb{G})$, such that $pk_{e+1} \leftarrow {pk_e}^{\Delta}$.

To perform the second step of the PKUE update ($\mathsf{UE.Upd}$), key rotation is first achieved by applying $\mathsf{Rise.Upd}$ on the ciphertext components (c_e^1, c_e^2) using tokens $\Delta = (\Delta^1, \Delta^2) = (sk_{e+1}^1/sk_e^1, sk_{e+1}^2/sk_e^2)$ and the output is (c_{e+1}^1, c_{e+1}^2). Second, the new consistency proof π_{e+1} is generated by running the $\mathsf{GS.Prove}$ algorithm from Π_{GS} on inputting of the freshly updated ciphertexts (c_{e+1}^1, c_{e+1}^2).

To perform the final two steps of the PKUE update process, the key-rotated components (c_{e+1}^1, c_{e+1}^2) and new proof π_{e+1} are perfectly re-randomised (Definition 40). The perfect property translates to the outputs looking like a freshly generated ciphertext and proof respectively, and we note that perfect re-randomisation of a proof is possible due to the linearity property of GS-proofs. In other words, the distribution of ciphertexts $((c_{e+1}^1), (c_{e+1}^2))$ is equivalent to a ciphertext produced from running the $\mathsf{Rise.Enc}$ algorithm for epoch $(e+1)$, and the distribution of the updated proof has an equal distribution to a proof generated by running $\mathsf{GS.Prove}$ for epoch $(e+1)$.

Therefore, the update procedure $\mathsf{UE.Upd}$ in Π_{PKUE} satisfies the correctness of the equation in Definition 40 and the updated ciphertext $C_{e+1} = (c^1_{e+1}, c^2_{e+1}, \pi_{e+1})$ is identically distributed to running decrypt-then-encrypt algorithms in Π_{PKUE}. Thus, Π_{PKUE} perfectly re-encrypts ciphertexts in the update procedure. □

Definition 41 and Definition 42 are related to the update token Δ, similar to those given in Section 3.2, but modified to the PKUE primitive. *Simulating reversible update tokens* is necessary for a PKUE scheme as security relies on the assumption that it is possible to simulate perfectly indistinguishable update tokens.

Definition 41 (Reversible Update Tokens)**.** *Update token Δ^{-1} is called a reverse token of Δ if for every pair of epoch keys (k^{old}, k^{new}) in key-space $\mathcal{KSP}$ such that $\Delta \in supp(\mathsf{UE.TG}(pp, sk^{old}, k^{new}))$, we have $\Delta^{-1} \in supp(\mathsf{UE.TG}(pp, sk^{new}, k^{old}))$*

Definition 42 (Simulatable Reversible Token Generation)**.** *The PKUE scheme defined in Definition 28 has simulatable token generation if the following properties hold:*

1. *There exists a PPT algorithm denoted $\mathsf{Sim.TG}(pp)$ which samples a pair of update tokens (Δ, Δ^{-1}) of the token and reverse token respectively.*
2. *For arbitrary (fixed) $k^{old} \leftarrow \mathsf{UE.KG}(pp, e_{old})$, the following token (Δ) distributions are the same:*
   - *Distribution induced by running $(\Delta, \cdot) \xleftarrow{\$} \mathsf{Sim.TG}(pp)$;*
   - *For epoch key $k^{new} \xleftarrow{\$} \mathsf{UE.KG}(pp, e_{new})$ the distribution is induced by running $(\Delta, \cdot) \xleftarrow{\$} \mathsf{UE.TG}(pp, sk^{old}, k^{new}))$.*

Lemma 3. *The public-key UE scheme Π_{PKUE} satisfies* simulatable *token and reversible token generation.*

Proof. Our construction sets an update token $\Delta_{e+1} := (\Delta', pk_{e+1}) \in (\mathbb{Z}_q^* \times \mathbb{G}_1)$ for new epoch $(e+1)$ such that $(sk_{e+1}/sk_e) := \Delta' \xleftarrow{\$} \mathbb{Z}_q^*$ and pk_{e+1} can be generated from the previous epoch as $pk_{e+1} \leftarrow {pk_e}^{\Delta'}$ since $\mathsf{UE.KG}$ derive the epoch public key as $pk_e := g^{sk_e}$. We define a reversible token $\Delta^{-1}_{e+1} := ((\Delta')^{-1}, (pk_{e+1})^{(\Delta')^{-1}})$ such that $(\Delta')^{-1} := \mathsf{invert}(\Delta')$, for some invertible function. Correctness holds as follows: $(\Delta')^{-1} := (sk_e/sk_{e+1}) \in \mathbb{Z}_q^*$ and $(pk_{e+1})^{(\Delta')^{-1}} = (g^{sk_{e+1}})^{sk_e/sk_{e+1}} = g^{sk_e} = pk_e$. Thus, Π_{PKUE} satisfies the necessary properties for simulatable token generation, as in Definitions 41, 42 since $\mathsf{UE.Upd}(pp, \Delta^{-1}_{e+1}, \mathsf{UE.Upd}(pp, \Delta_{e+1}, C_e))$ is distributed

equivalently to re-randomising the ciphertext C_e. The final statement follows from Lemma 2 of perfect re-encryption. □

Observe that the proof of simulatable update tokens implies our construction does not achieve $\mathsf{UE\text{-}IND\text{-}CCA}$ security because of the update mechanism in Π_{PKUE} since a CCA-secure UE scheme does not admit down-grading of ciphertexts.

MI-US Building Block The MI-US scheme in Definition 34 must be compatible with the NIZK GS proof system used in our construction Π_{PKUE}. That is, the signature scheme must satisfy the *structure-preserving signature* property (Definition 18, Section 2.4). Formally,

Definition 43 (Structure Preserving Signatures)**.** *Signature scheme* Π_{US} *is* structure-preserving *if messages and the signature are group elements in* $(\mathbb{G}_1, \mathbb{G}_2)$.

This holds for Π_{US} (see Definition 34, Section 3.6). Note that specific verification keys are not required for the structure-preserving property to hold, due to the GS-proof system used. Similarly to [80], a weakened form of Definition 18 can be used for construction Π_{PKUE}.

Definition 34 must satisfy a further two properties: perfect re-randomisation and simulatable token generation.

Lemma 4. *The message-independent updatable signature scheme* Π_{US} *of Definition 34, Section 3.6 satisfies perfect re-encryption and simulatable token generation.*

Proof. (Perfect Re-Encryption) In Π_{US}, we have a signature for epoch e denoted $\sigma_e = (\sigma_{e,1}, \sigma_{e,2}) = (h, h^{x+y\cdot m})$ with the update token $\Delta_{e+1} := (\Delta_{e+1,1}, \Delta_{e+1,2})$ with randomly chosen components from $(\mathbb{Z}_q^*, \mathbb{Z}_q)$, such that $vk_{e+1} = (\tilde{g}^{(x\cdot\Delta_{e+1,1}+\Delta_{e+1,2})}, \tilde{g}^{y\cdot\Delta_{e+1,1}})$. When running $\mathsf{US.Upd}$, key-rotation outputs $\sigma_{e+1} = (h^r, h^{r\cdot((x+y\cdot m)\cdot\Delta_{e+1,1}+\Delta_{e+1,2})})$ using randomness $r \xleftarrow{\$} \mathbb{Z}_q^*$. Randomisation is perfect in the sense that the output looks like a freshly generated signature since running $\mathsf{US.Sign}(pp, sig_{e+1}, m) \xrightarrow{\$} \sigma_{e+1} = (h, h^{(x+y\cdot m)})$ for signing key $sig_{e+1} = (x, y)$. Therefore, $\mathsf{US.Ver}(pp, vk_{e+1}, m, \sigma_{e+1}) = 1$ and accepts the signature since verification check $\hat{e}(h, \tilde{g}^x \cdot \tilde{g}^{y\cdot m}) = \hat{e}(h^{(x+y\cdot m)}, \tilde{g})$. Thus, the update procedure $\mathsf{US.Upd}$ in Π_{US} satisfies the correctness of the equation in Definition 40 and the updated signature σ_{e+1} is identically distributed to running $\mathsf{US.Sign}$ in Π_{US}. Thus, Π_{US} perfectly re-encrypts ciphertexts in the update procedure.

(Simulatable Token Generation) Scheme Π_{US} sets update token $\Delta_{e+1} := (\Delta_{e+1,1}, \Delta_{e+1,2})$ in $(\mathbb{Z}_q^* \times \mathbb{Z}_q)$ for new epoch $(e+1)$ such that both components were chosen randomly, the

verification key $vk_{e+1} = (\tilde{g}^{(x \cdot \Delta_{e+1,1} + \Delta_{e+1,2})}, \tilde{g}^{y \cdot \Delta_{e+1,1}})$ can be generated from the $\mathsf{US.Next}$ algorithm and $\mathsf{US.KG}$ derives the epoch verification key as $vk_e := (\tilde{g}^x, \tilde{g}^y)$. We define a reversible token $\Delta_{e+1}^{-1} := ((\Delta_{e+1,1})^{-1}, (\Delta_{e+1,2})^{-1})$ such that $\Delta_{e+1}^{-1} := \mathsf{invert}(\Delta_{e+1})$, for some invertible function. Construction Π_{US} satisfies the necessary properties for simulatable token generation, as in Definitions 41, 42 adapted to the signature setting since $\mathsf{US.Upd}(pp, \Delta_{e+1}^{-1}, \mathsf{US.Upd}(pp, \Delta_{e+1}, \sigma_e))$ is distributed equivalently to re-randomising the signature σ_e. The final statement follows from Lemma 2 of perfect re-encryption for the signature scheme setting. □

Key Privacy A PKE scheme secure against key-private, replayable CCA-attacks is modelled in the security experiment for $\mathsf{IND\text{-}IK\text{-}RCCA}$-security given in Figure 2.4, Section 2.4. Recall, modelling key privacy corresponds to an indistinguishability experiment run by a challenger whereby a polynomial-time adversary with access to a decryption oracle is given public parameters plus two public keys (pk_0, pk_1), which are inputs given by the challenger from the initialisation process. The adversary outputs two chosen messages $(m_0, m_1) \in \mathcal{MSP}$ and the challenger proceeds to encrypt message m_b using public key pk_b for chosen bit $b \in \{0, 1\}$. Security is captured by the adversary's success in the game which corresponds to distinguishing the message and public key used in encryption to produce the challenge ciphertext without knowledge of the corresponding secret keys.

$\underline{\mathsf{Exp}^{\mathsf{IND\text{-}IK\text{-}CPA}}_{\Pi_{\mathsf{Rise}},\mathcal{D}}(\mathbb{G}_1, q, x, y, t)}$

$b_{\mathcal{C}} \xleftarrow{\$} \{0,1\}$
$(u, v, w) \xleftarrow{\$} \mathbb{Z}_q^*;\ x_0 \leftarrow x; y_0 \leftarrow y, t_0 \leftarrow t;$
$x_1 \leftarrow x_0.g^u; y_1 \leftarrow (y_0)^w.g^v;\ t_1 \leftarrow t^w.x^v.y^{uw}.g^{uv}$
$pk_0 = g^{x_0}; pk_1 = g^{x_1}$
return (pk_0, pk_1)
$(m_0, m_1, s) \leftarrow \mathcal{A}(pp, pk_0, pk_1)$
$b_{\mathcal{D}} \xleftarrow{\$} \{0,1\}$
$C \xleftarrow{\$} \mathsf{Rise.Enc}(pp, pk_{b_{\mathcal{D}}}, m_{b_{\mathcal{D}}})$ `where` $C = (t_{b_{\mathcal{D}}}, y_{b_{\mathcal{D}}}.m_{b_{\mathcal{D}}})$
$b' \leftarrow \mathcal{A}(pp, C, s)$
return b'

Fig. 3.6 Experiment of between challenger (C) and distinguisher ($\mathcal{D}$) in the $\mathsf{IND\text{-}IK\text{-}CPA}$ game for Π_{Rise}, given a DDH instance in $\mathbb{G}_1$ with group generator $g \in \mathbb{G}_1$.

Remark 2 in Section 3.7 that states the authors of [12] have already proved that an El-Gamal based PKE scheme satisfies IND-IK-CPA-security assuming the hardness of DDH-problem. We note that the IND-IK-CPA security experiment follows Figure 2.4 from Section 2.4, with the omission of a decryption oracle.

In Figure 3.6, we present a distinguisher $\mathcal{D}$ using adversary $\mathcal{A}$ as a subroutine in an IND-IK-CPA game for the security of Π_{PKE} (see Definition 33), given a DDH instance $(x, y, z) \in \mathbb{G}_1$. We closely follow the distinguisher designed in [12] (see Remark 2) and use Figure 3.6 for the security analysis of Theorem 2.

The proof of security essentially shows that the advantage of a distinguisher in solving the DDH problem is greater than the advantage of adversary $\mathcal{A}$ distinguishing whether the challenge ciphertext is an encryption of the valid DDH triples (x_0, y_0, t_0) or (x_1, y_1, t_1) and so cannot determine which public key in (pk_0, pk_1) has been used for encryption either. The full reduction proof is given in [12] for the security of a key-private CPA-secure El-Gamal PKE scheme such as Π_{PKE} to the DDH problem.

3.8.2 Formal Security Proof

Theorem 2. *If the public key UE scheme* Π_{PKUE} *satisfies* perfect re-encryption *and* simulatable token generation *properties, then* $\Pi_{\mathit{PKUE}} = (\mathit{UE.Setup}, \mathit{UE.KG}, \mathit{UE.TG}, \mathit{UE.Enc}, \mathit{UE.Dec}, \mathit{UE.Upd})$ *is* UP-IND-EC-RCCA *secure assuming the SXDH problem is hard in the bilinear map* $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$.

Sketch Proof. We take a two-step modular approach in proving the Theorem 2, in which we *reduce* the proof of security from the updatable setting to the standard setting. In particular, we provide a proof reduction to the security of the underlying PKE scheme $\Pi_{\mathsf{PKE}} := (\mathsf{UE.Setup}, \mathsf{UE.KG}, \mathsf{UE.Enc}, \mathsf{UE.Dec})$ of Π_{PKUE}.

The first step of the proof is used to prove that Π_{PKE} satisfies IND-IK-RCCA security in an *isolated epoch* of Π_{PKUE}. That is, we demonstrate the security notion of key privacy against an adversary $\mathcal{A}$ admitting replayable chosen-ciphertext attacks (IND-IK-RCCA security, Section 2.4, Figure 2.4) assuming the hardness of the DDH problem in groups $\mathbb{G}_1, \mathbb{G}_2$ (SXDH, Definition 7, Section 2.3) and using the *updatable* US-EUF-CMA security of Π_{US} (Definition 36, Section 3.6).

Intuitively, to prove of IND-IK-RCCA security we use an adversary $\mathcal{A}$ to construct an adversary $\mathcal{B}$ against the PKE scheme Π_{PKE} in Definition 33 (Section 3.6), which is

interacting in an IND-IK-CPA security experiment. Remark 2 of Section 3.6 recalls the demonstration from the authors of [12] that IND-IK-CPA security in an El-Gamal PKE scheme reduces to the hardness of the DDH problem in $\mathbb{G}_1$. Therefore, we can demonstrate IND-IK-RCCA security of Π_{PKE} reduces to the proven secure IND-IK-CPA game against adversary $\mathcal{B}$ described as above, assuming the intractability of the SXDH problem.

Second, we look at proving epoch confidentiality and ciphertext unlinkability of the updatable construction Π_{PKUE} over *multiple epochs*. For clarity, we design a sequence of *hybrid games* H_l built for epochs $e_l \in \{0, \ldots, e_{\mathsf{max}} + 1\}$ of Π_{PKUE} where e_{max} is the maximum number of epochs in which an adversary $\mathcal{A}$ attacking Π_{PKUE} can query oracles (Figure 3.1). We use $\mathcal{A}$ to construct an adversary $\mathcal{B}_l$ against the standard PKE construction Π_{PKE} which is proven IND-IK-RCCA secure following the first part of our proof. Constructing adversaries in this way enables us to demonstrate the indistinguishability of games $\mathsf{H}_{l-1}, \mathsf{H}_l$ for the epochs of the Π_{PKUE} $e_l \in \{0, \ldots, e_{\mathsf{max}} + 1\}$. Thus, updatable security can be *reduced* to the security of Π_{PKE} in an isolated epoch of Π_{PKUE}.

In more detail, an adversary $\mathcal{B}_l$ *guesses* the *set* of challenge-equal epochs for $\mathcal{A}$, *embeds* their PKE challenge into epoch e_l and *simulates* the challenger in $\mathcal{A}$'s game (Figure 3.3). The effect is that game H_l behaves exactly like the game from Figure 3.3 up to epoch e_{l-1}. Further, from epoch e_l onwards $\mathcal{B}_l$ will randomly determine challenge ciphertexts. Consequently, we can show that game H_0 in challenge epoch $\tilde{e} = 0$ is run independently of challenge bit b, meaning there does not exist an adversary with a non-trivial advantage against H_0.

Moreover, a hybrid game $\mathsf{H}_{e_{\mathsf{max}}+1}$ for the final challenge epoch can be viewed as the UP-IND-EC-RCCA-game. To surmise, the above technique sees an adversary $\mathcal{B}_l$ guess a window of epochs; embeds a PKE challenge into the UE security game; simulates challenger responses of the game in Figure 3.3 for the window of epochs and the challenge epoch, then randomly determine the challenge ciphertext when simulating responses to $\mathcal{A}$ after the challenge epoch.

This method allows us to prove that $\mathcal{A}$ has a negligible advantage in distinguishing between games H_l, H_{l+1} for epochs $e_l \in \{0, \ldots, e_{\mathsf{max}} + 1\}$, thus proving that Theorem 2 holds. Observe that we achieve this final result using a *key insulation* method formalised in [80] (see Remark 1).

We present an in-depth analysis of the security of our construction. Before proceeding, we emphasise that it has already been demonstrated that GS-proofs (Π_{GS}) satisfy zero-knowledge [67] with the property that proofs are perfectly simulatable and contain a valid signature under certain conditions. Further, the structure-preserving signature scheme Π_{US} [34] satisfies US-EUF-CMA-security. The aforementioned proofs assumed hardness of the SXDH problem which translates to the hardness of DDH in groups $\mathbb{G}_1, \mathbb{G}_2$, presented in Definition 7. Finally, the underlying PKE scheme (Π_{PKE}) of Definition 33 satisfy IND-IK-CPA-security by reduction to the DDH-problem [12] in $\mathbb{G}_1$ (Remark 2). We use these facts in both stages of the proof of security following the formal statement Theorem 2.

Lemma 5. *The underlying public-key encryption scheme* $\Pi_{PKE} := (UE.Setup, UE.KG, UE.Enc, UE.Dec)$ *of construction* Π_{PKUE}*, satisfies IND-IK-RCCA security assuming the hardness of the SXDH problem in* $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$.

Proof. We model an adversary $\mathcal{A}$ (with access to $\mathcal{O}_{\mathsf{Dec}}$, Figure 3.1) against Π_{PKE} in the security game for IND-IK-RCCA by constructing an adversary $\mathcal{B}$ against Π_{PKE} in the IND-IK-CPA game. Adversary $\mathcal{B}$ acts as a challenger to adversary $\mathcal{A}$ in a specific challenge epoch $\tilde{e}$. We will demonstrate that $\mathcal{A}$'s advantage in guessing the correct bit is the same as $\mathcal{B}$'s advantage in the IND-IK-CPA game, which has been proven to reduce the DDH-problem in $\mathbb{G}_1$ (recall Remark 2.)

In more words, challenger C in the IND-IK-CPA game initialises the global parameters (**GS**) by running Rise.Setup and computes $k_0 := (pk_0, sk_0)$ such that sk_0 is sampled as $x_0 \xleftarrow{\$} \mathbb{Z}_q^*$ and $pk_0 := g^{x_0}$ for group generator $g \in \mathbb{G}_1$. The challenger generates a DDH instance (x, y, z) in $\mathbb{G}_1$ and follows Figure 3.6 (Section 3.8) in their participation in the IND-IK-CPA game; setting the challenge epoch public keys $pk_{e_0} = g^{x_{e_0}}, pk_{e_1} = g^{x_{e_1}}$, adding $\tilde{\mathcal{K}} \leftarrow \tilde{\mathcal{K}} \cup \{(pk_{e_0}, pk_{e_1})\}$ and sending (pk_{e_0}, pk_{e_1}) to $\mathcal{B}$. Adversary $\mathcal{B}$ forwards (pk_{e_0}, pk_{e_1}) to $\mathcal{A}$ and at some point in the game $\mathcal{A}$ outputs challenge messages (m_0, m_1), which $\mathcal{B}$ adds to list $\mathcal{M}^*$.

Our proof makes use of the NY-transform and uses the fact that the DDH assumption holds in $\mathbb{G}_1, \mathbb{G}_2$ (SXDH-assumption). This guarantees that GS-proofs are zero-knowledge, the underlying encryption of Π_{RISE} is secure under pk_{e_b} and US-EUF-CMA security of Π_{US} ensures an adversary $\mathcal{A}$ cannot simulate a consistent proof for a fresh message, even in the presence of simulated proofs. Thus, $\mathcal{B}$ proceeds to make the

second component of the challenge ciphertext c^2 (in Π_{PKE}) *inconsistent*[8] by always encrypting m_0. Then $\mathcal{B}$ switches to decryption of the second component c^2 in Π_{PKE} (rather than c^1 in construction Π_{PKE}) when $\mathcal{A}$ queries the decryption oracle. To succeed in the IND-IK-RCCA game, $\mathcal{A}$ would have to simulate a proof of consistency (π) of components (c^1, c^2) by breaking the US-EUF-CMA security of Π_{US}, which is a contradiction of security.

Secondly, we proceed by having $\mathcal{B}$ encrypt m_b in c^1 (using Rise.Enc), thus $\mathcal{A}$ must distinguish if $\mathcal{B}$ encrypts m_0 in c^1 (which perfectly hides the challenge bit b from $\mathcal{A}$) or $\mathcal{B}$ encrypts m_1 (this can be thought of as $\mathcal{A}$ distinguishing between a game G_0 and G_1 respectively). More precisely, $\mathcal{B}$ chooses a bit $b \in \{0, 1\}$ and proceeds to forward (m_0, m_b) to their own challenger C from the IND-IK-CPA game, receiving challenge ciphertext c^* (under chosen bit b_{C}) following Figure 3.6. Further, $\mathcal{B}$ runs $c^1 \xleftarrow{\$} \mathsf{Rise.Enc}(pp, pk_{e_b}, m_b)$, simulates the proof π of ciphertext consistency and returns $\tilde{\mathcal{C}} = (c^1, c^*, \pi)$ to $\mathcal{A}$.

Moreover, adversary $\mathcal{B}$ must handle queries that $\mathcal{A}$ makes to decryption oracle $\mathcal{O}_{\mathsf{Dec}}$ without the ability to query a decryption oracle. We note that $\mathcal{A}$ specifies decryption corresponding to a chosen epoch e_b. To start, $\mathcal{B}$ must decrypt the queried message C: first deriving the current epoch key by running $\mathsf{Rise.KG}(pp) \xrightarrow{\$} k_{\tilde{e}} := (pk_{\tilde{e}}, sk_{\tilde{e}})$, simulating a reverse or update token $\Delta_{\tilde{e}}$ (Δ^{-1} or Δ resp. using $\mathsf{Sim.TG}$) depending on whether $e_b > \tilde{e}$ or $e_b < \tilde{e}$ respectively. Next, $\mathcal{B}$ runs $\mathsf{Rise.Upd}(pp, \Delta_{\tilde{e}}, C) \xrightarrow{\$} C_{\tilde{e}}$ to update the queried ciphertext to the epoch in which they know the corresponding epoch secret-key $sk_{\tilde{e}} := (sk^1_{\tilde{e}}, sk^2_{\tilde{e}})$.

Thus, $\mathcal{B}$ can decrypt manually using $\mathsf{Rise.Dec}(pp, sk^2_{\tilde{e}}, c^2_{\tilde{e}}) \to m$. If the output $m \notin \mathcal{M}^*$, $\mathcal{B}$ responds to $\mathcal{A}$ with the decryption m. However, if the decryption message $m \in \mathcal{M}^*$ then $\mathcal{B}$ aborts their response and returns test, simulating the decryption oracle perfectly. Observe that we have used the fact that perfect token simulation and perfect re-encryption holds for Π_{PKUE} and Π_{PKE}.

[8] That is, the encryption randomness r_2 used to generate c^2 is inconsistent w.r.t the challenge ciphertext. Recall that inconsistency is directly related to the NIZK GS-proof. In the description preceding our construction Π_{PKUE} we explained the basic idea of a NIZK GS-proof following from [72, 80]. Essentially, one of two statements S_{NY}, S_{US} is true and either c_1, c_2 are encryptions of the same message ($m_1 = m_2$), implying consistency, or distinct messages ($m_1 \neq m_2$), implying inconsistency. The same idea holds for the consistency of the randomness r_1, r_2 used in the encryption of m_0, m_1 respectively.

The adversary $\mathcal{A}$ continues to query $\mathcal{O}_{\mathsf{Dec}}$ on non-challenge ciphertexts, as described above and finally guesses the game by outputting a bit b' which will also be output as $\mathcal{B}$'s guess to C. As such, the probability that $\mathcal{A}$ wins this game against IND-IK-RCCA-security (denoted by event Succ) is the same probability as adversary $\mathcal{B}$'s guess against IND-IK-CPA-security. However, $\mathcal{B}$ has a negligible advantage in winning this game (by guessing whether $b' = b_{\mathsf{C}}$). Thus,

$$\begin{aligned}|\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}IK\text{-}RCCA}}_{\mathcal{A},\Pi_{\mathsf{PKE}}}(1^\lambda) = 1] - \frac{1}{2}| &= |\Pr_{\mathcal{A},\Pi_{\mathsf{PKE}}}[\mathrm{Succ}] - \frac{1}{2}| \\ &\leq |\Pr_{\mathcal{B},\Pi_{\mathsf{PKE}}}[\mathrm{Succ}] - \frac{1}{2}| \leq \mathsf{negl}(1^\lambda).\end{aligned}$$

□

Hybrid Game Indistinguishability We have shown that Lemma 5 holds for Theorem 2. That is, Π_{PKE} satisfies IND-IK-RCCA-security. To complete the proof of Theorem 2, our approach is to construct a series of hybrid games H_l for epochs $e_l \in \{0, \ldots, e_{\mathsf{max}} + 1\}$ where e_{max} is the maximum number of epochs in which an adversary can query oracles.

Before diving into the involved key-insulation security proof method, we will provide a simplified version of the approach to proving hybrid game indistinguishability for a probabilistic UE scheme, per the authors of [80] who first introduced this method.

Recall that adversary $\mathcal{B}_l$ will embed a *static* challenge from the security experiment IND-IK-RCCA into the UP-IND-EC-RCCA game in response to adversary $\mathcal{A}$'s challenge in epoch e under one of the given public keys pk_{e_b}. Note that asides from key pk_{e_b}, other epoch keys and update tokens are unknown to $\mathcal{B}_l$ which means that responding to oracle queries from $\mathcal{A}$ calls for $\mathcal{B}_l$ to use its personal oracles. Namely, the decryption oracle in addition to simulating update tokens (Definition 42).

Given the current epoch is $\tilde{e}$, the overarching idea is that if $\mathcal{A}$ queries the token corruption oracle for some epoch e, adversary $\mathcal{B}_l$ can simulate the token Δ_e (and reverse token Δ_e^{-1}) for some epoch $e < \tilde{e}$. However, $\mathcal{B}_l$ does not know the corresponding keys for epoch e. To handle an update query from $\mathcal{A}$, adversary $\mathcal{B}_l$ can instead call their decryption oracle and then use the simulated token Δ_e to update the ciphertext to epoch e. Ciphertexts created in this way are distributed equally to fresh encryptions of a ciphertext under epoch public key pk_e assuming perfect re-encryption formalised in Definition 40. To handle a decryption query from $\mathcal{A}$, adversary $\mathcal{B}_l$ can instead use

the simulated tokens to up/downgrade the ciphertext to epoch $\tilde{e}$ and then call its decryption oracle on ciphertext $C_{\tilde{e}}$.

The brief overview above is extended in the proof of the following Lemma so that instead of adversary $\mathcal{B}_l$ guessing the challenge epoch plus epochs to the left and right in which the adversary $\mathcal{A}$ has corrupted tokens (or keys, but not both at the same time to prevent trivial attacks) and then embedding their static challenge in this guess epoch, the adversary $\mathcal{B}_l$ instead guesses a window of epochs that will contain the correct challenge epoch and embeds their static challenge there. More formally,

Lemma 6. *Hybrid games $\mathsf{H}_{l-1}, \mathsf{H}_l$ are indistinguishable.*

Proof. The indistinguishability of consecutive hybrid game pairs sees an adversary $\mathcal{A}$ against PKUE construction Π_{PKUE} in games $\mathsf{H}_{l-1}, \mathsf{H}_l$ for the UP-IND-EC-RCCA security experiment (Figure 3.3), such that $e_l \in \{0, \ldots, e_{\mathsf{max}} + 1\}$ with e_{max} being the final epoch in which $\mathcal{A}$ can make queries to oracles.

The reduction $\mathcal{B}_l$ attempts to simulate the challenger in $\mathcal{A}$'s game using a technique known as key-insulation, such that game H_l behaves exactly like the UP-IND-EC-RCCA security experiment in Figure 3.3 up to epoch e_{l-1} and the game randomly determines challenge ciphertexts after e_{l-1}. Observe that game H_0 is run independently of the challenge bit b, so $\mathcal{A}$'s advantage of winning is negligible against H_0 and game $\mathsf{H}_{e_{\mathsf{max}}+1}$ represents the UP-IND-EC-RCCA game.

The idea is for adversary $\mathcal{B}_l$ to correctly guess the boundaries of the set of challenge-equal epochs $\{\underline{e}, \ldots, \overline{e}\}$ containing the challenge-epoch, such that $\mathcal{A}$ does not corrupt the epoch keys in this set nor will they corrupt update tokens $\Delta_{\underline{e}}, \Delta_{\overline{e}+1}$. Adversary $\mathcal{B}_l$ proceeds to embed their IND-IK-RCCA challenge in epoch e_l, without knowledge of the corresponding epoch secret-key sk_{e_l}.

In more detail, $\mathcal{B}_l$ receives challenge epoch public keys $pk_{e_0} = g^{x_{e_0}}, pk_{e_1} = g^{x_{e_1}}$, for some group generator g of $\mathbb{G}_1$, from the challenger (C) in the IND-IK-RCCA-game. After updating $\tilde{\mathcal{K}} \leftarrow \tilde{\mathcal{K}} \cup \{(pk_{e_0}, pk_{e_1})\}$, they forward the challenge public keys to $\mathcal{A}$ and proceed to sample $\underline{e} \xleftarrow{\$} \{0, \ldots, e_l\}$, $\overline{e} \xleftarrow{\$} \{e_{l+1}, \ldots, e_{\mathsf{max}}\}$. Observe, reduction $\mathcal{B}_l$ has access to the PKE decryption oracle only and they will attempt to simulate responses in time-frame $\{\underline{e}, \ldots, \overline{e}\}$, without knowledge of the corresponding epoch secret keys and in turn without the capability of deriving the update tokens manually from epochs

$\{\underline{e}, \ldots, \overline{e}\}$. Instead, $\mathcal{B}_l$ must simulate the update tokens within this region (Lemma 3). The real challenge is in $\mathcal{B}_l$ simulating the remaining oracles in $\mathsf{Exp}^{\mathsf{UP\text{-}IND\text{-}EC\text{-}RCCA},b}_{\Pi_{\mathsf{PKUE}},\mathcal{A}}$ which we will describe in the ensuing paragraphs, including the period after which $\mathcal{A}$ outputs challenge messages $(m_0, m_1) \in \mathcal{M}^*$ and receives a challenge-ciphertext.

$\mathcal{O}_{\mathsf{Next}}(e)$: to start, if a ciphertext is queried in epoch $e \in \tilde{\mathcal{K}}$ then $\mathcal{B}_l$ aborts. Else, if the call is made by $\mathcal{A}$ in epoch $e < (\underline{e} - 1)$ *or* $e > \overline{e}$, then $\mathcal{B}_l$ generates a new epoch key $k_{e+1} := (pk_{e+1}, sk_{e+1})$ such that sk_{e+1} is sampled as $x_{e+1} \xleftarrow{\$} \mathbb{Z}_q^*$ and $pk_{e+1} = g^{x_{e+1}}$ *and* then generates update token $\Delta_{e+1} = (sk_{e+1}/sk_e, pk_{e+1})$. For query epoch $e = \underline{e} - 1$, $\mathcal{B}_l$ can simply set $\Delta_{e+1} = \perp$, since $\mathcal{A}$ is assumed not to query the token in this epoch, and set $k_{e+1} = g$. For query epoch $e = \overline{e}$, again $\mathcal{B}_l$ sets $\Delta_{e+1} = \perp$ and the epoch key $k_{e+1} = (g^{x_{e+1}}, x_{e+1})$ for some $x_{e+1} \xleftarrow{\$} \mathbb{Z}_q^*$. For query epochs $\underline{e} \leq e \leq \overline{e}$, $\mathcal{B}_l$ must sample a random value $\alpha_{e+1} \xleftarrow{\$} \mathbb{Z}_q^*$, setting the new epoch key as $k_{e+1} \leftarrow k_e^{\alpha_{e+1}}$ and update token as $\Delta_{e+1} = (\alpha_{e+1}, k_{e+1})$.

Finally, $\mathcal{B}_l$ must deal with the instance that $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Next}}$ in an epoch proceeding the current challenge epoch ($e > \tilde{e}$), by updating the list $\mathcal{L}$ as follows: if $e = \underline{e} - 1$ (meaning the challenge epoch $\tilde{e}$ is outside of the guessed epoch window), choose $r \xleftarrow{\$} \mathbb{Z}_q^*$ and set $C_{e+1} = (g^r, g^r \cdot m_{b'})$; if $e = \overline{e} - 1$ we sample $\alpha' := \{\alpha_{\underline{e}+1}, \ldots, \alpha_{\overline{e}}\} \xleftarrow{\$} \mathbb{Z}_q^*$ and set $C_{e+1} = (g^{\alpha'}, g \cdot m_{b'})$; if $e \geq \overline{e}$, choose C_{e+1} to be two random group elements. Otherwise, C_{e+1} is determined by regular updates. In all cases, $\mathcal{B}_l$ updates $\mathcal{L} \leftarrow \mathcal{L} \cup \{(C_{e+1}, e+1)\}$.

$\mathcal{O}_{\mathsf{Dec}}(C_{e_i})$: to start in current epoch e, if a ciphertext C is queried in some epoch $e_i \in \tilde{\mathcal{K}}$ then $\mathcal{B}_l$ aborts. Else, if the call is made by $\mathcal{A}$ for epoch $e_i \notin \{\underline{e}, \ldots, \overline{e}\}$, $\mathcal{B}_l$ can simply run $\mathsf{UE.Dec}(pp, sk_{e_i}, C_{e_i}) \rightarrow m$ by first deriving the epoch key $k_{e_i} := (pk_{e_i}, sk_{e_i})$. If however $e_i \in \{\underline{e}, \ldots, \overline{e}\}$, as $\mathcal{B}_l$ does not know the corresponding secret keys for these epochs, they must update ciphertext C_{e_i} first by iteratively running $C_{e_j} \xleftarrow{\$} \mathsf{UE.Upd}(pp, \Delta_j^{-1}, C_{e_i})$ for $j := \{i-1, \ldots, l\}$ such that $e_i > e_l$, or running $C_{e_j} \xleftarrow{\$} \mathsf{UE.Upd}(pp, \Delta_j, C_{e_i})$ for $j = \{i+1, \ldots, l\}$ such that $e_i < e_l$. This is achieved using reverse and forward tokens $(\Delta_j^{-1}, \Delta_j)$ respectively to down/up-grade the ciphertext into the previous epoch, assuming the existence of simulatable tokens (Lemma 3).

Next, $\mathcal{B}_l$ queries C on the PKE decryption oracle $\mathcal{O}_{\mathsf{Dec}}(C_{e_l})$ to obtain the message m as it does not know sk_{e_l}. Note, the queried ciphertext does not need to appear 'fresh' to $\mathcal{A}$ if downgraded by a reversible token since $\mathcal{A}$ does not see the downgraded version of the ciphertext. Provided the output $m \notin \mathcal{M}^*$, $\mathcal{B}_l$ responds to $\mathcal{A}$ with the decryption m.

$\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}(e_i)$: if the call is made by $\mathcal{A}$ for an epoch $e_i < e$, where e is the current epoch, then $\mathcal{B}_l$ can return Δ_{e_i} provided $e_i \neq \{\underline{e}, \overline{e}\}$ and returns $\perp$ otherwise. $\mathcal{O}_{\mathsf{Corrupt\text{-}Key}}(e_i)$: Similarly, if the call is made by $\mathcal{A}$ for an epoch $e_i < e$, $\mathcal{B}_l$ can return key k_{e_i} provided $e_i \notin \tilde{\mathcal{K}}$ and $e_i \notin \{\underline{e}, \ldots, \overline{e}\}$. Else, output $\perp$.

$\mathcal{O}_{\mathsf{Upd}}(C_{e_i})$: for some epoch $e_i < e$ (where e is the current epoch) adversary $\mathcal{B}_l$ must first check that $\{e_i, e\} \notin \tilde{\mathcal{K}}$ and proceed only if this is the case. Further, if $e_i \notin \{\underline{e}, \ldots, \overline{e}+1\}$ then $\mathcal{B}_l$ can simply run $C_e \xleftarrow{\$} \mathsf{UE.Upd}(pp, \Delta_e, C_{e_i})$ itself. If $e_i \in \{\underline{e}, \overline{e}+1\}$ then $\mathcal{B}_l$ can call the PKE decryption oracle $\mathcal{O}_{\mathsf{Dec}}(C_{e_i})$, obtaining a message m and proceed to encrypt for epoch e by running $C_e \xleftarrow{\$} \mathsf{UE.Enc}(pp, pk_e, m; r_1, r_2)$. Ciphertext C_e satisfies perfect re-encryption following Lemma 2, so $\mathcal{A}$ is unable to detect whether the ciphertext is a fresh or updated ciphertext.

If $\mathcal{A}$ queries an update of the challenge ciphertext $\tilde{C}$ (challenge epoch $\tilde{e}$) to current epoch e, $\mathcal{B}_l$ proceeds as follows: if $e \neq \{e_l, \overline{e}+1\}$ then simulated tokens can be used in the normal manner of simulated ciphertext updates. If $e = e_l$, $\mathcal{B}_l$ proceeds by decrypting $m \leftarrow \mathsf{UE.Dec}(pp, sk_{e-1}, \tilde{C})$ and send (m_0, m) to C and receive a challenge ciphertext $\tilde{C}_{e_l}$. This is possible as $\mathcal{B}_l$ can retrieve previous challenge ciphertexts since they're acting as the challenger to adversary $\mathcal{A}$. Finally, if $e = \overline{e}+1$ then $\mathcal{B}_l$ can decrypt $\tilde{C}$ for epoch $\overline{e}$ using the $\mathcal{O}_{\mathsf{Dec}}$ procedure described above, and run $\tilde{C}_e \xleftarrow{\$} \mathsf{UE.Enc}(pp, pk_e, m_0; r_1, r_2)$ which is compliant with both hybrid games $\mathsf{H}_{l-1}, \mathsf{H}_l$.

Assuming $\mathcal{B}_l$ correctly guesses the boundaries plus challenge-epoch e_l and following the above reduction, the view of $\mathcal{A}$ is that they are playing game H_{l-b}. Following the proof of Lemma 5 if $\mathcal{B}_l$ chooses $b = 0$ in the $\mathsf{IND\text{-}IK\text{-}RCCA}$ game, then $\mathcal{A}$ is against the security experiment for $\mathsf{UP\text{-}IND\text{-}EC\text{-}RCCA}$ up until epoch e_{l-1}, which translates to game H_l (similarly if $b = 1$ then $\mathcal{A}$ plays H_{l-1}). Thus, $\mathcal{A}$'s guess b' a guess of bit-b and so if we label the event that $b' = b$ as Succ, we have that $\frac{1}{e_{\mathsf{max}}+1} \Pr[\mathsf{Succ}]_{\mathcal{A},\Pi_{\mathsf{PKUE}}} \leq \Pr_{\mathcal{B}_l,\Pi_{\mathsf{PKE}}}[\mathsf{Succ}] \leq \mathsf{negl}(1^\lambda)$, where the second inequality derives from the proof of Lemma 5. □

3.8.3 Efficiency Considerations

Generally speaking, our construction Π_{PKUE} has a comparable cost to a standard public-key encryption scheme whose security relies on the intractability of the DDH problem. We remark that Π_{PKUE} is more expensive compared to some UE schemes from the literature. Precisely, we chose to have probabilistic updates, a property which

requires a server to produce good randomness, to model stronger corruption capabilities compared to the deterministic setting. All known UE schemes indicate that randomised updates are more expensive than deterministic updates [25]. Despite this, we believe modelling an adversary in this way is important in capturing epoch confidentiality or any other essential security notion in a PKUE security framework, especially one in which high levels of privacy are desired. Therefore, we are of the opinion that the resulting higher costs of Π_{PKUE} compared to deterministic UE schemes are outweighed by the security results produced, especially given the fact that no prior schemes satisfy the notion of epoch confidentiality.

In detail, the approximate size and cost measurements of construction Π_{PKUE} (defined in pairing groups $\mathbb{G}_1, \mathbb{G}_2$) support the decryption of ciphertexts with lengths of $(34|\mathbb{G}_1|, 34|\mathbb{G}_2|)$; the cost of ciphertext updates is approximated to be $(60\mathsf{E}, 70\mathsf{E})$ where E denotes the cost of one exponentiation and the approximate cost of ciphertext decryption is $22\mathsf{e}$ where e denotes a pairing of the groups $(\mathbb{G}_1, \mathbb{G}_2)$. Further, the signature produced by Π_{US} in our construction is always of the size $2|\mathbb{G}_1|$ regardless of the randomness used in the scheme.

Comparison The public-key UE construction Π_{PKUE} inherits the efficiency of the symmetric NYUE construction in [80] since we similarly use Π_{Rise}, and Π_{GS}. Distinctly, our scheme uses a different underlying signature scheme Π_{US} based on an updatable, multiplicative version of the Pointcheval-Sanders scheme [34, 107], compared to the authors of [80] who use a modification of the signature scheme from [79]. Assuming some security parameter $k > 2$, the size of the signature produced in Π_{PKUE} is smaller and therefore the scheme costs less than the one in [80] (that is, $2|\mathbb{G}_1|$ compared to signature size $k|\mathbb{G}_1|$ respectively). As such, the number of verification pairing computations in our construction is limited to 2, whereas the number of pairing computations in [80] is k.

Moreover, the verification keys vk_e in our construction are group elements of $\mathbb{G}_2$, whereas in [80] the verification key is a matrix distribution in $\mathbb{G}_2$. Therefore, the signature produced by [34] results in potential efficiency gains when compared to [80] in terms of both computations (number of pairings constant vs linear respectively) and communication costs, depending on the choice of parameter k in [80].

To reiterate, the computational cost of proofs and ciphertext updates in Π_{PKUE} may be expensive compared to some deterministic UE schemes from the literature: BLMR [24],

SHINE [25], and E&M [80]. However, more efficient schemes are either designed in different UE settings such as for ciphertext-dependent [52] or deterministic ciphertext updates [80, 25, 24], which are less desirable from a bandwidth and security modelling perspective respectively, or generally lower levels of security are satisfied [92].

3.9 Summary and Outlook

Our first contribution to this Chapter was re-imagining updatable encryption as a public-key primitive and modelling security for public-key standard security notions, which we deem to be necessary security requirements for all probabilistic UE schemes. Our second major contribution was to introduce a new concept of security called epoch confidentiality to tackle the problem of leakage of ciphertext age inherent in many UE schemes. Following the introduction of our new security notion, we modified an existing, symmetric UE construction to the PKUE setting with no impact on the cost/efficiency. In turn, providing a concrete scheme enabled us to show the feasibility of a PKUE construction satisfying epoch confidentiality.

In line with a general open problem inherent in a lot of UE schemes, we wish to improve the efficiency of a PKUE construction whilst maintaining the high level of security we demonstrated in this Chapter. Additionally, it is of interest to consider *post-quantum* security to construct a new and efficient PKUE scheme using, for example, the decisional learning with errors assumption.

We observe that security modelling allows bi-directional updates, meaning keys and ciphertexts can be upgraded to the next epoch or downgraded to a prior epoch. Whilst this property provides equivalent security to uni-directional schemes [75], we believe the next challenge in PKUE design is to define a *no-directional* update token [112] meaning that epoch keys cannot be derived from the tokens. We believe it is worth investigating no-directional key updates further, as the property would allow us to consider the concept of *expiry epochs*. Expiration decides how long a ciphertext can be updated to yield a new and decryptable ciphertext, and we think this concept should be examined in the PKUE security modelling context to see the effect it has on a scheme satisfying epoch confidentiality.

Chapter 4

Certificateless Public-Key Updatable Encryption

Contents

This Chapter introduces an extension of public-key updatable encryption in which we remove the need for certificates to reduce trust in entities captured in the wider public key infrastructure. Our contribution is CLUE, a certificateless PKUE primitive based on a certificateless public-key encryption primitive CL-PKE. We model standard and updatable security notions for our definition, propose a concrete construction, and demonstrate it provably satisfies the security notion we defined. Part of this Chapter, which is joint work with Elizabeth A. Quaglia, is due to be published at ITASEC 2023 [82].

4.1 Introduction

4.1.1 Motivation

In Chapter 3 we formalised a novel public-key updatable encryption primitive named PKUE (Definition 28, Section 3.6). In practice, the public-key infrastructure (PKI) in which PKUE will be used as a building block is a lot more involved than simply considering a data owner and a server. Specifically, digital certificates are typically associated with the public and secret epoch key pairs to authenticate the data owner (individual or organisation).

For context, identity-based encryption (IBE) [20] is an approach to simplifying the public key and certificate management in a PKI, in which a unique identifier is attached to the key pair. Realistically, a data owner cannot generate the secret key associated with the unique identifier. Instead, a key generation centre (KGC) computes the secret key and so the data owner must trust this entity which is in direct opposition to the goal of PKUE in which we aim to minimise the trust in external entities.

Placing trust in the KGC is futile if the entity becomes corrupted or behaves dishonestly. This is due to *key escrow* issues that can be summarised as follows. Traditional IBE schemes rely on an arrangement in which the key needed to decrypt a ciphertext is held in escrow so that under certain circumstances, an authorised third party, e.g. the *key generation centre* (KGC), can gain access to the secret keys of all users. Thus, if the KGC is corrupt, they can forge signatures on any message and decrypt the ciphertext without the consent of the users, which is a clear privacy issue. We observe that a corrupt KGC has even more power in a PKUE since the epoch secret keys are incorporated into update tokens, meaning the KGC would be able to maliciously update ciphertexts as well as learn the underlying information. Therefore, we must consider a solution to the key escrow problem with regard to PKUE, such that an *identifier is associated with an epoch* instead of a user. This is especially important given the sensitive nature of information encrypted in applications of a PKUE scheme.

Our chosen solution is a primitive introduced by [3] called certificateless public-key encryption (CL-PKE), detailed in Definition 44 of the proceeding Section. The CL-PKE primitive is an alternative primitive to PKI-supported IBE used to remove the need for certificate management and tackle the *key escrow* problem inherent in traditional identity-based encryption (IBE) schemes [111, 20, 60]. In other words, CL-PKE benefits

from the advantages of identity-based cryptography without suffering from its inherent key escrow issue [93].

At a high level, the KGC in a CL-PKE scheme generates a *partial secret key* that is distributed to the corresponding data owner who combines this cryptographic element with their own, randomly chosen secret value to generate the secret and public keys associated with their identity. In this way, the KGC does not learn the actual value of the secret key, which mitigates the key escrow problem and is crucial to the security of a CL-PKE scheme.

Observe that for tight security reductions in security modelling of CL-PKE schemes, two types of adversaries need to be captured. Namely, the traditional external adversary and an internal adversary such as a corrupted KGC who has access to the partial secret key, by definition, as well as corrupted secret keys.

We deem CL-PKE to be a suitable candidate for a revised version of the PKUE primitive if entities in the PKI cannot be trusted. Specifically, in this Chapter, we formalise a certificateless public-key UE primitive (CLUE). We do so with care considering both the security requirements of traditional CL-PKE (including inside and outside adversaries) and the intricacies of security modelling in UE arising from information inferred from tokens and epoch keys.

We stress, the CLUE primitive applies to any setting in which the KGC generating cryptographic keys and the server performing updates are separate entities that cannot be trusted or instances where individuals want to reduce trust in the KGC. Therefore, our main motivation in defining CLUE is to support long-term outsourced storage in an environment with *reduced trust*, with the intent to preserve privacy on behalf of the data owner.

Additionally, we believe in the context of real-world applications that our certificateless PKUE primitive should be considered in industry standards. In particular, the functionality of UE is included in the *payment card industry data security standard* (PCI DSS) for commercial operation [36] under the umbrella of key-rotatable encryption primitives. This standard identifies how credit card information is to be stored in an encrypted format, explicitly stating that *key rotation* should be used to move encryption from an old to a new key. Whilst removal or reduction of trust in cryptographic schemes is not explicitly covered in the PCI DSS document [36] or other standards, we think it

is of interest to extend standards to include cases where there is a desire to limit the amount of trust given to a KGC, as in our CLUE primitive.

4.1.2 Existing Work

Certificateless proxy re-encryption (CL-PRE) [122, 113, 123, 68] is a primitive closely related to CLUE due to the underlying similarities of standard PRE and UE primitives. For clarity, there are fundamental differences between the two, the most obvious being that PRE is for ciphertext decryption delegation with no time element, whereas UE is for ciphertext updates from one epoch to the next. Moreover, security modelling for CL-PRE does not focus on the same security notions as CLUE - the information an adversary can infer from the corruption of keys and the update token. Clear distinctions between the underlying PRE and UE building blocks have been researched in the works of [89, 39, 92, 80]. We discuss the similarities and likenesses between UE and PRE in Section 3.2, Table 3.1.

In a similar vein to CLUE, the recent work by [120] highlighted an interest in considering the notion of ciphertext updates in the identity-based setting. Albeit, the proposed scheme in [120] is different to CLUE in the sense that ciphertext updates are not performed using a token, but rather using the trivial method of decrypting under one key and re-encrypting with the new key. In addition, the recent works of [49, 45] further showed an interest in moving towards defining some form of an identity-based *updatable* cryptosystem.

Related in design to our proposed CLUE primitive is the identity-based encryption scheme presented by [114] which tackles the issue of user revocation in an outsourced environment. The suggested solution given by [114] is to update ciphertexts in a process of decryption followed by re-encryption to a new *time key*. Informally, public time keys are issued and are used to evolve the ciphertext to a new time. One may think that the scheme is the same as CLUE, however, it does not use update tokens and so in the scheme in [114] one needs to decrypt then re-encrypt with a new key, rather than apply an update token to a ciphertext without the need to decrypt as in UE schemes.

4.1.3 Our Contributions

Our contributions are threefold: first, we introduce and formalise CLUE, a certificateless public-key updatable encryption primitive, in Section 4.3, and present a definition of *correctness* for the CLUE primitive.

Secondly, we define and model a new security notion (CLUE-IND-RCCA security) in Section 4.4 which captures the indistinguishability of freshly generated and updated ciphertexts. In other words, this notion captures security again replayable chosen ciphertext attacks [28] (recall Definition 14 in Section 2.4 and Section 3.4). In essence, replayable ciphertext attacks allow an adversary to query the decryption of arbitrary ciphertexts, including versions of a challenge ciphertext, and it can be viewed as a relaxed version of chosen ciphertext attack security (IND-CCA). We chose to model replayable security as it is our belief that a relaxation in the security guarantees of CLUE will better reflect realistic adversaries in applications of CLUE. The literature further supports our belief, which we will discuss in more detail in Section 4.4.

Last but not least, we propose a concrete CLUE scheme in Section 4.6 which takes inspiration from the pairing-based CL-PKE scheme given in [93]. We prove that our construction satisfies CLUE-IND-RCCA security and discuss the scheme's efficiency in Section 4.7.

4.2 Chapter Preliminaries

Certificateless public-key encryption (CL-PKE) [3] is an alternative primitive to PKI-supported identity-based cryptography [20] used to remove the need for expensive *certificate management* whilst avoiding the *key escrow* problem inherent in IBE cryptosystems. In this Chapter, we extend the CL-PKE primitive to the *updatable encryption* setting. Here we present a standard CL-PKE definition.

Definition 44 (CL-PKE)**.** *Let a certificateless public-key encryption scheme be defined by a tuple of seven algorithms* $\Pi_{\mathsf{CL\text{-}PKE}} = (\mathsf{Setup}, \mathsf{Partial\text{-}SK\text{-}Extract}, \mathsf{Set\text{-}Secret\text{-}Value}, \mathsf{Set\text{-}SK}, \mathsf{Set\text{-}PK}, \mathsf{Enc}, \mathsf{Dec})$ *as follows,*

- $\mathsf{Setup}(1^\lambda) \rightarrow (pp, msk)$: given input of the security parameter λ, the setup algorithm run by the key generation centre (KGC) outputs public parameters pp and master secret key msk.

- $\mathsf{Partial\text{-}SK\text{-}Extract}(pp, msk, \mathrm{ID}_A) \rightarrow \mathrm{D}_A$: given public parameters pp, the master secret key msk and user A with identity $\mathrm{ID}_A \in \{0,1\}^*$ the partial secret-key extraction algorithm is run by the KGC and outputs the partial secret key D_A sent to the user.
- $\mathsf{Set\text{-}Secret\text{-}Value}(pp, \mathrm{ID}_A) \xrightarrow{\$} x_A$: the user A takes public parameters pp and their identity ID_A to randomly select a secret value x_A that is used to generate the full secret and public keys.
- $\mathsf{Set\text{-}SK}(pp, \mathrm{D}_A, x_A) \rightarrow sk_A$: given input pp, partial secret key D_A and secret value x_A, user A runs the set secret key algorithm to output the full secret key sk_A.
- $\mathsf{Set\text{-}PK}(pp, x_A) \rightarrow pk_A$: given input pp and secret value x_A, user A runs the set public key algorithm to output the full public key pk_A.
- $\mathsf{Enc}(pp, m, pk_A, \mathrm{ID}_A) \xrightarrow{\$} \{C, \perp\}$: given public parameters pp; message $m \in \mathcal{MSP}$; full public key pk_A and identity ID_A, the encryption algorithm outputs ciphertext $C \in \mathcal{CSP}$ or failure symbol $\perp$ if pk_A does not have the correct form.
- $\mathsf{Dec}(pp, C, sk_A) \rightarrow \{m, \perp\}$: given public parameters pp, ciphertext C and full secret key sk_A, user A runs the decryption algorithm to output message m or failure symbol $\perp$.

Correctness Given security parameter 1^λ, Definition 44 is correct if, for any valid message $m \in \mathcal{MSP}$, there exists a negligible function negl such that the following holds with overwhelming probability.

$$\Pr\left[\begin{array}{l}(pp, msk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda);\\ \mathrm{D}_A \leftarrow \mathsf{Partial\text{-}SK\text{-}Extract}(pp, msk, \mathrm{ID}_A);\\ x_A \xleftarrow{\$} \mathsf{Set\text{-}Secret\text{-}Value}(pp, \mathrm{ID}_A);\\ sk_A \leftarrow \mathsf{Set\text{-}SK}(pp, \mathrm{D}_A, x_A);\\ pk_A \leftarrow \mathsf{Set\text{-}PK}(pp, x_A);\\ \mathsf{Dec}(pp, \mathsf{Enc}(pp, m, pk_A, \mathrm{ID}_A), sk_A) = m\end{array}\right] \geq 1 - \mathsf{negl}(1^\lambda).$$

Security In this Chapter we comprehensively adapted the CL-PKE security framework to suit the requirements of updatable encryption. Nevertheless, our security framework inherits key elements of CL-PKE security modelling, including the considerations of

two types of adversaries. Namely an internal attacker such as an honest but curious key generation centre (KGC) and a traditional external adversary. Due to the "uncertified" nature of a public key generated by a user, CL-PKE security models assume an adversary is capable of replacing the users' public key [3]. Further extensions to the CL-PKE security model include an adversary's ability to extract partial private keys, secret keys, or both [126]. We defer the reader to Section 4.7 for an in-depth discussion and formal treatment of certificateless security modelling.

4.3 Certificateless Updatable Encryption

In this Section, we first introduce the syntax used for defining certificateless public-key updatable encryption (CLUE) and relevant security modelling. Second, we present the formal definition of a CLUE primitive and define correctness.

4.3.1 Syntax

We denote the current epoch as e, and use the subscript notation e_i if we define multiple epochs at once with the range of time $i = \{0, \ldots, \mathsf{max}\}$ such that e_{max} is the last epoch in the scheme. Further, (e_i, e_{i+1}) are two consecutive epochs for any $i \in \mathbb{N}$ and $\tilde{e}$ represents the challenge epoch in security games. In the context of certificateless encryption, we introduce new elements to the definition of a PKUE primitive. Specifically, in a given epoch e: ID_e is the identifier for epoch e, D_e is the partial secret key for epoch e, and x_e is the set secret value for this epoch which is renewed every period. In our construction, we use $< \cdot >$ to denote an encoding of the bracket contents to a string $\{0,1\}^*$.

4.3.2 Formal Definition of CLUE

Adding an update feature to the CL-PKE primitive presented in Definition 44, from Section 4.2 requires the addition of two algorithms: set token generation (Set-Token) and ciphertext update (Upd). Informally, the update feature of the scheme means by design it is defined in epochs of time in which the keys, token and ciphertext are associated with a given epoch in time and the ciphertext update algorithm rotates the ciphertext to encryption under a new epoch key using the token. For each epoch, a potentially untrusted key generation centre (KGC) generates a randomly chosen secret value, in line with the design of a CL-PKE scheme. The data owner proceeds to

generate their public and secret key pairs as well as an update token, using this set secret value. Encryption and decryption of information is run in a standard manner and ciphertext updates are outsourced to a host. More formally,

Definition 45 (CLUE). *Given n epochs identified by space $\mathcal{IDSP}$, security parameter $\lambda \in \mathbb{N}$, message space $\mathcal{MSP}$, and ciphertext space $\mathcal{CSP}$, a certificateless public-key updatable encryption scheme is defined by a tuple of nine algorithms* $\Pi_{\mathsf{CLUE}} = (\mathsf{Setup}, \mathsf{Partial\text{-}SK\text{-}Extract}, \mathsf{Set\text{-}Secret\text{-}Value}, \mathsf{Set\text{-}SK}, \mathsf{Set\text{-}PK}, \mathsf{Set\text{-}Token}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Upd})$ *as follows,*

- $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} (pp, msk)$: The key generation centre (KGC) takes security parameter λ as input and outputs public parameters pp and master secret key msk.
- $\mathsf{Partial\text{-}SK\text{-}Extract}(pp, msk, \mathrm{ID}_e) \rightarrow \mathrm{D}_e$: the KGC takes the public parameters pp, the master secret key msk and identity $\mathrm{ID}_e \in \mathcal{IDSP}$ for epoch e as input and outputs partial secret key D_e.[1]
- $\mathsf{Set\text{-}Secret\text{-}Value}(pp, e) \xrightarrow{\$} x_e$: the data owner takes the public parameters pp and the current epoch e that they are running the algorithm for as inputs and randomly chooses secret value x_e.
- $\mathsf{Set\text{-}SK}(pp, \mathrm{D}_e, x_e) \rightarrow sk_e$: the data owner takes the public parameters pp, partial secret key D_e and secret value x_e as inputs and computes their secret key sk_e.
- $\mathsf{Set\text{-}PK}(pp, x_e) \rightarrow pk_e$: the data owner takes the public parameters pp and secret value x_e as inputs and computes their public key pk_e.
- $\mathsf{Set\text{-}Token}(pp, sk_e, x_{e+1}) \rightarrow \Delta_{e+1}$: the data owner takes the public parameters pp, current epoch secret key sk_e, and new epoch secret value x_{e+1} as inputs and computes the update token Δ_{e+1} to the epoch $(e+1)$ which is sent to the server.
- $\mathsf{Enc}(pp, m, pk_e, \mathrm{ID}_e) \xrightarrow{\$} \{C_e, \perp\}$: the data owner takes the public parameters pp; message $m \in \mathcal{MSP}$; public key pk_e and identity ID_e as inputs and outputs the ciphertext $C_e \in \mathcal{CSP}$ or failure symbol $\perp$ if public key pk_e does not have the correct form.

[1] This algorithm is run once for each epoch and the KGC distributes the partial secret keys to the data owner in a secure manner [41].

- $\mathsf{Dec}(pp, C_e, sk_e) \rightarrow \{m, \perp\}$: the data owner takes the public parameters pp, ciphertext C and secret key sk_e as inputs and outputs the message m or failure symbol $\perp$.
- $\mathsf{Upd}(pp, C_e, \Delta_{e+1}) \rightarrow \{C_{e+1}, \perp\}$: the server takes the public parameters pp, ciphertext C_e and update token Δ_{e+1} as inputs and outputs the updates ciphertext C_{e+1} for epoch $(e+1)$ or failure symbol $\perp$.

Correctness Following the PKUE definition of correctness from Chapter 3 (Definition 29), the correctness of the CLUE primitive intuitively means that fresh and updated ciphertexts should decrypt to the corresponding plaintext given the appropriate epoch key. The formal definition of CLUE correctness follows.

Definition 46 (Correctness)**.** *Given security parameter $\lambda \in \mathbb{N}$, a certificateless updatable encryption scheme (Π_{CLUE}) formalised in Definition 45 is correct if, for any message $m \in \mathcal{MSP}$ and for any $j \in \{1, \ldots, \mathsf{max}\}$, $i \in \{0, \ldots, \mathsf{max}\}$ with $\mathsf{max} > i$, there exists a negligible function negl such that the following holds with overwhelming probability.*

$$\Pr\left[\begin{array}{l}(pp, msk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda);\\ \mathrm{D}_e \leftarrow \mathsf{Partial\text{-}SK\text{-}Extract}(pp, msk, \mathrm{ID}_e);\\ x_e \xleftarrow{\$} \mathsf{Set\text{-}Secret\text{-}Value}(pp, e);\\ sk_e \leftarrow \mathsf{Set\text{-}SK}(pp, \mathrm{D}_e, x_e);\\ pk_e \leftarrow \mathsf{Set\text{-}PK}(pp, x_e);\\ \Delta_{e_j} \leftarrow \mathsf{Set\text{-}Token}(pp, sk_e, x_{e+1});\\ C_{e_i} \xleftarrow{\$} \mathsf{Enc}(pp, m, pk_{e_i}, \mathrm{ID}_e);\\ \{C_{e_j} \leftarrow \mathsf{Upd}(pp, C_{e_{j-1}}, \Delta_{e_j}) : j \in \{i+1, \ldots, \mathsf{max}\}\};\\ \mathsf{Dec}(pp, C_{e_{\mathsf{max}}}, sk_{e_{\mathsf{max}}}) = m\end{array}\right] \geq 1 - \mathsf{negl}(1^\lambda).$$

4.4 Security modelling

In this Section, we formally model security for our CLUE primitive and define an essential privacy property capturing the indistinguishability of freshly generated and updated ciphertexts.

Defining the security of a cryptographic primitive is often a complex process. For CLUE we want to combine the approach to security taken in CL-PKE with the intricacies of UE security modelling to capture the *indistinguishability* of ciphertexts deriving from fresh encryption and updates. The notion of security we settle on is CLUE-IND-RCCA (Definition 47) and we give an intuition of this notion below.

Recollect the discussion we provided in Chapter 3, Section 3.4. Namely, the authors of [28] introduced a notion of security against a *replayable*, *adaptively* chosen-ciphertext (RCCA) attack that provides confidentiality of the underlying message. We conjecture that this property is the gold standard of security achievable, to generically satisfy both *probabilistic and deterministic* updates in any PKUE construction to satisfy ciphertext unlinkability.

Note, it is accepted in the literature that CCA-security cannot be obtained in the *probabilistic* UE setting [25, 53], as we highlighted in the previous Chapter. Whilst our CLUE primitive is designed as a *deterministic* ciphertext-independent updatable encryption scheme, we believe that relaxing adversarial corruption capabilities by enabling replayable ciphertext attacks will reflect a more realistic adversary in the outsourced environment under which CLUE is designed to be used.

In more detail, replayable CCA (RCCA) security is the same model as the CCA-notion with the following adaptation: the adversary can generate fresh ciphertexts that decrypt to the same value as a given ciphertext, however, this should not aid the adversary in winning the game. Technically speaking, the change in modelling RCCA-security for CLUE can be attributed to the behaviour of the decryption oracle: an adversary can invoke the oracle on *arbitrary* ciphertexts but the oracle will respond with test to queries that decrypt to either of the challenge messages (m_0, m_1). This includes queried ciphertexts that differ from the challenge ciphertext $\tilde{C}$ which the adversary obtains from the challenger.

Interestingly, capturing RCCA-security for CLUE resurrects an important security debate in the CL-PKE literature, highlighted by [41]: should the attacker be given as much power as possible *or* should the construction of security models reflect realistic attackers' capabilities?[2]

[2]Notably, there are two types of adversaries considered in CL-PKE schemes that must be distinguished in security modelling: an outside attacker and the honest but curious key-generation centre (KGC). Necessary to establishing an adversary in CL-PKE security experiments, we will distinguish between the two types of adversaries in the oracles that they have access to respectively in this Section.

We believe that modelling CLUE-IND-RCCA for the Definition 45 settles this debate given the prior reasoning that RCCA-security is deemed to be the *strongest* notion achievable in *any* updatable encryption scheme and yet, RCCA-security is more *realistic* in certificateless schemes. That is, compared to the seminal work of [3], heralded as the strongest notion of security achievable in CL-PKE literature, which captures the CCA-security of a CL-PKE scheme under tight adversarial restrictions. In the context of ciphertext evolution, we observe that the authors of [68] also defined the RCCA-security of a CL-PRE scheme.

The preceding discussion leads into the technical details for this Section, in which we construct an indistinguishability experiment where an adversary can access oracles and a challenger must continuously update lists to record every call to such oracles. We re-emphasise that it is essential to record and capture the *inferable* information available to an adversary during security modelling of any updatable scheme, including CLUE, which we detail in Figure 4.1 of Section 4.4.

Intuition First, we provide an intuition of the security experiment given in Figure 4.2 in which the adversary has access to oracles and the challenger records essential lists, both of which are key to capturing security given the challenging nuances of the update functionality in CLUE.

We define a notion of freshly encrypted and updated ciphertext indistinguishability for a CLUE scheme.[3] This is formalised in Definition 45 through the security experiment $\mathsf{Exp}^{\mathsf{CLUE\text{-}IND\text{-}RCCA}}_{\Pi_{\mathsf{CLUE}},\mathcal{A}}(1^\lambda)$ presented in Figure 4.2. Informally, the game is between a challenger and an adversary $\mathcal{A}$ such that the latter can query the oracles detailed in Figure 4.1. To win the experiment, adversary $\mathcal{A}$ must distinguish the underlying message of the challenge ciphertext without possession of the corresponding epoch secret key, given only access to the relevant oracles and a challenge ciphertext. Security is satisfied if the adversary's advantage in succeeding is negligible, as detailed in Definition 47.

Moving forwards, we detail the lists and oracles used in our security experiment for the definition of CLUE-IND-RCCA security. We extend the oracles inherited from Chapter 3, Figure 3.1 to additionally allow an adversary to extract partial secret keys, following CL-PKE modelling [3, 41, 93, 68]. On top of the extension to incorporate CL-PKE modelling, we discuss the corruption capabilities of the adversary in CL-PKE literature

[3]This indistinguishability notion is equivalent to the IND-ENC UE security notion defined by the authors of [92].

which necessitates security against both an outside threat as well as an honest but curious KGC [3, 41].[4]

4.4.1 Lists

To initialise the CLUE-IND-RCCA security experiment, the challenger runs $\mathsf{Init}(1^\lambda)$ which outputs the *global state* (GS) oracles have access to throughout. At the start, $\mathsf{GS} := (pp, sk_0, pk_0, \Delta_0, \mathbf{L}, 0)$ contains the public parameters pp generated by the CLUE setup algorithm; epoch secret and public keys (sk_0, pk_0) respectively; initial update token $\perp \rightarrow \Delta_0$; set $\mathbf{L} := \{\mathcal{L}, \mathcal{M}^*, \mathcal{T}, \mathcal{K}, \mathcal{C}^*\}$ containing initially empty lists that the challenger is required to maintain throughout the experiment to prevent $\mathcal{A}$ from trivially winning and setting the current epoch $0 \rightarrow e$.

List $\mathcal{L}$ is maintained to keep a log of updated versions of *honestly-generated* ciphertexts (and the corresponding epoch) that the adversary learns through calls to the update oracle. List $\mathcal{M}^*$ tracks the challenge messages the adversary sends to the challenger. Further, list $\mathcal{T}$ records the epoch(s) in which the adversary has obtained an update token and $\mathcal{K}$ tracks the epoch(s) in which the adversary has obtained an epoch secret key or epoch *partial* secret key.

The challenger records in list $\mathcal{C}$ the epochs in which an adversary obtains an updated version of the *challenge-ciphertext* through querying the ciphertext update oracle. Recall that the extended list denoted $\mathcal{C}^*$ records all of the *challenge-equal epochs* in which the adversary knows a *version* of the challenge ciphertext since there are epochs in which the adversary can infer information independently including epochs belonging to lists $\mathcal{C}, \mathcal{T}$. Challenge-equal ciphertexts are defined by a recursive predicate challenge-equal as follows,

$$\begin{aligned}\mathcal{C}^* \leftarrow \{e \in \{0, \ldots, e_{\max}\} | \mathsf{challenge\text{-}equal}(e) = \mathsf{true}\}\\ \text{and } \mathsf{true} \leftarrow \mathsf{challenge\text{-}equal}(e) \text{ iff :}\\ (e \in \mathcal{C}) \vee (\mathsf{challenge\text{-}equal}(e-1) \wedge e \in \mathcal{T})\\ \vee(\mathsf{challenge\text{-}equal}(e+1) \wedge (e+1) \in \mathcal{T}).\end{aligned}$$

[4] In-line with CL-PKE security modelling, observe that security games separate the inside and outside adversary by their access to oracles, however, the game itself and security definition unifies the two as one adversary capturing both scenarios.

4.4.2 Oracles

Figure 4.1 provides formal descriptions of the initialisation phase a challenger runs and the oracles an adversary has access to during the security experiment for Definition 47 (CLUE-IND-RCCA).

Informally, a ciphertext can be queried to decryption oracle $\mathcal{O}_{\mathsf{Dec}}$ provided it does not decrypt to one of the adversary's challenge messages $(m_0, m_1) \in \mathcal{M}^*$ nor is the output of $\mathsf{isChallenge} = \mathsf{true}$. If this is the case then the oracle returns test which is a reserved symbol different from all possible outputs of decryption [28]. Otherwise, the oracle returns a decryption of a valid ciphertext under the current epoch secret key for the epoch with identifier ID_e. We stress that returning test in the decryption oracle reveals to the adversary that the queried ciphertext decrypts to one of the challenge messages (m_0, m_1). As shown by [28], this intuitively captures the inability of an adversary to win the security game despite generating distinct ciphertexts that decrypt to the same message as the challenge ciphertext.

The adversary can update *arbitrary* ciphertexts via calls to $\mathcal{O}_{\mathsf{Upd}}$. In return, $\mathcal{A}$ receives a version of the queried ciphertext updated to the current epoch such that this *epoch* is added to list $\mathcal{C}$. Note, an RCCA-secure scheme allows any ciphertext to be updated from prior epoch e_i to the current challenge epoch e. Ciphertext C_e is computed by the update oracle iteratively running Upd from epoch e_i all the way through epochs $\{e_{i+1}, \ldots, e\}$. Note that if the queried ciphertext decrypts to one of the challenge messages $\{m_0, m_1\} \in \mathcal{M}^*$ or $\mathsf{isChallenge}(k_e, C_e) = \mathsf{true}$ for the updated ciphertext and current epoch, then the challenger adds the epoch to the list of *challenge-equal* epochs $\mathcal{C}^*$.

Querying oracle $\mathcal{O}_{\mathsf{Next}}$ in challenge epoch e results in the update of the global state to the epoch $(e + 1)$. To do so, algorithms $\mathsf{Set\text{-}Secret\text{-}Value}, \mathsf{Set\text{-}SK}$ and $\mathsf{Set\text{-}PK}$ are run to generate updated epoch keys $k_{e+1} = (sk_{e+1}, pk_{e+1})$ and the set token algorithm updates the token Δ_{e+1}. An additional step is taken given the query is in an epoch such that the adversary has corrupted the corresponding secret key. Namely, the challenge ciphertext must be updated to the next epoch using the generated update token Δ_{e+1} and the new ciphertext and epoch are added to the list $\mathcal{L}$.

Queries made to $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}$, $\mathcal{O}_{\mathsf{Corrupt\text{-}Key}}$ and $\mathcal{O}_{\mathsf{PSKE}}$ are for the corruption of an update token, epoch secret key and epoch partial secret key of the given epoch, respectively.

The restriction on all three oracles is that the adversary's query must be from an epoch preceding the current epoch e. Note that oracle $\mathcal{O}_{\mathsf{PSKE}}$ has the additional caveat that queried epoch e^* cannot belong to the list $\mathcal{K}$ to prevent the adversary from corrupting the secret key and partial secret key simultaneously, as this would enable them to compute the secret value for the epoch. If the above conditions are met then the corresponding token, secret key and partial secret key respectively for e^* are returned and the corruption epoch is added to the corresponding lists $\mathcal{T}$ and $\mathcal{K}$.

Recall that the CL-PKE *adversarial model* focuses on two types of adversaries, namely, an outside and inside (honest but curious KGC) attacker. Here we explicitly define the oracles in the set $\mathcal{O}$ that these distinct adversaries possess during the security experiment of Figure 4.2. In other words, adversary $\mathcal{A}_I$ has *no access to the master secret key*, however, they have access to all of the oracle described above and in the set $\mathcal{O} = \{\mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Next}}, \mathcal{O}_{\mathsf{Upd}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Token}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}, \mathcal{O}_{\mathsf{PSKE}}\}$.

Conversely, adversary $\mathcal{A}_{II}$ has *implicit access to a master secret key*, which means they can compute partial secret keys for their own use given the master secret key and therefore do not need access to oracle $\mathcal{O}_{\mathsf{PSKE}}$. Therefore, $\mathcal{A}_{II}$ has access to the set $\mathcal{O} = \{\mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Next}}, \mathcal{O}_{\mathsf{Upd}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Token}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}\}$.

Note that we do not allow either adversary to *adaptively* replace public keys of their choice, which deviates from CL-PKE security modelling. However, the adversary can call oracle $\mathcal{O}_{\mathsf{Next}}$ to update the public key to the next epoch. One strategy given in CL-PKE security modelling [93] is to have the challenger record ciphertexts related to decryption queries. This is not required when modelling from our CLUE security framework as the challenger in Figure 4.2 records epochs and ciphertext that would prevent such a public-key replace attack from occurring.

4.4.3 Security Game

The formal indistinguishability game given in Figure 4.2 is between a challenger and adversary $\mathcal{A} = (\mathcal{A}_I, \mathcal{A}_{II})$ defined as above. Adversary $\mathcal{A}$ must distinguish the underlying message that has been encrypted, given only access to the relevant oracles and a challenge ciphertext. Thus, possession of a challenge ciphertext should not give an adversary an advantage in determining the underlying message in the CLUE-IND-RCCA security experiment.

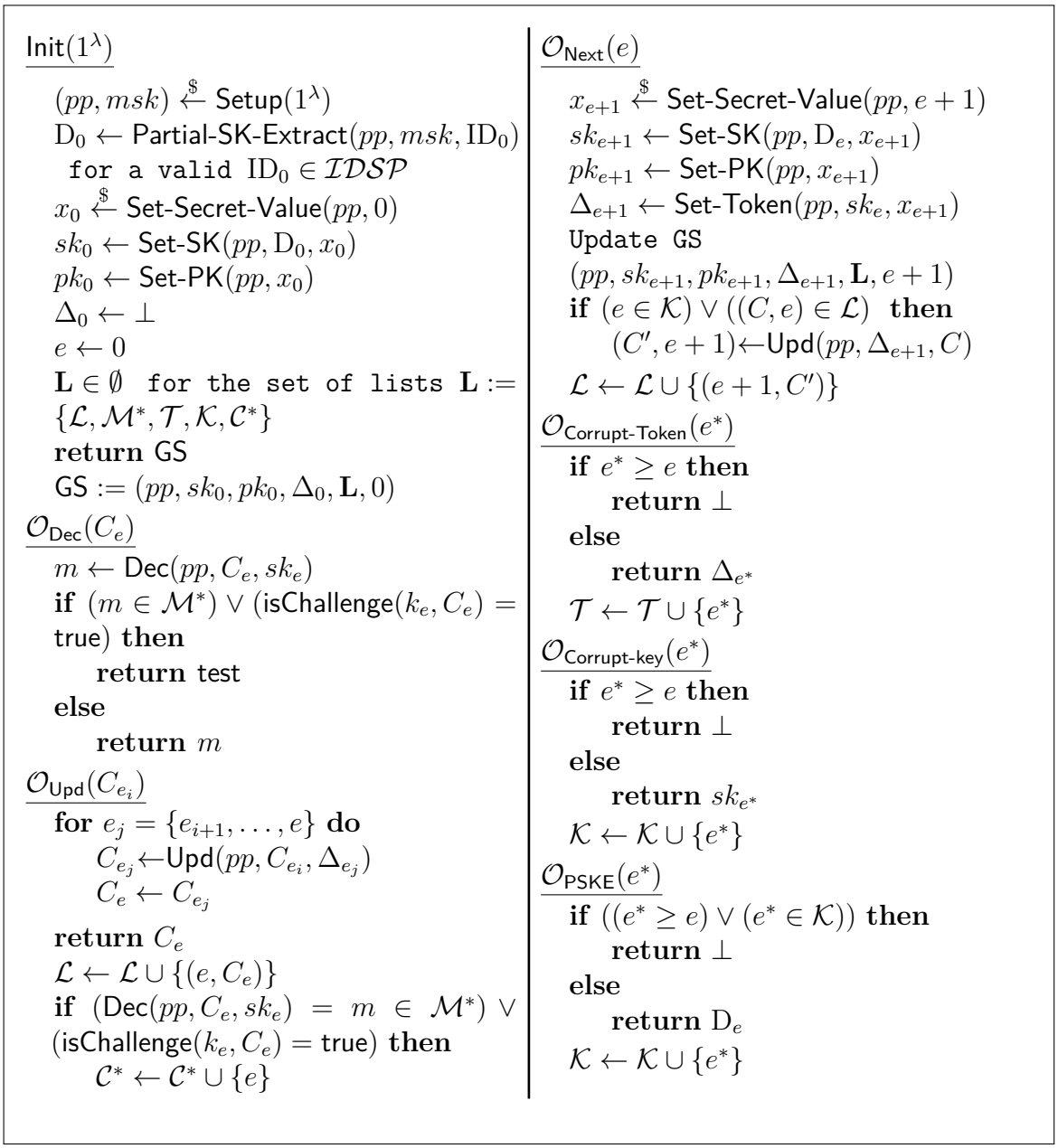

Fig. 4.1 Details of the initialisation phase run by the challenger and the oracles adversary $\mathcal{A}$ has access to during the security experiment of Definition 47.

In more words, in Figure 4.2, the initialisation process is first run by the challenger. Using the current epoch public key, the adversary proceeds to query the detailed oracles in Figure 4.1 with their relevant restrictions, outputting two challenge messages $(m_0, m_1) \in \mathcal{M}^*$ alongside some state information s. The challenger must check that

the given messages are of the same length and belong to the message space $\mathcal{MSP}$ of the scheme before proceeding, else the challenger aborts the game and returns $\perp$.

Next the challenger encrypts one of the messages m_b for chosen bit $b \in \{0,1\}$, outputting a challenge ciphertext C. The challenger sends C to $\mathcal{A}$ and updates the current epoch to the challenge epoch $\tilde{e}$ as well as the lists $\{\mathcal{M}^*, \mathcal{C}^*\}$. Given the challenge ciphertext and state information, the adversary proceeds to the second phase of calls to accessible oracles before outputting a guess bit $b' \in \{0,1\}$. The adversary succeeds in the security experiment if they satisfy the winning conditions and successfully guess the correct bit $(b' = b)$.

$\mathsf{Exp}^{\mathsf{CLUE\text{-}IND\text{-}RCCA},b}_{\Pi_{\mathsf{CLUE}},\mathcal{A}}(1^\lambda)$

`Initialise Global State`
$\mathsf{GS} \xleftarrow{\$} \mathsf{Init}(1^\lambda)$; $\mathsf{GS} = (pp, sk_0, pk_0, \Delta_0, \mathbf{L}, 0)$;
$\mathrm{D}_e \leftarrow \mathsf{Partial\text{-}SK\text{-}Extract}(pp, msk, \mathrm{ID}_e)$ `for epoch identity` $\mathrm{ID}_e \in \mathcal{IDSP}$
$x_e \xleftarrow{\$} \mathsf{Set\text{-}Secret\text{-}Value}(pp, e)$
$sk_e \leftarrow \mathsf{Set\text{-}SK}(pp, \mathrm{D}_e, x_e)$
$pk_e \leftarrow \mathsf{Set\text{-}PK}(pp, x_e)$
$(m_0, m_1, s) \leftarrow \mathcal{A}^{\mathcal{O}}(pp)$
`Some state information s`
if $|m_0| \neq |m_1| \vee \{m_0, m_1\} \notin \mathcal{MSP} \vee (m_0 = m_1)$ **then**
  return $\perp$
else
  $b \xleftarrow{\$} \{0,1\}$,
  $C \xleftarrow{\$} \mathsf{Enc}(pp, m_b, pk_e, \mathrm{ID}_e)$,
  $\mathcal{M}^* \leftarrow \mathcal{M}^* \cup (m_0, m_1)$; $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup \{e\}$; $\tilde{e} \leftarrow \{e\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}}(pp, C, s)$,
if $(\mathcal{K} \cap \mathcal{C}^* = \emptyset)$ **then**
  return b'
`Else abort.`

Fig. 4.2 The security experiment for CLUE-IND-RCCA security of a CLUE scheme, where the set of lists is $\mathbf{L} := \{\mathcal{L}, \mathcal{M}^*, \mathcal{T}, \mathcal{K}, \mathcal{C}^*\}$ is initially empty, s defines some state information output by the adversary and $\mathcal{O}$ denotes the oracles an adversary has access to, depending on whether they are a type I or type II adversary.

Definition 47 (CLUE-IND-RCCA Security)**.** *A CLUE scheme following Definition 45 is CLUE-IND-RCCA secure if an adversary $\mathcal{A}$ participating in the security game of Figure 4.2 has a negligible advantage in 1^λ, defined as follows:*

$$\mathsf{Adv}^{\mathsf{CLUE\text{-}IND\text{-}RCCA}}_{\Pi_{\mathsf{CLUE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{CLUE\text{-}IND\text{-}RCCA},0}_{\Pi_{\mathsf{CLUE}},\mathcal{A}}(1^\lambda) = 1] - \mathsf{Exp}^{\mathsf{CLUE\text{-}IND\text{-}RCCA},1}_{\Pi_{\mathsf{CLUE}},\mathcal{A}}(1^\lambda) = 1]| < \mathsf{negl}(1^\lambda).$$

Preventing Trivial Wins and Ciphertext Updates Similarly to security modelling in Chapter 3, the winning condition of the security experiment in Figure 4.2 requires the intersection of epochs contained within lists $\mathcal{K}$ and $\mathcal{C}^*$ to be empty. Recall that this condition prevents an adversary from winning trivially through inference using information obtained during oracle queries across multiple epochs.

That is, the challenge epoch of the experiment cannot belong to the set of epochs in which an update token has been learned or inferred, nor can there exist a single epoch where the adversary knows both the epoch key (public and secret key components) and the (updated) challenge-ciphertext. To see this, if the adversary $\mathcal{A}$ corrupts token Δ_{e+1} in an epoch after which $\mathcal{A}$ has obtained the challenge ciphertext $\tilde{C}$ during epoch e, either by inference or via an update, then the adversary is capable of updating the ciphertext into the next epoch $(e+1)$.

Remark 4. The security game for the CLUE-IND-RCCA notion satisfies indistinguishability for ciphertexts generated from updates, as well as from fresh encryption, despite the security game only incorporating encryption. We capture the indistinguishability of both types of ciphertext for our CLUE construction, detailed in the security analysis (Section 4.7), assuming certain properties hold. Namely, our scheme satisfies *randomness-preserving re-encryption* and *simulatable tokens* alongside standard indistinguishable security notions (see Section 4.5). This simplified method of proving fresh and updated ciphertexts are indistinguishable against an adaptive adversary follows from a core contribution given in [80]. Namely, a generic transformation demonstrating that it is sufficient to consider the underlying encryption and key-rotation capabilities of a scheme (almost) separately and therefore reduce proving to the standard-setting.

4.5 Construction Preliminaries

In this Section, we start by presenting a definition of key-homomorphic pseudorandom functions since we use this mechanism in our construction. Next, we define and present a notion of RCCA security for a CL-PKE scheme, upon which the security of our CLUE construction will rely. Further, we define the updatable security assumptions necessary for the security analysis of our CLUE scheme.

4.5.1 Key-Homomorphic Pseudorandom Functions

Key-homomorphic PRFs are useful tools for many cryptographic applications as their homomorphic property provides a simple way in which to rotate cryptographic keys that encrypt stored information, typically in an outsourced environment. KH-PRFs have previously been employed in updatable encryption [21] for this reason, and in the context of this Thesis, we make use of this primitive in Chapter 4 to construct a concrete certificateless updatable encryption scheme. The following is a formal definition of a KH-PRF.

Definition 48 (KH-PRF)**.** *Consider an efficiently computable function* $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ *such that* $(\mathcal{K}, \oplus)$ *and* $(\mathcal{Y}, \otimes)$ *are groups. Then* $(\mathsf{F}, \oplus, \otimes)$ *is a key-homomorphic PRF if the following properties hold,*

1. F is a secure pseudorandom function.
2. For every $k_1, k_2 \in \mathcal{K}$ and every $x \in \mathcal{X}$: $\mathsf{F}(k_1, x) \otimes \mathsf{F}(k_2, x) = \mathsf{F}((k_1 \oplus k_2), x)$.

4.5.2 CL-PKE Security

First, we detail the security model for a CL-PKE scheme (Definition 44, Section 4.2) secure against replayable chosen ciphertext attacks, as the proof of security for our CLUE scheme reduces to the standard (non-updatable) setting.

Replayable CCA-security for PKE schemes was a notion first introduced by [28] as a relaxed version of CCA-security since some encryption schemes are not CCA secure and yet they are sufficiently secure for most practical purposes. A CL-PKE scheme secure against replayable CCA-attacks is modelled in the security experiment for $\mathsf{CL\text{-}PKE\text{-}IND\text{-}RCCA}$-security given in Figure 4.3.

In more words, we use the Definition 49 presented below in the security analysis for the CLUE construction (Section 4.6) following the proof technique given in [80]. That is, we first prove the security of the underlying certificateless encryption scheme (in Lemma 11). Next, we prove updatable security using a so-called key-insulation method to reduce the security proof of CLUE to an isolated epoch, which is provably secure in the standard encryption setting. We will provide an intuition of the security experiment before formalising this security notion.

Just like the experiment run for updatable security in Section 4.4.3, the experiment in Figure 4.3 starts with the initialisation of public parameters and the list $\mathcal{M}^*$ which records messages output by the adversary. Note that the adversary only has access to a decryption oracle. The first phase of adversary queries to the decryption oracle follows. Next, the adversary outputs two messages $(m_0, m_1) \in \mathcal{MSP}$. The challenger proceeds to encrypt message m_b using public key pk_e given a randomly chosen bit $b \in \{0, 1\}$. Security is captured by the adversary's success in the game which corresponds to distinguishing the underlying message, without knowledge of the corresponding secret epoch key, given the challenge ciphertext. List $\mathcal{M}^*$ is kept by the challenger to record the challenge messages (m_0, m_1), and it is maintained for use when calls are made to the decryption oracle. We present the formal security definition and model of CL-PKE-IND-RCCA as follows:

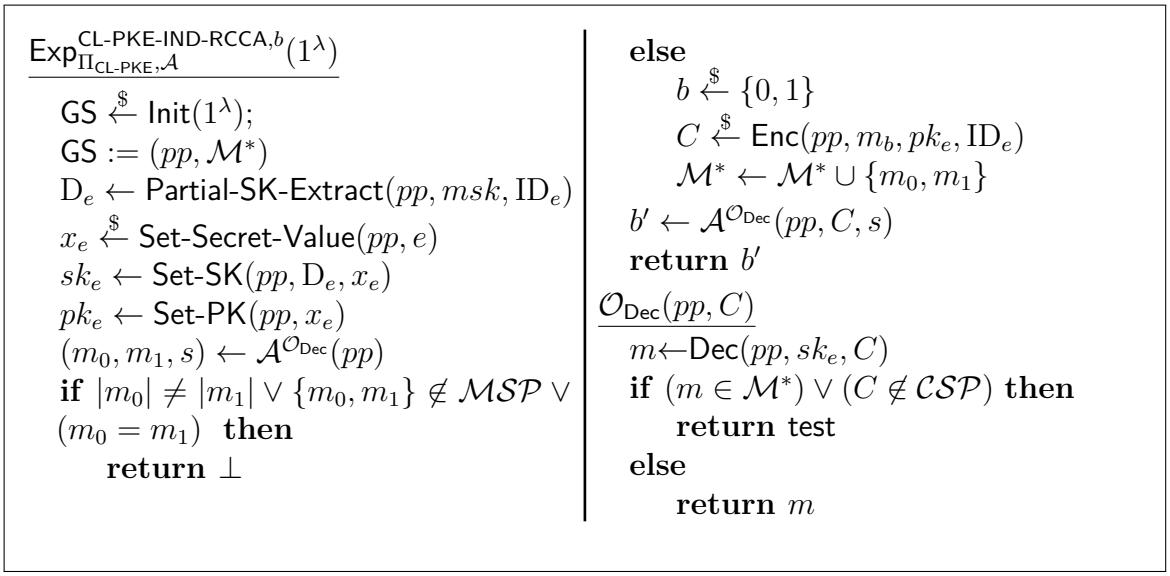

Fig. 4.3 Indistinguishability experiment for security a CL-PKE scheme $\Pi_{\mathsf{CL\text{-}PKE}}$ with s as some state information in this figure.

Definition 49 (CL-PKE-IND-RCCA Security)**.** *Scheme $\Pi_{\mathsf{CL\text{-}PKE}}$ is CL-PKE-IND-RCCA secure if for every efficient PPT adversary $\mathcal{A}$, the advantage the adversary has in the security experiments detailed in Figure 4.3 is negligible as a function of the security parameter 1^λ.*

$$\mathrm{Adv}^{\mathsf{CL\text{-}PKE\text{-}IND\text{-}RCCA}}_{\Pi_{\mathsf{CL\text{-}PKE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{CL\text{-}PKE\text{-}IND\text{-}RCCA},0}_{\Pi_{\mathsf{CL\text{-}PKE}},\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{CL\text{-}PKE\text{-}IND\text{-}RCCA},1}_{\Pi_{\mathsf{CL\text{-}PKE}},\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(1^\lambda).$$

4.5.3 Updatable Encryption Security Assumptions

Our construction is designed with *deterministic* ciphertext updates, therefore, the security of Π_{CLUE} assumes the properties of randomness-preserving re-encryption; the underlying CL-PKE scheme $\Pi_{\mathsf{CL\text{-}PKE}}$ is tidy and satisfies simulatable token generation. We utilise these properties in the security proof of Theorem 4 to argue that the indistinguishability of ciphertexts is satisfied. First, we present the notion of randomness-preserving re-encryption.

Definition 50 (Randomness-Preserving Re-Encryption)**.** *Given the updatable scheme CLUE is designed for* deterministic *updates, an updated ciphertext is* randomness-preserving *assuming CLUE encrypts with uniformly chosen randomness (*$\mathsf{Enc}(pp, m, pk_e, ID_e)$ *and* $\mathsf{Enc}(pp, m, pk_e, ID_e; r)$ *for uniformly chosen* r *are identically distributed). If for all* $(pp, msk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*; for all old and new epoch key pairs* $k_e := (pk_e, sk_e)$*,* $k_{e+1} := (pk_{e+1}, sk_{e+1})$ *generated from running the* $\mathsf{Partial\text{-}SK\text{-}Extract}$, $\mathsf{Set\text{-}Secret\text{-}Value}$, $\mathsf{Set\text{-}SK}$, $\mathsf{Set\text{-}PK}$ *algorithms in epoch* e *and* $(e+1)$ *respectively; for all valid ciphertexts* $C \in \mathcal{CSP}$ *and for all tokens* $\Delta_{e+1} \leftarrow \mathsf{Set\text{-}Token}(pp, sk_e, x_{e+1})$*, we then have the following:*

$$\mathsf{Enc}(pp, \mathsf{Dec}(pp, C_e, sk_e), pk_{e+1}, ID_e) = \mathsf{Upd}(pp, C_e, \Delta_{e+1}).$$

Tidy encryption informally means that decryption is randomness-recoverable. That is, a ciphertext is uniquely determined by the underlying message and randomness used. Therefore, a PKUE scheme is tidy if encryption and decryption algorithms are *bijections* (one-to-one correspondence for a fixed key) between the *message-randomness pairs* and *valid ciphertexts*. Formally,

Definition 51 (Randomness-Recoverable Tidy Encryption Scheme)**.** *A public-key certificateless encryption scheme is called* randomness-recoverable *if there is an associated efficient deterministic algorithm* $\mathsf{RDec}(pp, C_e, sk_e)$ *for epoch* e *such that* $\forall k_e = (pk_e, sk_e)$*,* $m, r : \mathsf{RDec}(pp, sk_e, \mathsf{Enc}(pp, pk_e, ID_e, m; r)) = (m, r)$*. We call a randomness-recoverable public-key encryption scheme* tidy *if* $\forall (pk_e, sk_e, C_e)$:

$$\mathsf{RDec}(pp, C_e, sk_e) = (m, r) \implies \mathsf{Enc}(pp, pk_e, ID_e, m; r) = C_e.$$

Note that encryption $\perp$ also yields $\perp$, therefore, Definition 51 of randomness recoverable tidy encryption is also satisfied for invalid ciphertexts [80].

In the ensuing, the *simulating reversible update tokens* assumption is used for the security analysis of Theorem 4. We use the notation $\mathrm{supp}(x)$ given in [80] to denote the set of

outcomes of positive probabilities. Second is a definition that formalises *simulatable token generation* which informally defines the indistinguishability of simulated versus honestly generated update tokens. More formally,

Assumption 2 (Reversible Update Tokens)**.** *Update token* Δ^{-1} *is called a reverse token of* Δ *if for every pair of epoch keys* $(k_{e_{old}} = (pk_{e_{old}}, sk_{e_{old}}), k_{e_{new}} = (pk_{e_{new}}, sk_{e_{new}}))$ *in key-space* $\mathcal{KSP}$ *such that* $\Delta \in supp(\mathsf{Set\text{-}Token}(pp, sk_{e_{old}}, x_{e_{new}}))$*, we have reversible token* $\Delta^{-1} \in supp(\mathsf{Set\text{-}Token}(pp, sk_{e_{new}}, x_{e_{old}}))$.

Definition 52 (Simulatable Token Generation)**.** *The CLUE scheme* Π_{CLUE} *defined in Section 4.6 has simulatable token generation if the following properties hold:*

1. There exists a PPT algorithm denoted $\mathsf{Sim\text{-}Set\text{-}Token}(pp)$ which samples a pair of update tokens (Δ, Δ^{-1}) of the token and reverse token respectively.

2. For arbitrary (fixed) $k_{e_{\text{old}}} := (pk_{e_{\text{old}}}, sk_{e_{\text{old}}})$ which is generated from running the $\mathsf{Partial\text{-}SK\text{-}Extract}, \mathsf{Set\text{-}Secret\text{-}Value}, \mathsf{Set\text{-}SK}, \mathsf{Set\text{-}PK}$ algorithms, the following token (Δ) distributions are the same:

   - Distribution induced by running $(\Delta, \cdot) \xleftarrow{\$} \mathsf{Sim\text{-}Set\text{-}Token}(pp)$;
   - For epoch key $k_{e_{\text{new}}} := (pk_{e_{\text{new}}}, sk_{e_{\text{new}}})$ the distribution is induced by running $(\Delta, \cdot) \xleftarrow{\$} \mathsf{Set\text{-}Token}(pp, sk_{e_{\text{old}}}, x_{e_{\text{new}}})$.

4.6 A Concrete CLUE Construction

In this Section, we present a concrete pairing-based CLUE scheme (Π_{CLUE}) which uses key-homomorphic pseudorandom functions to achieve the update feature. Primarily, we presented a concrete CLUE scheme to demonstrate that Π_{CLUE} is comparably efficient to other certificateless updatable PKE schemes such as the CL-PRE scheme from [68].

At a high level, Definition 45 formalised the CLUE primitive to be ciphertext-independent, therefore, the update feature is attained using tokens generated by the data owner comprised of old and new epoch keys alone. That is, tokens are generated by the data owner and contain the new epoch public key merged with a sum Δ' of the old and new secret values, which are portions of the old and new epoch secret keys respectively. The ciphertext update is achieved using only the update token (Δ_{e+1}) and the old epoch ciphertext (C_e) such that the untrusted host updating the ciphertext to the epoch $(e+1)$ learns nothing about the underlying message.

Informally, the foundations of our CLUE scheme are a concrete CL-PKE scheme and some update mechanism to enable the server to rotate keys in the update procedure without needing to decrypt the ciphertext.

Our choice of the underlying CL-PKE scheme is a modified version of the pairing-based NewFullCLE scheme proposed by [93]. Firstly, we deemed the construction from [93] to be a worthy candidate for the underlying CL-PKE scheme used in our construction due to the level of security satisfied. In particular, the authors of [93] utilise a modified *Fujisaki-Okamoto-Transform* (FOT) [57] which in essence, is a transform to lift an IND-CPA secure CL-PKE scheme to an IND-CCA secure one. In other words, an FOT is used to construct CCA-secure hybrid encryption schemes, and hybrid encryption usefully combines *symmetric* and *asymmetric* encryption to benefit from increased efficiency and more robust security guarantees respectively.

Secondly, we chose a pairing-based CL-PKE scheme for the same reasons as [93]. Namely, regarding CL-PKE literature all concrete schemes generated without pairings are supported by weaker security assumptions in the random oracle model. Whilst schemes without pairings are typically more efficient computationally speaking, the authors of [93] demonstrated that their NewFullCLE scheme attained comparable efficiency to some non-pairing schemes. We discuss efficiency in greater detail at the end of Section 4.7, Table 4.1.

Our choice for the update mechanism is a *key-homomorphic pseudorandom function* (KH-PRF) $\mathsf{F}_{\mathrm{DDH}}$. We chose this KH-PRF, not only for its desired homomorphic properties but also for its use in previous UE schemes [21, 52, 92]. To be clear, we necessitate the use of a KH-PRF building block ($\mathsf{F}_{\mathrm{DDH}}$) to support the update functionality in our CLUE construction and we note that the use of this mechanism is a key differentiator of our construction concerning that of [93].

Necessary to security, we require that the KH-PRF is proven secure in the random oracle model, assuming the hardness of the decisional Diffie-Hellman problem (Definition 4) in some finite cyclic group. We defer the reader to the formal definition of a KH-PRF and security of $\mathsf{F}_{\mathrm{DDH}}$ in Section 4.5.1. We denote the concrete KH-PRF as $\mathsf{F}_{\mathrm{DDH}} : \mathbb{Z}_q \times \mathbb{G}_2 \rightarrow \mathbb{G}_1$ whereby $\mathcal{K} = (\mathbb{Z}_q, \oplus)$ and $\mathcal{X} = (\mathbb{G}_2, \otimes)$ are additive and multiplicative groups respectively. Note that $(\mathbb{G}_1, \mathbb{G}_2)$ are cyclic (multiplicative) groups of prime order q. Evaluation of the KH-PRF is $\mathsf{F}_{\mathrm{DDH}}(k, x) = \mathcal{H}_2(x)^k$ (see Definition 48) for cryptographic hash function $\mathcal{H}_2 : \mathbb{G}_2 \rightarrow \mathbb{G}_1$, and $\mathsf{F}_{\mathrm{DDH}}(k_1 + k_2, x) = \mathsf{F}_{\mathrm{DDH}}(k_1, x) \cdot \mathsf{F}_{\mathrm{DDH}}(k_1, x)$ holds.

Now we present the formal definition of our concrete CLUE scheme and prove that the correctness property from Definition 46 is satisfied. We note that Section 4.7, proceeding this section, demonstrates our construction is provably secure in terms of the CLUE-IND-RCCA security notion.

Definition 53 (CLUE Construction)**.** *Given security parameter $\lambda \in \mathbb{N}$, n epochs, identity space $\mathcal{IDSP} = \{0,1\}^*$, message space $\mathcal{MSP} = \mathbb{G}_1$ and ciphertext space $\mathcal{CSP} = \mathbb{G}_1 \times \mathbb{G}_1$, let groups $(\mathbb{G}_1, \mathbb{G}_2)$ be cyclic (multiplicative) groups of prime order q (a 1^λ-bit prime). We define the CLUE scheme $\Pi_{CLUE} =$ (Setup, Partial-SK-Extract, Set-Secret-Value, Set-SK, Set-PK, Set-Token, Enc, Dec, Upd) as follows,*

- $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} (pp, msk)$:

  1. Given the security parameter λ as input, the setup algorithm defines a symmetric bilinear map $\hat{e} : (\mathbb{G}_1 \times \mathbb{G}_1) \to \mathbb{G}_2$ which is a Type I pairing in Definition 5, Section 2.3.
  2. Choose an arbitrary value $P \in \mathbb{G}_1$ to be the generator of $\mathbb{G}_1$ such that we have the element $g = \hat{e}(P, P) \in \mathbb{G}_2$.
  3. Given $s \xleftarrow{\$} \mathbb{Z}_q^*$ chosen uniformly at random, set the master secret key $msk = s$ and set $P' = P^s \in \mathbb{G}_1$.
  4. Choose three cryptographic hash functions used are as follows[5]:
     - $\mathcal{H}_1 : \{0,1\}^* \to \mathbb{Z}_q^*$;
     - $\mathcal{H}_2 : \mathbb{G}_2 \to \mathbb{G}_1$;
     - $\mathcal{H}_3 : \{0,1\}^* \to \mathbb{Z}_q^*$.

  Set $pp = (q, 1^\lambda, \mathbb{G}_1, \mathbb{G}_2, P, P', \hat{e}, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, n, \mathcal{MSP}, \mathcal{CSP})$ to be the public parameters and master secret key $msk = s \in \mathbb{Z}_q^*$.

- $\mathsf{Partial\text{-}SK\text{-}Extract}(pp, msk, \mathrm{ID}_e) \to \mathrm{D}_e$: Given $\mathrm{ID}_e \in \{0,1\}^*$ input as an epoch e identifier, set the partial secret key as $\mathrm{D}_e = (P^{(s+\mathcal{H}_1(\mathrm{ID}_e))^{-1}}) \in \mathbb{G}_1$. Securely send D_e to the server over a secure broadcast channel.[6]

[5]Importantly, hash function $\mathcal{H}_2$ differs from the CL-PKE scheme in [93] to suit the needs of our construction. That is, we require the homomorphic property from the KH-PRF to satisfy updatability, and $\mathcal{H}_2$ is used in the definition of $\mathsf{F}_{\mathrm{DDH}}$.

[6]Note that a server possessing partial secret key D_e and update token Δ_{e+1} is incapable of decrypting the ciphertext without corrupting either of the secret keys (sk_{e+1},sk_e), which we assume impossible in our security model.

- $\mathsf{Set\text{-}Secret\text{-}Value}(pp, e) \xrightarrow{\$} x_e \in \mathbb{Z}_q$: the data owner randomly selects set secret value x_e for epoch e.

- $\mathsf{Set\text{-}SK}(pp, \mathrm{D}_e, x_e) \rightarrow \mathrm{sk}_e$: for epoch e the data owner sets secret key $\mathrm{sk}_e := (x_e, \mathrm{D}_e) \in (\mathbb{Z}_q \times \mathbb{G}_1)$.

- $\mathsf{Set\text{-}PK}(pp, x_e) \rightarrow \mathrm{pk}_e$: for epoch e the data owner computes the public key $\mathrm{pk}_e := y_e = g^{x_e} \in \mathbb{G}_2$.

- $\mathsf{Set\text{-}Token}(pp, sk_e, x_{e+1}) \rightarrow \Delta_{e+1}$: Using $sk_e := (x_e, \mathrm{D}_e)$ and new epoch secret value x_{e+1}, we set the token $\Delta'_{e+1} := (-x_e + x_{e+1}) \in \mathbb{Z}_q$; secret key $sk_{e+1} = (x_{e+1}, \mathrm{D}_e)$ and compute $pk_{e+1} = g^{x_{e+1}}$. Set $\Delta_{e+1} := (\Delta'_{e+1}, pk_{e+1}) \in (\mathbb{Z}_q \times \mathbb{G}_2)$.

- $\mathsf{Enc}(pp, m, pk_e, \mathrm{ID}_e) \xrightarrow{\$} \{C_e, \perp\}$: the data owner performs the following three steps.

  1. Select uniform randomness $\sigma \xleftarrow{\$} \mathbb{Z}_q^*$.
  2. Set $r = \mathcal{H}_3(< m^\sigma || pk_e || \mathrm{ID}_e >) \in \mathbb{Z}_q^*$.[7]
  3. Set $C_e = (c_e^1, c_e^2) = (P^{r\mathcal{H}_1(\mathrm{ID}_e)} \cdot (P')^r, m^\sigma \cdot \mathsf{F}_{\mathrm{DDH}}(x_e, g^r))$.

- $\mathsf{Dec}(pp, C_e, \mathrm{sk}_e) \rightarrow \{m, \perp\}$: parse ciphertext $C_e = (c_e^1, c_e^2)$ and secret key $sk_e = (x_e, \mathrm{D}_e)$ and go through the following steps,

  1. Compute $\omega = \hat{e}(c_e^1, \mathrm{D}_e)$ such that

$$\begin{aligned}\omega = \hat{e}(c_e^1, \mathrm{D}_e) &= \hat{e}(P^{r\mathcal{H}_1(\mathrm{ID}_e)} \cdot P^{rs}, P^{(s+\mathcal{H}_1(\mathrm{ID}_e))^{-1}}) \\ &= \hat{e}(P^{r(\mathcal{H}_1(\mathrm{ID}_e)+s)}, P^{(s+\mathcal{H}_1(\mathrm{ID}_e))^{-1}}) \\ &\overset{(*)}{=} \hat{e}(P, P)^{r(\mathcal{H}_1(\mathrm{ID}_e)+s)\cdot(\mathcal{H}_1(\mathrm{ID}_e)+s)^{-1}} = g^r,\end{aligned}$$

     where equality $(*)$ holds due to the bilinearity property of $\hat{e}$ (Definition 5, Section 2.3).

  2. In order for the data owner to compute r in the next step, m^σ needs to be determined. Given step 1 in which it is determined that $\omega = g^r$, the following can be computed $c_e^2 \cdot \mathsf{F}_{\mathrm{DDH}}(-x_e, \omega) = m^\sigma \in \mathbb{G}_1$. Correctness holds as follows,[8]

[7] Let $< \cdot >$ denote an encoding of the bracket contents to a string $\{0,1\}^*$.

[8] To see the penultimate equation differently, given the definition of the KH-PRF: $\mathsf{F}_{\mathrm{DDH}}(x_e - x_e, g^r) = \mathcal{H}_2(g^r)^{x_e - x_e} = \mathcal{H}_2(g^r)^0 = \mathsf{id}_{\mathbb{G}_1}$.

$$c_e^2 \cdot \mathsf{F}_{\mathrm{DDH}}(-x_e, \omega) = m^\sigma \cdot \mathsf{F}_{\mathrm{DDH}}(x_e, g^r) \cdot \mathsf{F}_{\mathrm{DDH}}(-x_e, \omega)$$
$$= m^\sigma \cdot \mathsf{F}_{\mathrm{DDH}}(x_e - x_e, g^r) = m^\sigma.$$

Note that the *data owner* randomly chose σ during encryption, so knowledge of this enables the computation of the message $(m^\sigma)^{-\sigma} := m$.[9]

3. Use the epoch secret-key and public parameters (sk_e, pp) in addition to the previous two steps to compute $r = \mathcal{H}_3(< m^\sigma || pk_e || \mathrm{ID}_e >) \in \mathbb{Z}_q^*$. Message m is accepted *iff* $c_e^1 = (P^{\mathcal{H}_1(\mathrm{ID}_e)} \cdot P')^r$ from the computed r value, else failure $(\perp)$ is output.

- $\mathsf{Upd}(pp, C_e, \Delta_{e+1}) \rightarrow \{C_{e+1}, \perp\}$: recall the update token and ciphertext $\Delta_{e+1} := (\Delta'_{e+1}, pk_{e+1})$, $C_e = (c_e^1, c_e^2)$ respectively. The server must perform the following steps:

1. Check $pk_{e+1}^q = 1_{\mathbb{G}_2}$. Abort the update and output failure symbol $\perp$ if this does not hold. Note, validity holds with an honestly generated epoch public key: $pk_{e+1}^q = (g^{x_{e+1}})^q = (g^q)^{x_{e+1}} = (1_{\mathbb{G}_2})^{x_{e+1}}$.
2. Compute $\omega = \hat{e}(c_e^1, \mathrm{D}_e) = g^r \in \mathbb{G}_2$. See step 1 of the decryption algorithm for correctness. Set $c_{e+1}^1 := c_e^1$.
3. Use step 2 and the given public key pk_{e+1} to compute $c_{e+1}^2 := c_e^2 \cdot \mathsf{F}_{\mathrm{DDH}}(\Delta'_{e+1}, \omega)$ and output $C_{e+1} = (c_{e+1}^1, c_{e+1}^2)$. Consistency is upheld using ω as follows:

$$\begin{aligned} c_{e+1}^2 &= c_e^2 \cdot \mathsf{F}_{\mathrm{DDH}}(\Delta'_{e+1}, \omega) \\ &= m^\sigma \cdot \mathsf{F}_{\mathrm{DDH}}(x_e, g^r) \cdot \mathsf{F}_{\mathrm{DDH}}(-x_e + x_{e+1}, \omega) \\ &= m^\sigma \cdot \mathsf{F}_{\mathrm{DDH}}(x_e - x_e + x_{e+1}, g^r) \\ &= m^\sigma \cdot \mathsf{F}_{\mathrm{DDH}}(x_{e+1}, g^r). \end{aligned}$$

Remark 5. Note that only the second component (c_e^2) of the ciphertext gets updated and the first component (c_e^1) remains the same, in line with previous identity-based approaches used in CL-PKE literature [125, 18]. The first component contains a secure signature of an identifier for the epoch in which the ciphertext was created and is crucial for computing the value ω in the decryption and update process.[10]

[9] The technique of using the corresponding randomness for a given epoch to decrypt the ciphertext is utilised in various UE schemes including [21, 92].

[10] For the sake of continuity within this Thesis, we note that the pairing notation used for Definition 53 is multiplicative, following Definition 5 from Section 2.3. This differs from the additive notation used in the published work [82] on which this Chapter is based.

The fact that c_e^1 remains unchanged is the reason why we do not achieve the stronger notion of full ciphertext unlinkability (IND-UPD), and instead, our construction only achieves encrypted and updated ciphertext indistinguishability (IND-ENC).

Correctness In the following, we show that our construction Π_{CLUE} presented in Definition 53 satisfies correctness (Definition 46).

Theorem 3 (Correctness of CLUE Construction)**.** *Construction Π_{CLUE} from Definition 53 is correct assuming the KH-PRF F_{DDH} satisfies Definition 48.*

Proof. Intuitively Π_{CLUE} is correct, following Definition 46 from Section 4.3, if the decryption algorithm Dec outputs message m in the final epoch e_{max} given honest inputs $(pp, sk_{e_{\mathsf{max}}})$ and correctly updated ciphertext $C_{e_{\mathsf{max}}}$. In other words, assuming that the final ciphertext has been correctly updated to encryption of m under honestly generated epoch public key $pk_{e_{\mathsf{max}}}$, it follows that decryption of $C_{e_{\mathsf{max}}}$ will return m with overwhelming probability.

Observe step 3 of Upd which demonstrates the consistency of the updated ciphertext to valid encryption under the new epoch public key. Consistency is supported by the homomorphic property of $\mathsf{F}_{\mathsf{DDH}}$, under the assumption that the KH-PRF satisfies Definition 48. Given the aforementioned ciphertext update consistency, step 2 of Dec illustrates the correctness of decryption assuming honest inputs. □

4.7 Security Analysis

In this Section, we analyse the security of our construction Π_{CLUE}. Recall from Section 4.4, Remark 4 - in proving CLUE-IND-RCCA security of Π_{CLUE} to achieve ciphertext *indistinguishability*, we assume that Π_{CLUE} satisfies the *randomness-preserving re-encryption* (Definition 50); simulatable- token generation (Definition 52) and the underlying CL-PKE scheme $\Pi_{\mathsf{CL\text{-}PKE}}$ satisfies *tidy encryption* (defined in Section 4.2). First, in Section 4.7.1 we prove several statements hold, proceeded by a formal statement and analysis of security in Section 4.7.2.

4.7.1 Assumptions

To start, we demonstrate the security of the KH-PRF used in our construction Π_{CLUE}. Formally,

Lemma 7. *Given the KH-PRF used in* Π_{CLUE} *defined as* $\mathsf{F}_{DDH} : \mathbb{Z}_q \times \mathbb{G}_2 \rightarrow \mathbb{G}_1$ *with* $\mathcal{K} = (\mathbb{Z}_q, \oplus)$, $\mathcal{X} = (\mathbb{G}_2, \otimes)$ *the additive and multiplicative groups of prime order* q *respectively such that* $(\mathbb{G}_1, \mathbb{G}_2)$ *are cyclic (multiplicative) groups of prime order* q*, evaluation of the KH-PRF is* $\mathsf{F}_{DDH}(k, x) = \mathcal{H}_2(x)^k$*. Further,* $\mathsf{F}_{DDH}(k_1 + k_2, x) = \mathsf{F}_{DDH}(k_1, x) \cdot \mathsf{F}_{DDH}(k_2, x)$ *(that is,* F_{DDH} *satisfies Definition 48, Section 4.5.1). Then* F_{DDH} *is a secure KH-PRF in the random oracle model assuming the hardness of the decisional Diffie-Hellman problem in* $\mathbb{G}_1$*.*

Informally, Naor-Reingold [101] first demonstrated security in the random oracle model of a so-called fully-fledged (large domain) KH-PRF. Further, the authors of [47] proposed a new framework to construct any bounded (small domain) KH-PRFs, like the one used in our construction for Π_{CLUE}, to a large domain PRF as in [101]. The authors of [47] were able to demonstrate the security of the bounded KH-PRF by reduction to the standard DDH problem with an almost tight security proof by exploiting the algebraic properties of underlying number theoretic assumptions. In the case of our KH-PRF, we assume the hardness of the DDH problem (Definition 4, Section 2.3). We defer the reader to [47] for more detail. More formally,

Sketch Proof. We denote the Naor-Reingold [101] PRF as $\mathsf{NR}_{\mathrm{n}} : \mathcal{K}_n \times \{0,1\}^n \rightarrow \mathbb{G}_1$ in line with the notation of [47], for some $n \in \mathbb{N}$ whereby $\{0,1\}^n$ is a large domain, elements from key space $\mathcal{K}_n$ are randomly chosen from $\mathbb{Z}_q$, and $\mathbb{G}_1$ a cyclic group of prime order q with generator g. Evaluation is as follows: $\mathsf{NR}_n(k, x) = g^{a\Pi_{j=0}^{n-1} s_j^{x_j}}$, for any $x \in \{0,1\}^n$, and $(a, s_0, \ldots, s_{n-1}) \xleftarrow{\$} \mathcal{K}_n$. The framework of [47] takes multiple steps.

- Take a small domain of size l ($\mathsf{NR}_{log(l)}$), we can adapt this PRF into an l-bounded PRF with large input domain $\mathbb{G}_2$, that is, $\mathsf{F}_l : \mathcal{K}_n \times \mathbb{G}_2 \rightarrow \mathbb{G}_1$ such that $\mathsf{F}_l(k, x) = g^{a\Pi_{j=0}^{log(l)-1}(s_j - x^{2^j})}$, for any $x \in \mathbb{G}_2$.

- The security of F_l tightly reduces to that of $\mathsf{NR}_{log(l)}$ by looking at the exponent, following the reduction technique of [47]. Essentially, F_l can be written in terms of $\mathsf{NR}_{log(l)}$ which itself can be replaced by a random function (recall the latter is secure assuming the intractability of the DDH problem), provided $l \leq \mathsf{poly}(\lambda)$, whereby λ is the security parameter. Thus F_l can be written in terms of a random function.

- The final step is to demonstrate that F_l can be embedded into our KH-PRF $\mathsf{F}_{\mathrm{DDH}}$ and which itself can be replaced by a random function. Choosing $t = \omega(log(\lambda))$ and given $l \leq \mathsf{poly}(\lambda)$, we can write our KH-PRF as follows: $\mathsf{F}_{\mathrm{DDH}} =$

$\left[g^{a\Pi_{j=0}^{log(l)-1}(s_j+x^{2^j})}\right]^{\Pi_{j=log(l)}^{t-1}(s_j+x^{2^j})} = (\mathsf{F}_l(k,x))^{\Pi_{j=log(l)}^{t-1}(s_j+x^{2^j})}$. The remainder of the proof is to show that the exponent $\Pi_{j=log(l)}^{t-1}(s_j + x^{2^j})$ only produces a negligible error, and as a consequence, security of $\mathsf{F}_{\mathrm{DDH}}$ in Lemma 7 reduces to the security of large domain PRF $\mathsf{NR}_{log(l)}$ which is proven to be secure.

Now we present the proofs that essential properties related to the update functionality of Π_{CLUE} are satisfied.

Lemma 8. *The scheme Π_{CLUE} satisfies* randomness preserving re-encryption *given in Definition 50.*

Proof. In honestly running the construction Π_{CLUE} we have public parameters pp; master secret key s; epoch public and secret keys $k_e = (pk_e, sk_e) := (g^{x_e}, (x_e, \mathrm{D}_e))$ for epoch e with partial secret key D_e (for epoch identifier ID_e); update token $\Delta_{\mathsf{e+1}} := (\Delta'_{e+1}, pk_{\mathsf{e+1}}) \in (\mathbb{Z}_q^* \times \mathbb{G}_1)$ such that $\Delta'_{e+1} := (-x_e + x_{e+1})$ and $pk_{e+1} = g^{x_{e+1}}$. Definition 50 is satisfied as follows:

$$\begin{aligned}
&\mathsf{Enc}(pp, \mathsf{Dec}(pp, C_e, sk_e), pk_{e+1}, \mathrm{ID}_e) = \mathsf{Enc}(pp, m, pk_{e+1}, \mathrm{ID}_e)\\
&= C_{e+1} := (P^{r\mathcal{H}_1(\mathrm{ID}_e)} \cdot (P')^r, m^\sigma \cdot \mathsf{F}_{\mathrm{DDH}}(x_{e+1}, g^r))\\
&= (c^1_{e+1}, c^2_{e+1}) = (c^1_e, m^\sigma \cdot \mathsf{F}_{\mathrm{DDH}}(x_e, g^r) \cdot \mathsf{F}_{\mathrm{DDH}}(\Delta'_{e+1}, g^r))\\
&= (c^1_e, m^\sigma \cdot \mathsf{F}_{\mathrm{DDH}}(x_{e+1}, g^r)).
\end{aligned}$$

Where the final equality is the output of $\mathsf{Upd}(pp, C_e, \Delta_{e+1})$. The correctness of the scheme is further detailed in the construction itself. □

Due to the deterministic nature of ciphertext updates in Π_{CLUE}, the scheme must satisfy *tidy encryption*. We note that the construction presented in Section 4.6 demonstrates that randomness-recoverable tidy encryption holds in the sense of Definition 51 using uniformly chosen $\sigma \xleftarrow{\$} \mathbb{Z}_q^*$ randomness detailed in the construction.

Lemma 9. *The CLUE scheme Π_{CLUE} satisfies the and* tidy encryption *and* randomness preserving updates *properties given in Definition 51 and Definition 25 respectively.*

Proof. Firstly, CLUE encrypts with uniformly chosen randomness $\sigma \in \mathbb{Z}_q^*$, so the outputs of $\mathsf{Enc}(pp, pk_e, m, \mathrm{ID}_e)$ and $\mathsf{Enc}(pp, pk_e, m, \mathrm{ID}_e; \sigma)$ are identically distributed. Thus, there exists an efficient deterministic algorithm $\mathsf{RDec}(pp, sk_e, C_e) = (m, \sigma)$ such that construction Π_{CLUE} satisfies the notion of *tidy* encryption:

$$\mathsf{RDec}(pp, C_e, sk_e) = (m, \sigma) \implies \mathsf{Enc}(pp, pk_{e+1}, m, \mathrm{ID}_e; \sigma) = C_{e+1}.$$

Given this fact, $\forall (pp, msk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, $\forall (pk_e, pk_{e+1}), (sk_e, sk_{e+1})$ generated from $\mathsf{Set\text{-}SK}, \mathsf{Set\text{-}PK}$ respectively, $\forall \Delta_{e+1} \leftarrow \mathsf{Set\text{-}Token}(pp, sk_e, x_{e+1})$ and all valid ciphertexts $C \in \mathcal{CSP}$ encrypted under pk_e, we have:

$$\mathsf{Enc}(pp, pk_{e+1}, \mathrm{ID}_e, \mathsf{RDec}(pp, sk_e, C_e)) = \mathsf{Enc}(pp, pk_{e+1}, \mathrm{ID}_e, (m, \sigma)) = C_{e+1},$$

and,

$$\mathsf{Enc}(pp, pk_{e+1}, \mathrm{ID}_e, \mathsf{Dec}(pp, sk_e, C_e)) = \mathsf{Enc}(pp, pk_{e+1}, \mathrm{ID}_e, m) = C_{e+1} \overset{\text{dist}}{=} \mathsf{Upd}(pp, C_e, \Delta_{e+1}).$$

The first equation is for randomness recoverable, tidy encryption and the second equation demonstrates that randomness-preserving updates are satisfied. Note that the ciphertexts obtained from both equations are identically distributed. □

The following statement focuses on the update token Δ generated from running $\mathsf{Set\text{-}Token}$ in Definition 45.

Lemma 10. *The CLUE scheme Π_{CLUE} defined in Section 4.6 satisfies* simulatable *token and reversible token generation given in Definition 52.*

Proof. Given epoch public and secret keys $k_e = (pk_e, sk_e) := (g^{x_e}, (x_e, \mathrm{D}_e))$ for epoch e and epoch identifier ID_e and update token $\Delta_{\mathsf{e+1}} := (\Delta'_{e+1}, pk_{\mathsf{e+1}}) \in (\mathbb{Z}_q^* \times \mathbb{G}_1)$ for old epoch e updated to the new epoch $(e+1)$, we have $(-x_e + x_{e+1}) := \Delta'_{e+1} \xleftarrow{\$} \mathbb{Z}_q$ such that pk_{e+1} can be generated from the previous epoch as $pk_{e+1} \leftarrow pk_e \cdot g^{\Delta}$. This holds since the CLUE construction derives the epoch public key as $pk_e := g^{x_e}$. We define a reversible token $(\Delta_{e+1})^{-1} := ((\Delta'_{e+1})^{-1}, pk_{e+1} \cdot g^{(\Delta'_{e+1})^{-1}})$. Given $\mathsf{invert}(\Delta'_{e+1}) = (\Delta'_{e+1})^{-1} = (x_e - x_{e+1})$ for some invertible function invert, we can show that $(\Delta_{e+1})^{-1} := ((x_e - x_{e+1}), g^{x_{e+1}} \cdot g^{(x_e - x_{e+1})}) = ((x_e - x_{e+1}), g^{x_e})$. Therefore, we can use downgraded ciphertexts by applying this reversible token as follows:

$$\begin{aligned}
\mathsf{Upd}(pp, C_{e+1}, (\Delta_{e+1})^{-1}) &= (c^1_{e+1}, c^2_{e+1} \cdot \mathsf{F}_{\mathrm{DDH}}((\Delta'_{e+1})^{-1}, \omega)) \\
&= (c^1_e, m^\sigma \cdot \mathsf{F}_{\mathrm{DDH}}(x_{e+1}, g^r) \cdot \mathsf{F}_{\mathrm{DDH}}((x_e - x_{e+1}), g^r) \\
&= (c^1_e, m^\sigma \cdot \mathsf{F}_{\mathrm{DDH}}(x_e, g^r)) = C_e.
\end{aligned}$$

Therefore, simulatable token generation given in Definition 52 is satisfied. □

4.7.2 Proving Security

Theorem 4. *Given* Π_{CLUE} *is a deterministic updatable encryption scheme satisfying randomness-preserving tidy updates (Lemma 8); simulatable token generations (Lemma 10) and the underlying certificateless encryption scheme* $\Pi_{CL\text{-}PKE}$ *satisfies* CL-PKE-IND-RCCA *security (Lemma 11), then the construction* Π_{CLUE} *satisfies security notion* CLUE-IND-RCCA *assuming the intractability of the p-BDHI problem formalised in Definition 8 (Section 2.3).*

Similarly to Chapter 3, Section 3.8, we take a two-step modular approach in proving Theorem 4 to reduce the proof of security from the updatable setting (CLUE) to the standard setting. That is, we provide proof of the security of the underlying CL-PKE scheme $\Pi_{\mathsf{CL\text{-}PKE}} := (\mathsf{Setup}, \mathsf{Partial\text{-}SK\text{-}Extract}, \mathsf{Set\text{-}Secret\text{-}Value}, \mathsf{Set\text{-}SK}, \mathsf{Set\text{-}PK}, \mathsf{Enc}, \mathsf{Dec})$ used to construct Π_{CLUE}.

The *first step* of the proof demonstrates that $\Pi_{\mathsf{CL\text{-}PKE}}$ satisfies CL-PKE-IND-RCCA security in an *isolated epoch* of Π_{CLUE} against the adaptive adversary $\mathcal{A} = (\mathcal{A}_{\mathrm{I}}, \mathcal{A}_{\mathrm{II}})$, which we defined in Section 4.4. To do so we use the CL-PKE-IND-RCCA security model presented in Section 4.5.2, which is designed over a singular epoch of CLUE. We observe that the authors of [93] demonstrated that their CL-PKE scheme, on which ours is based, satisfies the *strictly stronger* notion of CL-PKE-IND-CCA security against adversary $\mathcal{A}$ in the random oracle model assuming the hardness of Definition 8 from Section 2.3. Following the implication from [28, 68] that satisfaction of CCA security implies that the same construction will also satisfy CL-PKE-IND-RCCA security, intuitively one can see that our construction will satisfy RCCA security.

In the *second step* of the proof we look at proving the security of the updatable construction Π_{CLUE} over *multiple epochs*. This part of the proof sees a series of *hybrid games* H_l built for epochs $e_l \in \{0, \ldots, e_{\mathsf{max}} + 1\}$ of Π_{CLUE} where e_{max} is the maximum number of epochs in which an adversary $\mathcal{A}$ can query oracles (Figure 4.1). Suppose we have adversary $\mathcal{A}$ against Π_{CLUE}, defined in Section 4.4. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}_l$ against the standard CL-PKE construction $\Pi_{\mathsf{CL\text{-}PKE}}$ which is proven CL-PKE-IND-RCCA secure in the first part of our proof. Constructing adversaries in this way enables us to demonstrate the indistinguishability of games $\mathsf{H}_{l-1}, \mathsf{H}_l$ for the epochs of the CLUE scheme $e_l \in \{0, \ldots, e_{\mathsf{max}} + 1\}$. Thus, updatable security can be *reduced* to the static security of $\Pi_{\mathsf{CL\text{-}PKE}}$ in an isolated epoch of the CLUE scheme.

At a high level, $\mathcal{B}_l$ *guesses* the *set* of challenge-equal epochs for $\mathcal{A}$, *embeds* their PKE challenge into epoch e_l and *simulates* the challenger in $\mathcal{A}$'s game (Figure 4.2). As a result, game H_l behaves exactly like the CLUE-IND-RCCA game in Figure 4.2 up to epoch e_{l-1} and from epoch e_l onwards, $\mathcal{B}_l$ will randomly determine challenge ciphertexts. As a consequence, we can show that game H_0 is run independently of the challenge bit b, meaning there does not exist an adversary with a non-trivial advantage against H_0. Lastly, one can view $\mathsf{H}_{e_{\mathsf{max}}+1}$, the game in the final challenge epoch, as the CLUE-IND-RCCA-game.

In summary, the above technique sees an adversary $\mathcal{B}_l$ guess a window of epochs; embed a PKE challenge into the CLUE security game; simulate challenger responses of the game in Figure 4.2 for the window of epochs and the challenge epoch, then randomly determine the challenge ciphertext when simulating responses to $\mathcal{A}$ after the challenge epoch. This method allows us to prove that $\mathcal{A}$ has a negligible advantage in distinguishing between games H_{l-1}, H_l for epochs $e_l \in \{0, \ldots, e_{\mathsf{max}} + 1\}$, thus proving that Theorem 4 holds. Observe that we achieve this final result using a *key insulation* method formalised in [80]. More formally,

Lemma 11. *Given a secure KH-PRF F_{DDH}, if the underlying deterministic encryption scheme $\Pi_{\mathsf{CL\text{-}PKE}}$ of Π_{CLUE} satisfies randomness-recoverable tidy encryption (Lemma 9) then $\Pi_{\mathsf{CL\text{-}PKE}} := (\mathsf{Setup}, \mathsf{Partial\text{-}SK\text{-}Extract}, \mathsf{Set\text{-}Secret\text{-}Value}, \mathsf{Set\text{-}SK}, \mathsf{Set\text{-}PK}, \mathsf{Enc}, \mathsf{Dec})$, the underlying encryption scheme of the construction Π_{CLUE} in Section 4.6, satisfies CL-PKE-IND-RCCA-security (Definition 49) assuming the hardness of the p-BDHI problem.*

Sketch Proof. Informally, the underlying CL-PKE scheme $\Pi_{\mathsf{CL\text{-}PKE}}$ of Π_{CLUE} is a modified version of the CL-PKE scheme defined by the authors of [93], excluding the update algorithms. In [93] it is proven that their construction satisfies the *strictly stronger* notion of CL-PKE-IND-CCA security against an adversary $\mathcal{A} = (\mathcal{A}_{\mathrm{I}}, \mathcal{A}_{\mathrm{II}})$, in the random oracle model, assuming the hardness of Definition 8. The proof of CL-PKE-IND-CCA follows similarly for scheme $\Pi_{\mathsf{CL\text{-}PKE}}$.

Proof. The security experiment presented in Figure 4.3 asks an adversary to distinguish bit $b \in \{0, 1\}$ from challenge ciphertext $C = (c^1_{e,b}, c^2_{e,b}) = ((P^{\mathcal{H}_1(\mathrm{ID}_e)} \cdot P')^r, m^\sigma_b \cdot \mathsf{F}_{\mathrm{DDH}}(x_e, g^r))$. Intuitively, this means $\mathcal{A}$ needs to distinguish $c^2_{e,b}$ to win the game. Recall Lemmas 7 and 9, in which we proved the KH-PRF $\mathsf{F}_{\mathsf{DDH}}$ used in Π_{CLUE} is secure assuming the hardness of the DDH problem and that Π_{CLUE} satisfies randomness

recoverable tidy re-encryption respectively. The consequence of these Lemmas means adversary $\mathcal{A}$ has no means to compute $r = \mathcal{H}_3(< m_b^\sigma || pk_e || \mathrm{ID}_e >)$ given uniform randomness $\sigma \in \mathbb{Z}_q^*$, nor is $\mathcal{A}$ capable of computing $g^r := \hat{e}(c_{e,b}^1, \mathrm{D}_e)$ without knowledge of randomness σ, identity ID_e and partial secret key D_e (see construction encryption and decryption details of Π_{CLUE}). The second phase of the experiment presented in Figure 4.3 essentially asks $\mathcal{A}$ to *guess* the input and output of the KH-PRF as well as the uniform randomness σ to distinguish whether the challenge ciphertext is an encryption of challenge message m_0 or m_1.

To conclude, adversary $\mathcal{A}$ in the CL-PKE-IND-RCCA security game detailed in Figure 4.3 is reduced to guessing the bit $b \in \{0, 1\}$ of the game. Therefore, the security of the scheme $\Pi_{\mathsf{CL\text{-}PKE}}$ (which is the standard encryption scheme of our construction Π_{CLUE}) is reduced to solving the p-BDHI problem (Definition 8) which is assumed to be intractable in polynomial time. □

The second step of the proof of Theorem 4 is as follows.

Claim 1. *Hybrid games $\mathsf{H}_{l-1}, \mathsf{H}_l$ following Figure 4.2 for the construction Π_{CLUE} are indistinguishable.*

Sketch Proof. Recall that adversary $\mathcal{B}_l$ will embed a *static* challenge in the security experiment CL-PKE-IND-RCCA into the CLUE-IND-RCCA game in response to adversary $\mathcal{A}$'s challenge in epoch e under given public key pk_e. Note that asides from key pk_e, other epoch keys and update tokens are unknown to $\mathcal{B}_l$ which means that responding to decryption queries from $\mathcal{A}$ calls for $\mathcal{B}_l$ to use its decryption oracle in addition to simulating update tokens (Definition 52).

The simplified method described above is extended to the in-depth proof, in which we require $\mathcal{B}_l$ to guess a *window of epochs* that will contain the correct challenge epoch and embeds their static challenge there. More formally,

Proof. The indistinguishability of consecutive hybrid game pairs sees an adversary $\mathcal{A}$ against the CLUE construction Π_{CLUE} in games $\mathsf{H}_{l-1}, \mathsf{H}_l$ for the CLUE-IND-RCCA security experiment (Figure 4.2), such that $e_l \in \{0, \ldots, e_{\mathsf{max}} + 1\}$ with e_{max} being the final epoch in which $\mathcal{A} = (\mathcal{A}_{\mathrm{I}}, \mathcal{A}_{II})$ can make queries to oracles detailed in Figure 4.1. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}_l$ against the underlying CL-PKE scheme $\Pi_{\mathsf{CL\text{-}PKE}}$ of Π_{CLUE} in the CL-PKE-IND-RCCA game (Figure 4.3 in Section 4.5.2).

The reduction $\mathcal{B}_l$ attempts to simulate the challenger in $\mathcal{A}$'s game using a technique known as *key-insulation*, such that game H_l behaves exactly like the CLUE-IND-RCCA security experiment in Figure 4.2 up to epoch e_{l-1} and the game randomly determines challenge ciphertexts after e_{l-1}. Observe that game H_0 is run independently of the challenge bit b, so $\mathcal{A}$'s advantage of winning is negligible against H_0 and game $\mathsf{H}_{e_{\mathsf{max}}+1}$ represents the CLUE-IND-RCCA game. The idea is for adversary $\mathcal{B}_l$ to correctly guess the boundaries of the set of challenge-equal epochs $\{\underline{e}, \ldots, \overline{e}\}$ containing the challenge-epoch, such that $\mathcal{A}$ does not corrupt the epoch keys in this set nor will they corrupt update tokens $\Delta_{\underline{e}}, \Delta_{\overline{e}+1}$. Adversary $\mathcal{B}_l$ proceeds to embed their CL-PKE-IND-RCCA challenge in epoch e_l, without knowledge of the corresponding epoch secret-key sk_{e_l}.

In more detail, $\mathcal{B}_l$ receives challenge epoch public key $pk_e = g^{x_e}$ from the challenger (C) in the CL-PKE-IND-RCCA game. After updating $\mathcal{K} \leftarrow \mathcal{K} \cup \{e\}$, they proceed to sample $\underline{e} \xleftarrow{\$} \{0, \ldots, e_l\}$, $\overline{e} \xleftarrow{\$} \{e_{l+1}, \ldots, e_{\mathsf{max}}\}$. Observe, reduction $\mathcal{B}_l$ has access to the CL-PKE decryption oracle only and they will attempt to simulate responses in time-frame $\{\underline{e}, \ldots, \overline{e}\}$, without knowledge of the corresponding epoch secret-key and in turn without the capability of deriving the update tokens manually from epochs $\{\underline{e}, \ldots, \overline{e}\}$. Instead, $\mathcal{B}_l$ must simulate the update tokens within this region (Lemma 10). The real challenge is in $\mathcal{B}_l$ simulating the remaining oracles in $\mathsf{Exp}^{\mathsf{CLUE\text{-}IND\text{-}RCCA}}_{\Pi_{\mathsf{CLUE}},\mathcal{A}}$ which we will describe in the ensuing paragraphs, including the time after which $\mathcal{A}$ outputs challenge messages $(m_0, m_1) \in \mathcal{M}^*$ and receives a challenge-ciphertext.

$\mathcal{O}_{\mathsf{Next}}(e)$: if the call is made by $\mathcal{A}$ in epoch $e < (\underline{e} - 1)$ *or* $e > \overline{e}$, then $\mathcal{B}_l$ generates a new epoch public and secret keys $k_{e+1} := (pk_{e+1}, sk_{e+1})$ such that sk_{e+1} derives from sampling $x_{e+1} \xleftarrow{\$} \mathbb{Z}_q^*$, $\mathrm{D}_e \xleftarrow{\$} \mathbb{G}_1$ and setting $sk_{e+1} := (x_{e+1}, \mathrm{D}_e)$ followed by computing $pk_{e+1} = g^{x_{e+1}}$. Then $\mathcal{B}_l$ generates an update token $\Delta_{e+1} = ((-x_e + x_{e+1}), pk_{e+1})$. For query epoch $e = \underline{e} - 1$, $\mathcal{B}_l$ can simply set $\Delta_{e+1} = \perp$, since $\mathcal{A}$ is assumed not to query the token in this epoch and set $pk_{e+1} = g$. For query epoch $e = \overline{e}$, again $\mathcal{B}_l$ sets $\Delta_{e+1} = \perp$ and the epoch key $k_{e+1} = (g^{x_{e+1}}, (x_{e+1}, \mathrm{D}_e))$ for randomly sampled secret key components (x_{e+1}, D_e).

For query epochs $\underline{e} \leq e \leq \overline{e}$, $\mathcal{B}_l$ must sample a random value $\alpha_{e+1} \xleftarrow{\$} \mathbb{Z}_q^*$, setting the new epoch key as $k_{e+1} \leftarrow (pk_e^{\alpha_{e+1}}, (x_{e+1} \cdot \alpha_{e+1}, \mathrm{D}_e))$ and update token as $\Delta_{e+1} = ((-x_e + (x_{e+1} \cdot \alpha_{e+1})), pk_{e+1})$. Finally, $\mathcal{B}_l$ must deal with the instance that $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Next}}$ in an epoch proceeding the challenge epoch ($e > \tilde{e}$), by updating the list $\mathcal{L}$ as follows: if $e = \underline{e} - 1$, choose $r' \xleftarrow{\$} \mathbb{Z}_q^*$ and set $C_{e+1} = ((c_e^1)^{r'}, (m_b)^{\sigma} \cdot \mathsf{F}_{\mathrm{DDH}}(x_e, \omega^{r'}))$

where $\omega = g^r$ (see the CLUE construction in the main text); if $e = \overline{e} - 1$, sample $\alpha' := \{\alpha_{\underline{e}+1}, \ldots, \alpha_{\overline{e}}\} \xleftarrow{\$} \mathbb{Z}_q^*$ and set $C_{e+1} = ((c_e^1)^{\alpha'}, (m_{b'})^{\sigma} \cdot \mathsf{F}_{\mathrm{DDH}}(x_e, \omega^{\alpha'}))$; if $e \geq \overline{e}$, choose C_{e+1} to be two random group elements. Otherwise, C_{e+1} is determined by regular updates. In all cases $\mathcal{B}_l$ updates $\mathcal{L} \leftarrow \mathcal{L} \cup \{(C_{e+1}, e+1)\}$ and the global state is updated to epoch $(e+1)$.

$\mathcal{O}_{\mathsf{Dec}}(C_{e_i})$: if the call is made by $\mathcal{A}$ for epoch $e_i \notin \{\underline{e}, \ldots, \overline{e}\}$, $\mathcal{B}_l$ can simply run $\mathsf{Dec}(pp, C_{e_i}, sk_{e_i}) \rightarrow m$ by first deriving the epoch key $k_{e_i} := (pk_{e_i}, sk_{e_i})$. If however $e_i \in \{\underline{e}, \ldots, \overline{e}\}$, as $\mathcal{B}_l$ does not know the corresponding secret key for these epochs, they have to manually update C_{e_i} to epoch e by iteratively running $C_{e_j} \leftarrow \mathsf{Upd}(pp, C_{e_i}, \Delta_j^{-1})$, for $j = \{i-1, \ldots, l\}$, such that $e_i > e_l$, or $C_{e_j} \leftarrow \mathsf{Upd}(pp, C_{e_i}, \Delta_j)$, for $j = \{i+1, \ldots, l\}$ such that $e_i < e_l$. This is achieved using simulated reverse or forward tokens to down or up-grade the ciphertext respectively, assuming the existence of simulatable tokens (Definition 52, Lemma 10). Next, $\mathcal{B}_l$ queries their own challenger C on the CL-PKE decryption oracle $\mathcal{O}_{\mathsf{Dec}}(C_{e_l})$ to obtain the message m. The adversary has to do so, as they do not know the secret epoch key sk_{e_l} to manually decrypt. Note, the queried ciphertext does not need to appear 'fresh' to $\mathcal{A}$ once downgraded by a reversible token since $\mathcal{A}$ does not see the downgraded version of the ciphertext. Provided the output $m \notin \mathcal{M}^*$, $\mathcal{B}_l$ responds to $\mathcal{A}$ with the decryption m.

$\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}(e_i)$: if the call is made by $\mathcal{A}$ for epoch $e_i < e$, $\mathcal{B}_l$ returns Δ_{e_i} provided $e_i \neq \{\underline{e}, \overline{e}\}$ and they return $\perp$ otherwise. Note that the $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}$ returns $\perp$ for $e_i \geq e$ and for $e_i \in \{\underline{e}, \overline{e}\}$ which is important as the ciphertext update functionality is deterministic, therefore, corruption of an update token for such an epoch would allow an adversary to trivially update ciphertext C_e to C_{e_i} themself and compare the results with the output of $\mathcal{O}_{\mathsf{Upd}}(C_{e_i})$. This would allow the adversary to distinguish between hybrid games by checking for inconsistencies between the two games.

$\mathcal{O}_{\mathsf{PSKE}}(e_i)$: if the call is made by $\mathcal{A}$ for epoch $e_i \geq e$ or $e_i \in \mathcal{K}$ then $\mathcal{B}_l$ returns $\perp$. Otherwise, $\mathcal{B}_l$ returns D_A provided $e \neq \{\underline{e}, \overline{e}\}$.

$\mathcal{O}_{\mathsf{Corrupt\text{-}Key}}(e_i)$: Similarly, if the call is made by $\mathcal{A}$ for an epoch $e_i < e$, $\mathcal{B}_l$ can return key sk_{e_i} provided $e_i \notin \{\underline{e}, \ldots, \overline{e}\}$ and $e_i < e$. Else, output $\perp$.

$\mathcal{O}_{\mathsf{Upd}}(C_{e_i})$: for some epoch $e_i < e$, adversary $\mathcal{B}_l$ first checks if $e_i \notin \{\underline{e}, \ldots, \overline{e}+1\}$ then they can run $C_e \leftarrow \mathsf{Upd}(pp, C_e, \Delta_e)$ itself using simulated token Δ_e. If $e_i \in \{\underline{e}, \overline{e}+1\}$ then $\mathcal{B}_l$ can call the CL-PKE decryption oracle $\mathcal{O}_{\mathsf{Dec}}(C_{e_i})$, obtaining a message m and

proceed to encrypt for epoch e by running $C_e \xleftarrow{\$} \mathsf{Enc}(pp, m, pk_e, \mathrm{ID}_e)$. Ciphertext C_e satisfies randomness-preserving re-encryption following Lemma 8, so $\mathcal{A}$ is unable to detect whether the ciphertext is a fresh or updated ciphertext.

If $\mathcal{A}$ queries an update of the challenge ciphertext $\tilde{C}$ (challenge epoch $\tilde{e}$) to current epoch e, then $\mathcal{B}_l$ proceeds as follows: if $e \neq \{e_l, \overline{e}+1\}$ then simulated tokens can be used in the normal manner of ciphertext updates. If $e = e_l$, $\mathcal{B}_l$ proceeds by decrypting $m \leftarrow \mathsf{Dec}(pp, \tilde{C}, sk_{e-1})$ then sending (m_0, m) to C to receive a challenge ciphertext $\tilde{C}_e$.[11] This is possible as $\mathcal{B}_l$ can retrieve previous challenge ciphertexts since they're acting as the challenger to adversary $\mathcal{A}$. Finally, if $e = \overline{e}+1$ then $\mathcal{B}_l$ can run $\tilde{C}_e \xleftarrow{\$} \mathsf{Enc}(pp, m_0, pk_e, \mathrm{ID}_e)$ which is compliant with both hybrid games $\mathsf{H}_{l-1}, \mathsf{H}_l$.

Assuming $\mathcal{B}_l$ correctly guesses the boundaries plus challenge-epoch e_l and following the above reduction, the view of $\mathcal{A}$ is that they are playing game H_{l-b}. Following the proof of Claim 11 if $\mathcal{B}_l$ chooses $b = 0$ in the CL-PKE-IND-RCCA game, then $\mathcal{A}$ is against the security experiment for CLUE-IND-RCCA up until epoch e_{l-1}, which translates to game H_l (similarly if $b = 1$ then $\mathcal{A}$ plays H_{l-1}). Thus, $\mathcal{A}$'s guess b' a guess of bit-b and so if we label the event that $b' = b$ as Succ, we have that $\frac{1}{e_{\max}+1} \Pr[\mathsf{Succ}]_{\Pi_{\mathsf{CLUE}},\mathcal{A}} \leq \Pr_{\Pi_{\mathsf{CL\text{-}PKE}},\mathcal{B}_l}[\mathsf{Succ}] \leq \mathsf{negl}(1^\lambda)$, where the second inequality derives from the proof of Claim 11. □

4.7.3 Efficiency

In Table 4.1 we make explicit the cost of encryption, decryption and ciphertext updates of our CLUE scheme (Π_{CLUE}). To be precise, the encryption algorithm only has one exponentiation in both $\mathbb{G}_2$ and a multi-exponentiation in $\mathbb{G}_1$. Decryption requires the computation of a pairing, one exponentiation in $\mathbb{G}_2$ and a multi-exponentiation in $\mathbb{G}_1$. In addition, the token generation algorithm Set-Token requires one exponentiation in $\mathbb{G}_2$; the Upd algorithm requires the computation of a pairing and one exponentiation in both $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. The size of an updated ciphertext in Π_{CLUE} is the same as from encryption, that is, $2|\mathbb{G}_1|$. One can see that running the ciphertext update algorithm is more efficient than decryption, albeit both require a pairing computation.

Nevertheless, the encryption in our construction has comparable efficiency to a pairing-free CL-PKE scheme [7]. The underlying CL-PKE scheme in Π_{CLUE} from Section 4.6

[11] That is, due to updates being deterministic in construction Π_{CLUE}, $\mathcal{B}_l$ will update a challenge ciphertext from $\mathcal{A}$ by decrypting the message and sending this message alongside the *fixed challenge message* m_0 (which has consistent randomness) as it's own challenge to C.

Efficiency			
	Encryption	**Decryption**	**Update**
Our Construction	1 multi-exp ($\mathbb{G}_1$), 1 exp ($\mathbb{G}_2$)	1 pairing, 1 multi-exp ($\mathbb{G}_1$), 1 exp ($\mathbb{G}_2$)	1 pairing, 1 exp ($\mathbb{G}_1, \mathbb{G}_2$)
[93] - NewFullCLE	1 multi-exp ($\mathbb{G}_1$), 2 exp ($\mathbb{G}_2$)	1 pairing, 1 multi-exp ($\mathbb{G}_1$), 1 exp ($\mathbb{G}_2$)	N/A
[7] - CLPKE (No pairings)	3 exp in group $\mathbb{G}$	3 exp in group $\mathbb{G}$	N/A
[2] - FullCL-PKE	1 pairing, 2 exp ($\mathbb{G}_1$), 1 exp ($\mathbb{G}_2$)	1 pairing, 2 exp ($\mathbb{G}_1$)	N/A
[68] - CL-PRE	2 pairings, 3 exp ($\mathbb{G}_1$), 1 exp ($\mathbb{G}_2$)	1 pairing, 1 exp ($\mathbb{G}_1$), 2 exp ($\mathbb{G}_2$)	6 pairings

Table 4.1 Efficiency comparisons for algorithms (Enc, Dec, Upd) of our construction against the literature.

slightly improves upon the efficiency of the CL-PKE scheme given in [93] as one less exponentiation is $\mathbb{G}_2$ is required for encryption.

Additionally, we observe that the data owner may *pre-compute* and store $(P^{\mathcal{H}_1(\mathrm{ID}_e)} \cdot P')$, which is part of the first component of the ciphertext (c^1), so that validation of the ciphertext in the decryption process only requires a scalar multiplication in $\mathbb{G}_1$. This pre-computation also speeds up the encryption operation for the data owner if they encrypt several messages using the same public-key [93].[12] We note the following observations made by the authors of [93]. With pre-computation, our construction is comparably efficient to the pairing-based CL-PKE scheme from [2]. However, with the removal of pre-computation both the CL-PKE scheme from [93], and by extension in our construction, are more efficient than the scheme from [2] due to the fact encryption has no pairing computation.

Importantly, our final comparison of efficiency is against the most similar updatable CL-PKE scheme that satisfies CL-PKE-IND-RCCA security. Namely, the CL-PRE scheme proposed in [68] requires three exponentiations in $\mathbb{G}_2$ for the re-encryption key; the computation of six pairings in their ciphertext re-encryption process and the re-encrypted ciphertext is of size $(|\mathbb{G}_2| + l)$ for $l \in \mathbb{N}$ which differs in size compared to

[12] In the case of Π_{CLUE}, encrypting several messages under the same public-key translates to encryption of messages in a single epoch of the scheme.

a ciphertext generated from fresh encryption. In comparison to [68], one can therefore see that our construction Π_{CLUE} in Section 4.6 is more efficient with regards to the update feature. In summary, Table 4.1 provides explicit efficiency details for the works mentioned, and in the Table, we detail the cost of encryption, decryption and update algorithms (if applicable) of the aforementioned schemes.

4.8 Summary and Outlook

In our first contribution of this Chapter, we formally defined a novel certificateless public-key updatable encryption primitive CLUE to mitigate the risk of a malicious key generation centre, when considering applications of a PKUE primitive in a public key infrastructure. In our second contribution, we provided a security framework to model the first notion of ciphertext indistinguishability in certificateless public key updatable schemes. In particular, security against replayable chosen-ciphertext attacks from an adaptive adversary. Our third contribution was to propose a concrete CLUE scheme (Π_{CLUE}) derived from a modified pairing-based CL-PKE scheme [93], which we used as the underlying PKE scheme, and KH-PRFs applied to support the necessary update mechanism in CLUE. In doing so, we were able to demonstrate an efficiency improvement compared to other certificateless updatable schemes and provide proof that our construction satisfies ciphertext indistinguishability.

Generally speaking, it is of interest to extend the security framework of CLUE and realise a more efficient scheme, especially one that does not rely on pairings, to limit the cost of computation. Additionally, for future work, we believe it is important to define a concrete CLUE scheme that achieves the stronger notion of *ciphertext unlinkability*, meaning an adversary cannot determine whether a ciphertext derives from an update or fresh encryption. Currently, our proposed construction Π_{CLUE} only achieves ciphertext indistinguishability since the first component of the ciphertext does not evolve with ciphertext updates. In terms of security, we would also like to construct a secure scheme from post-quantum assumptions.

Moving forwards, it is of interest to explore different methods to reduce trust in the KGC which plays a role in a CLUE scheme. That is, instead of taking the certificateless approach to mitigate the key escrow problem, we believe it would be interesting to apply methods similar to [20] which employ multiple KGCs. Alternatively, we could consider the accountable authority IBE (A-IBE) primitive first proposed by the authors

of [63], which is a primitive in which there is some type of penalty imposed on a singular malicious KGC.

Chapter 5

Dynamic Multi-Server Updatable Encryption

Contents

In this Chapter we focus on mitigating a problem inherent in the design of public-key updatable encryption. Concretely, a PKUE scheme lacks resilience due to a single point of failure if the server performing ciphertext updates becomes corrupted. Our solution is a multi-server definition of a PKUE primitive (DMUE) which supports dynamic changes in the participating servers from one epoch to the next. We model security for a DMUE scheme, generically define a DMUE construction and prove the construction satisfies our notion of security. Additionally, we consider the practicalities of DMUE in the real world, which leads us to use dynamic secret sharing as a building block in order to achieve a concrete scheme.

5.1 Introduction

5.1.1 Motivation

Standard encryption schemes required a secure solution to so-called encrypted 'data at rest' due to the increased usage of outsourced data storage. To mitigate the risk of long-term cryptographic key exposure, the authors of [24] introduced the symmetric updatable encryption primitive. Recollect, from Chapters 3 of this Thesis, that we extended the UE primitive to the public-key setting (PKUE from Section 3.3). We will continue to extend PKUE in this Chapter to support additional features.

We emphasise that the core purpose of UE is to reduce the impact of key exposure and, in turn, token exposure, preserving standard encryption security such as *confidentiality* and the updatable security notion of *ciphertext unlinkability*. Consequently, the primitive is an essential tool for privacy preservation in multiple applications such as storage on the cloud, retention of online medical information, and storing information on the blockchain. Despite efforts to increase the security of UE schemes, there remain risks to the security and resilience of a system in which we rely on a *single* server to perform ciphertext updates. That is, there is a *single point of failure* if the server is corrupted by an external adversary or acts dishonestly. As a consequence, the server could fail to update the ciphertext correctly, if at all.

In more detail, if the corrupted server fails to update the ciphertext, then the data owner's encrypted data will remain encrypted under the same key. As such, the scheme is effectively reduced to a standard encryption scheme which defeats the core purpose of UE since an adversary has more time in which to corrupt the cryptographic key and learn the underlying message. The other cause for concern is if the corrupted server incorrectly updates the ciphertext, which will compromise the integrity of the data and potentially prevent the retrieval of the original data. In the real world, if the data owner is storing encrypted financial information, and it is updated incorrectly, they may be misinformed about their assets upon decryption. In summary, it is vitally important that the chance of either scenario occurring is reduced to protect the *privacy and integrity* of the information. Especially given the highly sensitive nature of the information outsourced in applications of UE.

We believe a natural solution to the single point of failure issue, in the public-key UE (PKUE) setting (Chapter 3), is to *distribute* the update of ciphertexts to *multiple*

servers such that some pre-defined *threshold* of servers can carry out the update process in each epoch. An added benefit is the spreading of the computational load on the server's behalf.

Alas, a *static* set of servers (i.e the chosen set of servers at the advent of the scheme does not change, as in single server UE) does not reflect the real world because servers often change over long periods or possibly need to be removed from a scheme due to dishonest behaviour. To illustrate, suppose we wish to store a secret on a public-key blockchain such that nodes of the blockchain structure are considered to be the servers in a multi-server UE scheme. The authors of [15] demonstrated that *node churning* needs to be taken into consideration when designing a scheme for this application. This led us to design a multi-server PKUE primitive supporting a so-called *dynamic committee* of servers from one epoch to the next, labelled as dynamic multi-server UE (DMUE). We note that the approach of having a dynamic committee of servers is similar to previous works such as [9, 86, 98, 109].

Categorically, DMUE captures servers in specific epochs each possessing an update token. Servers participate in updating the encrypted data, whereby the committee of servers in consecutive epochs may differ and so a *handover protocol* is required. Defining the security of a DMUE primitive will prove challenging and is nuanced. Categorically, the inherent challenges in capturing security in the update setting due to the pervasive leakage of information in UE schemes are only exacerbated in the multi-server setting. At a high level, our security modelling approach stipulates that an adversary can only succeed in their attack if they corrupt a *threshold* of server tokens in any given time period, with the servers and threshold potentially evolving with each epoch. We observe that the adversary modelled is assumed to be *mobile* [109], which means they can dynamically and actively corrupt servers at any given time in a DMUE scheme, provided their corruption capabilities are bounded.

5.1.2 Existing Work

We hark back to Chapters 3 and 4 which discuss the closely aligned *proxy re-encryption* (PRE) primitive introduced by [17] as a symmetric primitive in which a proxy server re-encrypts a ciphertext under a sender's secret key and *delegates* decryption under a recipient's secret key. As noted throughout and detailed in Section 3.2 (Table 3.1), extensions of PRE and UE such as DMUE, remain distinct as they do not affect the

fundamental differences between the underlying standard UE and PRE primitives. However, it is important to observe the existence of *threshold PRE* [124, 33] schemes which distribute the process of ciphertext re-encryption and decryption delegation using secret sharing and standard PRE as building blocks. The concept of using threshold techniques is similar to our DMUE primitive.

In the same vain as DMUE, which is a dynamic threshold updatable encryption primitive, the authors of [74] propose an *updatable* oblivious key management system (UOKMS) in the public-key setting, achieved using a *distributed threshold implementation*. Crucial differences between UOKMS and traditional UE are highlighted by the authors of [74], including but not limited to the design in a UOKMS scheme for the data owner to interact between two outsourced remote services, a KMS and a storage server, whereas, UE literature only captures a server. Moreover, the UOKMS is uniquely designed to have obliviousness of computation on the side of KMS, something that is not considered in the design of UE schemes.

5.1.3 Our Contributions

To summarise, our contributions are threefold: we formalise a dynamic multi-server updatable primitive called DMUE in Section 5.2, used to mitigate the problem of a single point of failure in standard UE schemes. Importantly, the extension of PKUE to the multi-server setting broadens the landscape in which updatable encryption can be applied.

In Section 5.3 we present a new notion of security against update unlinkable chosen ciphertext attacks (MUE-IND-CCA), which captures a mobile adversary attempting to corrupt a threshold or more of secret update tokens. Recall, it is crucial to maintain ciphertext update unlinkability as it guarantees a ciphertext generated by the update algorithm is unlinkable from a ciphertext generated by fresh encryption, even when the adversary sees many updated ciphertexts of chosen messages. We highlight that this indistinguishability notion is the first CCA security definition defined in the PKUE framework, as Chapters 3 and 4 captured the weaker notion of RCCA security. To be clear, we choose to capture CCA security in this Chapter in line with the opinion of UE literature which places emphasis on modelling strong security notions. Moreover, we are able to do so, unlike Chapter 3, since the update mechanism of DMUE is designed to be *deterministic*. Note that Chapter 4 is also designed with deterministic updates,

but we modelled an RCCA notion for a CLUE scheme to suit the security needs of a PKUE *and* certificateless encryption, which does not need to be considered for DMUE as it is not certificateless. We defer the reader to the aforementioned Chapters for an in-depth discussion of RCCA security but we note that formalising MUE-IND-CCA security, and proving this notion is satisfied by a DMUE scheme, directly implies a notion of replayable security (RCCA) is also satisfied.

In Section 5.4 we present the first notion of integrity for a PKUE scheme. Specifically, we model and formalise a notion of security against ciphertext integrity attacks (MUE-INT-CTXT). This security notion is important to consider when there are *deterministic* ciphertext updates since it is easier for an adversary to successfully forge a ciphertext that correctly decrypts to a valid message. Even more so, we are interested in a notion of integrity for the multi-server setting assuming an adversary corrupting a threshold of servers (or there is a collusion of servers) seeks to output forged ciphertexts via the update process. Without guaranteeing ciphertext integrity, a forged ciphertext could mislead the data owner (individual or organisation) which affects the reliability and accuracy of the underlying data.

In Section 5.5 we present a generic construction of DMUE built from a single-server public-key UE primitive and a *dynamic* threshold secret sharing scheme. The crux of our generic construction is that the data owner acts as the dealer and distributes a vector of n update tokens shares per epoch to the corresponding servers. Then at least a threshold of t servers can honestly reconstruct an updated ciphertext to encryption in epoch $(e+1)$ using standard PKUE and secret sharing techniques. Next, we consider the practicalities of applying DMUE by providing an overview of a concrete scheme built from dynamic proactive secret sharing, in which an old server committee participates in a handover process to refresh and securely distribute update tokens to the new epoch server committee. We conclude our work by proving the generic DMUE scheme satisfies our ciphertext unlinkability and integrity security notions.

5.2 Dynamic Multi-Server Updatable Encryption

In this Section, we first introduce the syntax used for defining DMUE and the notation used in the ensuing security modelling Sections. Second, we present the formal definition of a DMUE primitive and define correctness.

5.2.1 Syntax

Recollect general syntax for UE from Section 3.2. In the *dynamic multi-server* setting, we define for epoch e_i a set of servers $S_{e_i} = \{S^j\}_{j\in[n]}$ where $S_{e_{i+1}}$ may not be the same set and update token $\Delta^j_{e_i}$ pertains to the token server $S^j \in S_{e_i}$ possesses.

5.2.2 Formal Definition of DMUE

When extending the PKUE primitive to the dynamic multi-server setting (DMUE), a data owner must distribute tokens to every qualified server for that epoch, who respectively work together to update the ciphertext. Furthermore, the dynamic aspect of this primitive enables different sets of servers, chosen by the data owner at the time of token creation, to perform the ciphertext update in successive epochs. We define the DMUE primitive as follows.

A DMUE primitive is defined as $\Pi_{\mathsf{DMUE}} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{TG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Upd})$ whereby algorithms $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ are formalised as in standard PKUE and the data owner runs all algorithms asides from Upd, the latter of which is run by the servers in a given epoch. More formally,

Definition 54 (DMUE)**.** *Given a set of servers S of size $n \in \mathbb{N}$ and a threshold $t \leq n$, a dynamic multi-server updatable encryption scheme is defined by a tuple of six PPT algorithms $\Pi_{DMUE} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{TG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Upd})$ as follows.*

1. $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$: given security parameter 1^λ, the setup algorithm randomly outputs the public parameters pp.

2. $\mathsf{KG}(pp, e_i) \xrightarrow{\$} k_{e_i} := (pk_{e_i}, sk_{e_i})$: given public parameters, the probabilistic key generation algorithm outputs the public and private key pair (pk_{e_i}, sk_{e_i}) for epoch $\{e_i\}_{i\in[0,\mathsf{max}]}$.

3. $\mathsf{TG}(pp, sk_{e_i}, k_{e_{i+1}}, S_{e_{i+1}}) \rightarrow \{\Delta^j_{e_{i+1}}\}_{j\in[n]}$: the token generation algorithm takes as inputs the public parameters, the old epoch secret key sk_{e_i}, the new epoch public and secret key-pair $k_{e_{i+1}} := (pk_{e_{i+1}}, sk_{e_{i+1}})$ generated by the key generation algorithm and the new set of servers $S_{e_{i+1}} = \{S^j\}_{j\in[n]}$. The deterministically computed output is n update tokens $\{\Delta^j_{e_{i+1}}\}_{j\in[n]}$, which are securely sent to the *chosen* servers $S^j \in S_{e_{i+1}}$.[1]

[1]In the definition of DMUE the data owner chooses the committee of servers $\{S_{e_{i+1}}\}_{\forall i\in\mathbb{N}}$. In Section 5.5 we design a generic DMUE scheme built from PKUE and threshold dynamic proactive

4. $\mathsf{Enc}(pp, pk_{e_i}, m) \xrightarrow{\$} C_{e_i}$: given public parameters and the epoch public key pk_{e_i}, the data owner runs the probabilistic encryption algorithm on message $m \in \mathcal{MSP}$ and outputs the ciphertext C_{e_i}.

5. $\mathsf{Dec}(pp, sk_{e_i}, C_{e_i}) \rightarrow \{m, \perp\}$: given public parameters and the epoch secret key, the owner is able to run the deterministic decryption algorithm in order to output message m or abort ($\perp$).

6. $\mathsf{Upd}(pp, \{\Delta^k_{e_{i+1}}\}_{k\in\mathbb{N}}, C_{e_i}) \rightarrow C_{e_{i+1}}$: for some $k \geq t$, the subset $S' \in S_{e_{i+1}}$ of servers, such that $|S'| = k$, can deterministically update ciphertext C_{e_i} using their tokens $\Delta^k_{e_{i+1}}$ to output an updated ciphertext $C_{e_{i+1}}$.

Correctness Intuitively, defining the correctness of the DMUE primitive follows from Definition 29 for the single server PKUE primitive. However, the multi-server setting additionally has to encapsulate the concept of ciphertext updates from a *threshold* number of tokens. The formal definition of correctness follows.

Definition 55 (Correctness)**.** *Given security parameter λ and threshold $t \leq k \leq n$, dynamic multi-server updatable encryption scheme (Π_{DMUE}) for n servers, as formalised in Definition 54, is correct if for any message $m \in \mathcal{MSP}$, for any $l \in \{0, \ldots, \mathsf{max}-1\}$ such that max denotes the final epoch of the scheme, and $i = (l+1)$, there exists a negligible function negl such that the following holds with overwhelming probability.*

$$\Pr\left[\begin{array}{l} pp \xleftarrow{\$} \mathsf{Setup}(1^\lambda); \\ k_{e_i} = (pk_{e_i}, sk_{e_i}) \xleftarrow{\$} \mathsf{KG}(pp, e_i); \\ \{\Delta^j_{e_i}\}_{j\in[n]} \leftarrow \mathsf{TG}(pp, sk_{e_{i-1}}, k_{e_i}, S_{e_i}); \\ C_{e_l} \xleftarrow{\$} \mathsf{Enc}(pp, pk_{e_l}, m); \\ \{C_{e_i} \leftarrow \mathsf{Upd}(pp, \{\Delta^k_{e_i}\}_{k\in\mathbb{N}}, C_{e_{i-1}}) : \\ i \in \{l+1, \cdots, \mathsf{max}\} \wedge |k| \geq t\}; \\ \mathsf{Dec}(pp, sk_{e_{\mathsf{max}}}, C_{e_{\mathsf{max}}}) = m \end{array}\right] \geq 1 - \mathsf{negl}(1^\lambda).$$

secret sharing, such that each server in an epoch committee holds a share of the complete update token for the next epoch. As a consequence, we require a threshold of servers from one epoch to the next to be *distinct*, otherwise, the security of the scheme may be affected if a threshold or more servers collude whilst possessing token shares from consecutive epochs. We defer the readers to Section 5.5 for a more in-depth discussion regarding this matter.

Remark 6. The *multi-server* aspect of DMUE affects the Setup, TG, and Upd algorithm definitions compared to standard UE algorithms. We note that if $t = n = 1$ then Definition 54 reduces to the single server, traditional UE definition.

5.3 Security Modelling

As in single server PKUE, security modelling in this Section must capture the *inferable* information an adversary can learn from *querying* relevant oracles. We start by detailing the lists recorded and tracked by the challenger, plus the oracles necessary to model security of Definition 54 concerning the dynamic multi-server setting. For ease of reading, we also recall the details of the remaining oracles required in security modelling that are unchanged from the single-server PKUE.

Briefly, our security notion satisfies the benchmark level of security for *confidentiality* in *deterministic* PKUE schemes. More specifically, the experiment models our MUE-IND-CCA notion capturing security against *update unlinkable chosen ciphertext* attacks. Crucially, unlinkability needs modelling to ensure that a ciphertext generated by the update algorithm is indistinguishable from a ciphertext generated by fresh encryption. Recall our contributions discussion, from Section 5.1, in which we cited the motivation for modelling the first CCA security notion of a PKUE scheme, rather than RCCA security. Simply put, UE literature places utmost importance on satisfying high levels of security, and given we are designing a generic DMUE scheme capable of achieving CCA security we believe it is important to model the strongest notion of security. Observe, should a different setting for DMUE need to be considered, such as a scheme with probabilistic updates, we would have to relax security to a replayable CCA (RCCA) notion as in Chapter 3.

We highlight that our security model defines an adversary $\mathcal{A} := \{\mathcal{A}_I, \mathcal{A}_{II}\}$, representing a malicious outside adversary ($\mathcal{A}_I$) and dishonest server ($\mathcal{A}_{II}$). Typically, only adversary $\mathcal{A}_I$ is considered in single server UE, as the literature assumes the lone server is honest. Given the main motivator of this Chapter, to tackle the issue of a single point of failure regarding server updates, we have to consider adversary $\mathcal{A}_{II}$ to capture the threat of dishonesty or collusion of a *threshold* or more servers.

Observe that our security experiment and definition of ciphertext unlinkability will use the notation $\mathcal{A}$ to capture both adversarial types simultaneously. Note that both

types of adversaries are encapsulated in the modelling of the lists and oracles we will describe in the proceeding subsections. Informally, an outside adversary is modelled in the usual manner of UE, through lists recording corrupted and inferable information. Specific to a corrupt server, the token corruption oracle is crucial in recording any epoch in which a threshold or more tokens have been corrupted.

5.3.1 Lists

We provide Figure 5.1 as a descriptive summary of the lists maintained by the challenger (as part of the global state GS) in the ensuing security experiment. Observe that intuition for these lists has previously been provided in Section 3.4 of PKUE security modelling. Note, we do not require list $\mathcal{M}^*$ to record challenge messages (a necessary list in the RCCA setting as in Chapter 3) as we are modelling CCA security. Further, the main deviation in the list description compared to the single server PKUE setting is found in list $\mathcal{T}$. This difference is necessary for the security of DMUE as we seek to model schemes that tolerate up to a threshold of tokens being corrupted in any given epoch. However, list $\mathcal{T}$ is incorporated into the recording of corruption epochs in list $\mathcal{C}^*$, therefore, the latter list is also modified to the multi-server setting in the same manner.

Before presenting Figure 5.1, we briefly summarise our discussion of list $\mathcal{C}^*$ and the *challenge-equal predicate* (Definition 30 from Section 3.4). Recall, list $\mathcal{C}^*$ is built from Definition 30, and it is an essential list maintained by the challenger to prevent trivial wins. Specifically, the list records all of the *challenge-equal epochs* in which the adversary knows a *version* of the challenge ciphertext, either from calls to the update oracle or through computation. The latter is possible since there are epochs in which the adversary can infer information using corrupted ciphertexts and tokens, especially epochs belonging to lists $\mathcal{C}$ and $\mathcal{T}$.

5.3.2 Oracles

First, we describe the five oracles $\mathcal{O} = \{\mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Next}}, \mathcal{O}_{\mathsf{Upd}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Token}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}\}$ at a high level before providing detail of how they run.

- $\mathcal{O}_{\mathsf{Dec}}$: to prevent an adversary from trivially winning by querying the decryption of a queried challenge ciphertext, the following condition must be satisfied. The predicate $\mathsf{isChallenge}$ (Definition 23, Section 3.2) must return false. In this case,

- $\mathcal{L} = \{(e', C_{e'})_{e' \in [e]}\}$: the list containing the epoch and corresponding ciphertext in which the adversary learns (through queries to the update oracle $\mathcal{O}_{\mathsf{Upd}}$) an updated version of an *honestly generated* ciphertext.
- $\mathcal{K} = \{e' \in [e]\}$: the list of epoch(s) in which the adversary has obtained an epoch secret key through calls to $\mathcal{O}_{\mathsf{Corrupt\text{-}Key}}(e')$.
- $\mathcal{T} = \{e' \in [e]\}$: the list of epoch(s) in which the adversary has obtained at least a threshold number of update tokens through calls to $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}(e')$.
- $\mathcal{C} = \{(e', C_{e'})_{e' \in [e]}\}$: the list containing the epoch and corresponding ciphertext in which the adversary learns (through queries to the update oracle $\mathcal{O}_{\mathsf{Upd}}$) an *updated version* of the challenge ciphertext.
- $\mathcal{C}^* \leftarrow \{e' \in \{0, \ldots, e_{\max}\} | \mathsf{challenge\text{-}equal}(e') = \mathsf{true}\}$: the list of challenge-equal ciphertexts, defined by a recursive predicate $\mathsf{challenge\text{-}equal}$, such that $\mathsf{true} \leftarrow \mathsf{challenge\text{-}equal}(e')$ iff : $(e' \in \mathcal{C}) \vee (\mathsf{challenge\text{-}equal}(e' - 1) \wedge e' \in \mathcal{T}) \vee (\mathsf{challenge\text{-}equal}(e' + 1) \wedge (e' + 1) \in \mathcal{T})$.

Fig. 5.1 The set of lists $\mathbf{L} := \{\mathcal{L}, \mathcal{K}, \mathcal{T}, \mathcal{C}^*\}$ the challenger maintains in the global state (GS) as a record of during security games.

the decryption of a valid ciphertext under the current epoch secret key is returned. Else, the failure symbol $\perp$ is returned.

- $\mathcal{O}_{\mathsf{Upd}}$: the update oracle only accepts and responds to calls regarding *honestly generated* ciphertexts or derivations of the challenge ciphertext, by checking lists $\{\mathcal{L}, \mathcal{C}^*\}$ respectively. If this is the case, the output is an update of the queried ciphertext to the current epoch. Next, the updated ciphertext and current epoch are added to the list $\mathcal{L}$. Moreover, if the $\mathsf{isChallenge}$ predicate returns *true* on the input of the *queried* key and ciphertext, then the current epoch is added to the *challenge-equal* epoch list $\mathcal{C}^*$.
- $\mathcal{O}_{\mathsf{Next}}$: queries to the next oracle in challenge epoch e result in an update of the global state to the epoch $(e + 1)$. This is achieved by running key and token generation algorithms to output the epoch key pair $k_{e+1} = (pk_{e+1}, sk_{e+1})$ and tokens $\Delta^j_{e+1}, \forall j \in [n]$, respectively. If the query is in an epoch such that the adversary has corrupted the epoch key *or* the epoch belongs to list $\mathcal{L}$, then the current challenge ciphertext must be updated to the next epoch using a threshold

or more of the generated update tokens and the new ciphertext is added to the list of honestly updated ciphertexts ($\mathcal{L}$).

- $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}$: queries to these oracles allows the corruption of a threshold number of tokens and epoch secret key respectively. The restriction for both oracles is that the adversary's query must be from an epoch preceding the challenge-epoch e. Additionally, if an adversary queries the corrupt-token oracle for server S^j, not in the queried epoch server committee $S_{e'}$ then the corrupt-token oracle returns a failure symbol $\perp$.

5.3.3 Security Game

After the initialisation phase which outputs a global state (see Figure 5.3) and with challenge public key pk_e, the adversary queries the oracles in Figures 5.2 and 5.3. They output a challenge message m' and ciphertext C' in the queried epoch e. Before proceeding, the challenger must check that the given message and ciphertext are valid (belongs to $\mathcal{MSP}, \mathcal{CSP}$ respectively). Otherwise, the challenger aborts the game and returns $\perp$.

Moving forwards the challenger randomly chooses bit $b \in \{0, 1\}$ and outputs an encryption of m' or an update of ciphertext C' respectively. That is, the resulting output is a challenge ciphertext $C^{(b)}$ such that for $b = 0$ the ciphertext is from fresh encryption and for $b = 1$ the ciphertext is generated by a version of the update algorithm, denoted $\mathsf{UpdateCh}$.[2] The global state must be updated by the challenger, especially the set of lists $\mathbf{L}$. Equipped with a challenge output $C^{(b)}$ and public parameters, the adversary can query the oracles again before outputting a guess bit $b' \in \{0, 1\}$. The adversary succeeds in the security experiment if they satisfy certain winning conditions and successfully guess the correct bit ($b' = b$).

Preventing Trivial Wins and Ciphertext Updates Following Figure 5.4, we demonstrate the importance of the challenger recording lists $\mathcal{T}$ in the corrupt-token oracle, and list $\mathcal{C}^*$ in the update oracle. Without restrictions of the corrupt-token oracle: if an adversary $\mathcal{A}$ corrupts t or more tokens $\{\Delta^k_{e+1}\}_{k \geq t}$ from the corresponding server committee S_{e+1}, in an epoch proceeding the challenge epoch $\tilde{e}$, then $\mathcal{A}$ is capable

[2] Algorithm $\mathsf{UpdateCh}$ is used as compact notation, following the notation of [34], to denote the process of repeated application of the update algorithm from epoch $\{e_i, \ldots, e\}$.

$\underline{\mathcal{O}_{\mathsf{Upd}}(C_{e_i})}$
if $((e_i, C_{e_i}) \notin \mathcal{L}) \vee (e_i \notin \mathcal{C}^*)$ **then**
  return $\perp$
else
  for $e_l = \{e_{i+1}, \ldots, e\}$ **do**
    $C_{e_l} \leftarrow \mathsf{Upd}(pp, \{\Delta^k_{e_l}\}_{t \le k \le n}, C_{e_i})$
$C_e \leftarrow C_{e_l}$
return C_e
$\mathcal{L} \leftarrow \mathcal{L} \cup \{(e, C_e)\}$
if $\mathsf{isChallenge}(k_{e_i}, C_{e_i}) = \mathsf{true}$ **then**
  $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup \{e\}$

$\underline{\mathcal{O}_{\mathsf{Next}}(e)}$
$k_{e+1} := (pk_{e+1}, sk_{e+1}) \xleftarrow{\$} \mathsf{KG}(pp, e+1)$
$\{\Delta^j_{e+1}\}_{j \in [n]} \leftarrow \mathsf{TG}(pp, sk_e, k_{e+1}, S_{e+1})$
`Update GS to` $(pp, k_{e+1}, T_{e+1}, \mathbf{L}, e+1)$
if $(e \in \mathcal{K}) \vee (e, C) \in \mathcal{L}$ **then**
  $(e+1, C') \leftarrow \mathsf{Upd}(pp, \{\Delta^k_{e+1}\}_{|k| \ge t}, C)$
$\mathcal{L} \leftarrow \mathcal{L} \cup \{(e+1, C')\}$

$\underline{\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}(e', j)}$
if $(e' \ge e) \vee (S^j \notin S_{e'})$ **then**
  return $\perp$
else
  return $\Delta^j_{e'}$ `some` $j \in [n]$
  `Store tokens in a list` $T_{e'}$
if $|T_{e'}| \ge t$ `tokens have been corrupted in epoch` e' **then**
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{e'\}$

Fig. 5.2 Details of oracles an adversary $\mathcal{A}$ has access to during the security experiment of Definition 56 that are specific to the multi-server setting.

of trivially updating the ciphertext into the next epoch $(e+1)$ following Definition 54. Consequently, we place restrictions on calls to $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}$ and impose the winning condition in Figure 5.4. This condition states that the intersection of lists $\mathcal{K}$ and $\mathcal{C}^*$ must be empty, thus, the challenge epoch cannot belong to the set of epochs in which a threshold of update tokens have been obtained/inferred, and there doesn't exist a single epoch where the adversary knows both the epoch key (public and secret key components) and the (updated) challenge-ciphertext [92]. The distinction of DMUE

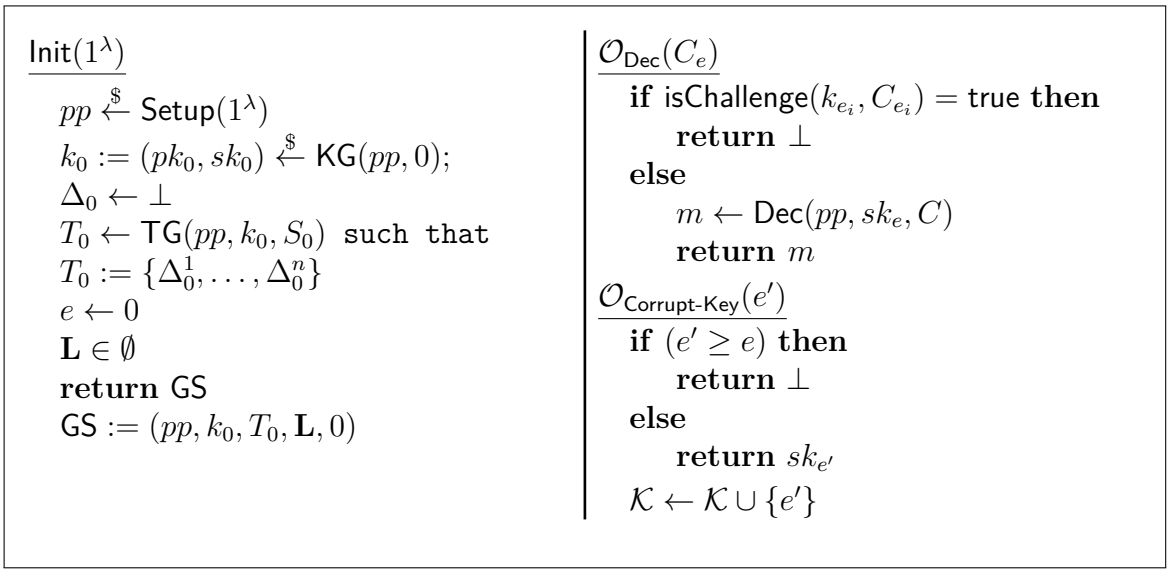

Fig. 5.3 The oracles an adversary has access to for the experiment capturing Definition 56 that remain unchanged from the single-server setting of a PKUE scheme.

security modelling from single-server PKUE is that list $\mathcal{T} \in \mathcal{C}^*$ does not record epochs in which token corruption occurred when the number of tokens corrupted is less than some threshold. That is, DMUE security modelling tolerates a certain level (below the threshold) of token corruption in any given epoch as less than the threshold of corrupted tokens does not provide the adversary with meaningful information.

The formal notion of ciphertext unlinkability in a DMUE scheme is as follows,

Definition 56 (MUE-IND-CCA Security)**.** *A dynamic multi-server updatable encryption scheme* (Π_{DMUE}) *as in Definition 54 is MUE-IND-CCA secure against update unlinkable chosen ciphertext attacks if for any PPT adversary* $\mathcal{A}$ *the following advantage is negligible over security parameter* λ*:*

$$\mathrm{Adv}^{\mathsf{MUE\text{-}IND\text{-}CCA}}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}IND\text{-}CCA},0}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}IND\text{-}CCA},1}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(1^\lambda)$$

for some negligible function $\mathsf{negl}(\cdot)$.

5.4 Integrity

In this Section, we define the security notion of *ciphertext integrity* (MUE-INT-CTXT) for a DMUE primitive. Informally, ciphertext integrity seeks to ensure that an adversary

$\underline{\mathsf{Exp}^{\mathsf{MUE\text{-}IND\text{-}CCA,b}}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda)}$

$\mathsf{GS} \xleftarrow{\$} \mathsf{Init}(1^\lambda)$; $\mathsf{GS} := (pp, k_0, T_0, \mathbf{L}, 0)$;
$\mathbf{L} := \{\mathcal{L}, \mathcal{K}, \mathcal{T}, \mathcal{C}^*\}$
$k_{e-1} \xleftarrow{\$} \mathsf{KG}(pp, e-1)$; $k_e \xleftarrow{\$} \mathsf{KG}(pp, e)$ `such that`
$k_{e-1} := (pk_{e-1}, sk_{e-1})$, $k_e := (pk_e, sk_e)$
$\{\Delta_e^j\}_{j\in[n]} \leftarrow \mathsf{TG}(pp, sk_{e-1}, k_e, S_e)$
$(m', C') \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(pp, pk_e)$
if $(m' \notin \mathcal{MSP}) \vee (C' \notin \mathcal{CSP})$ **then**
    return $\perp$
else
    $b \xleftarrow{\$} \{0,1\}$
    $C^{(0)} \xleftarrow{\$} \mathsf{Enc}(pp, pk_e, m')$ `and`
    $C^{(1)} \leftarrow \mathsf{UpdCh}(pp, \{\Delta_e^k\}_{|k|\geq t}, C')$
    $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup \{e\}$; $\tilde{e} \leftarrow \{e\}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(pp, C^{(b)})$
if $(\mathcal{K} \cap \mathcal{C}^* = \emptyset)$ **then**
    return b'
`Else abort.`

Fig. 5.4 The security experiment for MUE-IND-CCA-security of a DMUE scheme. Let $\mathcal{O} = \{\mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}, \mathcal{O}_{\mathsf{Next}}, \mathcal{O}_{\mathsf{Upd}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Token}}\}$ denote the set of oracles that adversary $\mathcal{A}$ calls during the experiment, where the latter three oracles capture the multi-server aspect of a DMUE scheme.

cannot produce a ciphertext that correctly decrypts to a valid message, so an adversary should be incapable of *forging* a ciphertext that will correctly decrypt to a message $m \neq \perp$ for some $m \in \mathcal{MSP}$. It is important that we model ciphertext integrity to ward against possible collusion or corruption of t or more servers in a given epoch committee, tasked with updating the ciphertext. This is especially important for the DMUE setting where we do not trust servers and it is likely a number of servers in a given committee may act dishonestly.

By design, DMUE is a *deterministic update* PKUE scheme, therefore, the stronger notion of ciphertext integrity can be satisfied since a challenger is easily able to record challenge-ciphertext updates and prevent decryption of these ciphertexts. Conversely, probabilistic update UE schemes like Π_{PKUE} from Chapter 3 require ciphertexts to be re-randomised when updated. Thus, an adversary can corrupt an update token and simply create various valid ciphertexts by manually updating an old ciphertext into

the new epoch, in turn creating a forgery of a ciphertext. This trivial attack occurs because the challenger has no way to track the updated ciphertext since it has been re-randomised. As a consequence, probabilistic UE schemes can only attain a notion of plaintext integrity, and ciphertext integrity is reserved for deterministic schemes [80].

The security experiment for ciphertext integrity (Definition 57), formally presented in Figure 5.5, starts with an initial setup run by the challenger as in Figure 5.2. Next, the challenger asks an adversary ($\mathcal{A}$) to output a ciphertext after calls to the oracles $\mathcal{O} := \{\mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Next}}, \mathcal{O}_{\mathsf{Upd}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Token}}\}$ all of which are described in Figure 5.2 and Figure 5.3. However, the update oracle requires slight adaptations. In particular, an adversary is only allowed to query updates of *honestly* generated ciphertexts to prevent trivial forgeries.

At a high level, the challenger is required to maintain an additional list $\mathcal{L}^*$, itself an extension of list $\mathcal{L}$, which updates a ciphertext (C) contained in $\mathcal{L}$ to the next epoch ($e+1$) whenever an adversary has corrupted at least a threshold (t) number of update token Δ^k_{e+1}. That is, whenever $(e+1) \in \mathcal{T}$. More specifically,

$\mathcal{L}^* \leftarrow \mathcal{L}$
for $(e, C) \in \mathcal{L}$ **do**
  $\mathcal{L}^* \leftarrow \mathcal{L}^* \cup \{(e, C)\}; C_{e_{i-1}} \leftarrow C; e_i \leftarrow e+1$
  if $e_i \in \mathcal{T}$ **then**
    $C_{e_i} \leftarrow \mathsf{Upd}(pp, \{\Delta^k_{e_i}\}_{t \le k \le n}, C_{e_{i-1}})$
    $\mathcal{L}^* \leftarrow \mathcal{L}^* \cup \{(e_i, C_{e_i})\}; e_i \leftarrow e_{i+1}$

For concreteness, the adaptation to the update oracle is detailed as follows:

$\underline{\mathcal{O}_{\mathsf{Upd}}(C_{e_i})}$
if $((e_i, C_{e_i}) \notin \mathcal{L}) \vee (e_i \notin \mathcal{C}^*)$ **then**
  return $\perp$
else
  for $e_l = \{e_{i+1}, \ldots, e\}$ **do**
    $C_{e_l} \leftarrow \mathsf{Upd}(pp, \{\Delta^k_{e_l}\}_{t \le k \le n}, C_{e_i})$
return C_e
$\mathcal{L}^* \leftarrow \mathcal{L}^* \cup \{(e, C_e)\}$

Once an adversary queries the oracles and outputs a forgery attempt, the challenger checks whether winning conditions are met. First, the challenger decrypts the output

$\underline{\mathsf{Exp}^{\mathsf{MUE\text{-}INT\text{-}CTXT}}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda)}$

$\mathsf{GS} \xleftarrow{\$} \mathsf{Init}(1^\lambda)$; $\mathsf{GS} = (pp, k_0, T_0, \mathbf{L}, 0)$; $\mathbf{L} := \{\mathcal{L}, \mathcal{L}^*, \mathcal{K}, \mathcal{T}, \mathcal{C}^*\}$
$C' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(pp)$
if $(\mathsf{Dec}(pp, sk_{e_{\mathsf{max}}}, C') = m' \neq \perp) \wedge ((e_{\mathsf{max}}, C') \notin \mathcal{L}^*) \wedge ((\mathcal{K} \cap \mathcal{T}) = \emptyset$ for epochs $\{e, \ldots, e_{\mathsf{max}}\})$ **then**
  return 1
`Else abort.`

Fig. 5.5 The security experiment for MUE-INT-CTXT-security of a DMUE scheme. Let $\mathcal{O} = \{\mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}, \mathcal{O}_{\mathsf{Next}}, \mathcal{O}_{\mathsf{Upd}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Token}}\}$ denote the set of oracles that adversary $\mathcal{A}$ calls during the experiment.

ciphertext to some message m', using the final secret epoch key ($sk_{e_{\mathsf{max}}}$) of the DMUE scheme. Winning condition checks are made to see whether the output message is valid ($m' \neq \perp$), and if $(e_{\mathsf{max}}, C') \notin \mathcal{L}^*$. Lastly, there cannot exist an epoch (e) from the challenge epoch of the security game to the final epoch (e_{max}) inclusive, such that the adversary has corrupted t or more update tokens *and* the corresponding epoch secret key. The final condition is essential to prevent *trivial forgeries*, otherwise, an adversary can exploit the sequential nature of update tokens, namely, tokens derived from the old and new epoch keys alone.[3] To see the attack, if the adversary corrupts a secret key sk_e and a threshold or more tokens for each epoch $\{e+1, \ldots, e_{\mathsf{max}}\}$, then they are capable of manually determining $sk_{e_{\mathsf{max}}}$ and generating valid forgeries of a ciphertext.

Given the informal description of how the ciphertext integrity security notion is captured for a DMUE scheme, we formally present the security experiment required in the ensuing definition.

The formal notion of ciphertext integrity in a DMUE scheme is as follows.

Definition 57 (MUE-INT-CTXT Security)**.** *A multi-server public-key updatable encryption scheme Π_{DMUE} as in Definition 54 is MUE-INT-CTXT secure if for any PPT adversary $\mathcal{A}$, there exists a negligible function negl over security parameter λ, such that the adversary has the following advantage in winning,*

$$\mathsf{Adv}^{\mathsf{MUE\text{-}INT\text{-}CTXT}}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) := \Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}INT\text{-}CTXT}}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(1^\lambda).$$

[3] Sequential update tokens are typical in ciphertext-independent update UE schemes such as DMUE. See Section 3.2 for further details.

5.5 Our Construction

In this Section, we use Definition 54 as a basis for formalising a *generic* DMUE construction. We achieve this using *dynamic proactive secret sharing* (DPSS) [9, 86, 98] and single server PKUE (Definition 28 from Chapter 3) primitives as building blocks. Before going into detail about our construction, we present the formal definition of a DPSS protocol, as well as defining DPSS correctness and secrecy properties.

5.5.1 Construction Preliminaries

Dynamic proactive secret sharing (DPSS) [98] is an extension of traditional secret sharing (see Section 2.4) such that shares belonging to a committee of parties are refreshed after a while. This feature is important to security to support a key feature of DPSS, namely, dynamic changes in the parties within a committee that possess secret shares.

Definition 58 (DPSS Protocol)**.** *Given a dealer $\mathcal{D}$, a secret $s \in \mathcal{S}_\lambda$ for security parameter λ, $L \in \mathbb{N}$ periods, and a set of $\{n^{(i)}\}_{i\in[L]}$ authorised parties $P^i = \{P_1^i, \ldots, P_n^i\}$, a (t, n) dynamic proactive secret sharing scheme is a tuple of four PPT algorithms $\Pi_{DPSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Redistribute}, \mathsf{Recon})$ defined as follows:*

- **Share Phase**: $\mathcal{D}$ takes as input the secret s and performs the following steps non-interactively:

  1. $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$: a probabilistic algorithm that takes as input security parameter 1^λ and outputs public parameters pp, which are broadcast to all parties in P.
  2. $\mathsf{Share}(pp, s, i) \xrightarrow{\$} \{s_1^i, \ldots, s_n^i\}$: a probabilistic algorithm that takes as input the secret $s \in \mathcal{S}_\lambda$ and period i, outputting n secret shares $\{s_j^i\}_{j\in[n]}$, one for each party in P.
  3. Distribute s_j^i to party $P_j^i \in P^i$ for every $i \in [L]$ over a secret, authenticated channel.

- **Redistribution Phase**: the algorithm $\mathsf{Redistribute}$ takes as input consecutive periods $(i, i+1) \leq L$, the set of parties (P^i, P^{i+1}) and the vector of secrets $\{s_j^i\}_{j\in[n]}$ belonging to P^i, such that P^i need to refresh and communicate their vector of secret shares to the potentially different set of parties P^{i+1}. The output is a vector of secrets $\{s_{j'}^{i+1}\}_{j'\in[n]}$.

- **Reconstruction Phase**: In period i, any party in $P^i = \{P_1^i, \ldots, P_n^i\}_{i\in[L]}$ can participate in the following steps.

  1. Communication:
     (a) Each party $P_j^i, j \in [n]$ sends their share s_j^i over a secure broadcast channel to all other parties in P^i.
     (b) P^i parties independently check that they have received $(t-1)$ or more shares. If so, they proceed to the processing phase.
  2. Processing: Once P_j^i has a set of t' shares labelled S', they independently do the following:
     (a) $\mathsf{Recon}(pp, S', i) \rightarrow \{s, \perp\}$: a deterministic algorithm that takes as input the set S' of t' shares and outputs the secret s for period $i \in [L]$ if $t' \geq t$ or outputs abort $\perp$ otherwise.

The following two definitions are regarding the correctness and secrecy of a dynamic proactive secret sharing scheme Π_{DPSS}. We assume these properties hold when proving the correctness and security of a proposed construction presented in Chapter 5.

Definition 59 (DPSS Correctness)**.** Π_{DPSS} *is* correct *if* $\forall\lambda \in \mathbb{N}$ *and for all possible sets of* $\{n^{(i)}\}_{i\in[L]}$ *authorised parties* P^i*, given* $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$*; for all secrets* $s \in \mathcal{S}_\lambda$ *and any subset of* $t' \geq t$ *shares* S' *from* $\mathsf{Share}(pp, s, i) \xrightarrow{\$} \{s_1^i, \ldots, s_n^i\}$ *communicated by parties in* P^i*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathit{Recon}(pp, S', i) \neq s] \leq \mathsf{negl}(1^\lambda).$$

Definition 60 (DPSS Secrecy)**.** Π_{DPSS} *is* secret *if* $\forall\lambda \in \mathbb{N}$ *and for all possible sets of* $\{n^{(i)}\}_{i\in[L]}$ *authorised parties* P^i*, given* $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$*; for all secrets* $s \in \mathcal{S}_\lambda$ *and any subset of* $t' < t$ *shares* S' *from* $\mathsf{Share}(pp, s, i) \xrightarrow{\$} \{s_1^i, \ldots, s_n^i\}$ *communicated by parties in* P*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathit{Recon}(pp, S', i) \neq \perp] \leq \mathsf{negl}(1^\lambda).$$

Remark 7. We chose to focus on building our DMUE scheme from dynamic threshold secret sharing for continuity within this thesis (see Chapter 6). However, observe that we can easily extend the construction of DMUE to be built from an alternative multi-party functionality, namely, a version of multi-party computation (MPC) [37, 51].

5.5.2 Building DMUE

Recall, a DMUE primitive is designed for distributed ciphertext updates across multiple untrusted servers. A threshold of servers can reconstruct the whole update token (Δ_e) for a given epoch (e), using the corresponding server tokens. By design, the threshold is necessary to correctly update the ciphertext into a new epoch. Moreover, the set of servers in any given epoch is fluid to allow for the removal of corrupted servers and support the realistic nature of long-term secret storage in which servers may need to change. Critical to security in this setting, it is vital to prevent a threshold or more servers overlapping in two consecutive epoch committees. Else, a collusion of these t overlapping servers in epochs $\{e, e+1\}$ means tokens for both epochs can be manually determined and further information can therefore be inferred regarding the sharing process, which we discuss in more depth after presenting our generic construction. To be clear, a DMUE scheme *can* tolerate $l < t$ servers belonging to both epochs.

Intuitively, DPSS (Definition 58) is an ideal building block candidate since the techniques used cater to changes in the shareholders, achieved via a handover process from one epoch to the next. Additionally, it is required in a DPSS protocol that the secret is re-shared in every period in such a way that the shares from different windows of time cannot be combined to recover the secret. The only way to recover the secret is to obtain enough shares from the *same* period, a task which the literature [71] assumes is beyond the adversary's grasp and the redundancy of sharing allows robustness in the periods of the scheme. We incorporate the aforementioned techniques into the design of our DMUE construction.

High-Level Idea The key idea of our construction is that we use a single-server PKUE scheme and share the update token using a threshold secret sharing protocol. Intuitively, the update token in our construction will be formed from the current and preceding epoch keys, such that the data owner ($\mathcal{D}$), taking the position of the dealer in the DPSS scheme, distributes a vector of *token shares* $\{\Delta_{e_i}^j\}_{j\in[n]}$ to the set of n servers $S_{e_i} := \{S_{e_i}^1, \ldots, S_{e_i}^n\}$ for current epoch $e_i, \forall i \in \mathbb{N}$. Token share generation will take place after $\mathsf{UE.TG}$ is run by $\mathcal{D}$ and this will occur for every epoch up to the final epoch (e_{max}).

The algorithm Upd will also be adapted to the multi-server setting in line with Definition 54, such that a threshold of t or more servers in set $S_{e_{i+1}}$ are required to reconstruct the update token $\Delta_{e_{i+1}}$ and then independently perform the update process in the classical

PKUE sense. Crucial to security, a key point clarified in Remark 8 is that the set of servers in consecutive epochs may overlap, and so they should not be able to learn the shares of the old or new epochs even though they participate in the redistribution process.

For ease of defining a generic construction, we design the scheme such that the *dynamic* feature is achieved in a trivial way, and does not use the DPSS techniques to evolve server committees. In other words, we do not trust the servers and assume the server committees for each epoch are selected by data owner $\mathcal{D}$ in some way. However, below Definition 61 we will make practical considerations which allow for the servers in a given epoch to participate in the redistribution process of token shares in order to reduce the data owners' computational cost. More formally, a generic DMUE scheme is defined as follows:

Definition 61 (DMUE Generic Construction)**.** *Given a (t,n) dynamic secret sharing scheme* $\Pi_{SS} = (SS.Setup, Share, Redistribute, Recon)$ *from Definition 19 (Section 2.4) and a standard public-key UE scheme* $\Pi_{PKUE} = (UE.Setup, UE.KG, UE.Next, UE.Enc, UE.Dec, UE.Upd)$ *from Definition 28 (Section 3.3), a dynamic public-key multi-server updatable encryption scheme is defined by a tuple of six algorithms* $\Pi_{DMUE} = (Setup, KG, TG, Enc, Dec, Upd)$ *as follows,*

1. $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$: run $\mathsf{SS.Setup}$ and $\mathsf{UE.Setup}$ on input security parameter 1^λ to randomly output the public parameters $pp := (pp_{\mathsf{SS}}, pp_{\mathsf{UE}})$ respectively.

2. $\mathsf{KG}(pp, e_i) \xrightarrow{\$} k_{e_i} := (pk_{e_i}, sk_{e_i})$: given public parameters pp, run the probabilistic key generation algorithm $\mathsf{UE.KG}$ to output the public and private key pair $k_{e_i} = (pk_{e_i}, sk_{e_i})$ for epoch $e_i, i \in \mathbb{N}, i \leq \mathsf{max}$.

3. $\mathsf{TG}(pp, sk_{e_i}, k_{e_{i+1}}, S_{e_{i+1}}) \rightarrow \{\Delta^j_{e_{i+1}}\}_{j\in[n]}$: the data owner runs the PKUE token generation algorithm $\mathsf{UE.TG}$ to determine update token $\Delta_{e_{i+1}}$, followed by $\mathsf{Share}(pp_{\mathsf{SS}}, S_{e_{i+1}}, \Delta_{e_{i+1}}) \rightarrow \{\Delta^j_{e_{i+1}}\}_{j\in[n]}$. Next, the data owner securely distributes $\Delta^j_{e_{i+1}}$ to server $S^j \in S_{e_{i+1}}$, where $S_{e_{i+1}}$ is the committee of servers for new epoch e_{i+1}.

4. $\mathsf{Enc}(pp, pk_{e_i}, m) \xrightarrow{\$} C_{e_i}$: given public parameters and the epoch public key pk_{e_i}, the data owner runs the probabilistic encryption algorithm $\mathsf{UE.Enc}$ on message $m \in \mathcal{MSP}$ and outputs the ciphertext C_{e_i}.

5. $\mathsf{Dec}(pp, sk_{e_i}, C_{e_i}) \rightarrow \{m, \perp\}$: given public parameters and the epoch secret key, the owner is able to run the deterministic decryption algorithm $\mathsf{UE.Dec}$ in order to output message m or abort ($\perp$).

6. $\mathsf{Upd}(pp, \{\Delta^k_{e_{i+1}}\}_{k\in\mathbb{N},|k|\geq t}, C_{e_i}) \rightarrow C_{e_{i+1}}$: given any valid subset $S' \subseteq S_{e_{i+1}}$ of the epoch e_{i+1} committee of servers, such that $|S'| \geq t$, shareholders in S' can reconstruct the update token by running $\mathsf{Recon}(pp_{\mathsf{SS}}, \{\Delta^k_{e_{i+1}}\}_{k\geq t}) \rightarrow \Delta_{e_{i+1}}$. Individually they can then update the ciphertext using the update algorithm $\mathsf{UE.Upd}(pp_{\mathsf{UE}}, \Delta_{e_{i+1}}, C_{e_i}) \rightarrow C_{e_{i+1}}$.

Correctness In the following, we show that our construction Π_{DMUE} presented in Definition 61 satisfies correctness (Definition 55). Observe that by definition, the DPSS scheme Π_{DPSS} secret reconstruction algorithm Recon used in step 6 of the update process satisfies correctness following Definition 59. Moreover, the correctness of Π_{PKUE} is defined in Definition 29, Section 3.3.

Theorem 5 (Correctness of Construction)**.** *Π_{DMUE} is correct assuming the underlying public-key UE scheme Π_{PKUE} and the underlying secret sharing scheme Π_{SS} satisfy their respective definitions of correctness.*

Proof. Following Definition 55, Π_{DMUE} is correct if $\mathsf{Dec}(pp, sk_{e_{\mathsf{max}}}, C_{e_{\mathsf{max}}})$ outputs m with overwhelming probability, whereby $C_{e_{\mathsf{max}}}$ has been generated iteratively by the update algorithm Upd.

In fact, this means the decryption algorithm $\mathsf{UE.Dec}$ is run and outputs m on the same honestly generated inputs. Note, one of the inputs is an update of the ciphertext to the final epoch ($C_{e_{\mathsf{max}}}$). Therefore, we assume this ciphertext has been generated correctly by entering a reconstruction phase of the SS scheme, that is, $\mathsf{Recon}(pp_{\mathsf{SS}}, \{\Delta^k_{e_i+1}\}_{k\geq t})$ is run to output token Δ_{e_i+1}. In turn, the resulting token is input into $\mathsf{UE.Upd}(pp_{\mathsf{UE}}, \Delta_{e_i+1}, C_{e_i})$ such that $C_{e_{\mathsf{max}}}$ is output.

Let us assume instead that Recon and/or $\mathsf{UE.Upd}$ output $\perp$, contradicting both correctness assumptions, resulting in $\mathsf{UE.Dec}$ outputting $\perp$. In turn, the DMUE decryption algorithm Dec will also output $\perp$ instead of m, which violates correctness in Definition 55. However, the assumptions that the failure symbol $\perp$ is output by the reconstruction phase or PKUE update algorithm contradict our assumption that the SS and PKUE schemes satisfy correctness. Thus, using proof by contradiction we can conclude that the DMUE scheme Π_{DMUE} also satisfies correctness. □

Practical Considerations: In Definitions 54 and 61 the selection of epoch committees and generation of their respective update tokens arise from the data owner ($\mathcal{D}$). The advent of every epoch calls for $\mathcal{D}$ to generate token shares for the newly selected server committee. However, in Definition 61 we can consider the involvement of the epoch server committee as a more elegant and practical solution to sharing the computational cost of token generation. That is, we introduce a *redistribution phase* during the running of token generation (TG), following Definition 58. The redistribution will utilise the secret share handover techniques from DPSS literature to support changes in the servers from one epoch to the next as well as a refresh of the secret token shares. More formally,

$\underline{\mathsf{TG}(pp, sk_{e_i}, k_{e_{i+1}}, S_{e_{i+1}}) \rightarrow \{\Delta^j_{e_{i+1}}\}_{j\in[n]} :}$

1. $\mathsf{TG}(pp, sk_{e_0}, k_{e_1}, S_{e_1}) \rightarrow \{\Delta^j_{e_1}\}_{j\in[n]}$: the DMUE token generation algorithm is run in epoch e_0, as detailed in step 3 of Definition 61.

2. *Redistribution Phase:* To proactively redistribute token shares to a new epoch, the redistribution phase is run by data owner $\mathcal{D}$ and the committee of servers (S_{e_i}) in epoch $e_i, \forall i \in [1, \mathsf{max} - 1]$. The servers in S_{e_i} proceed to *mask* their individual secret token shares $\{\Delta^j_{e_i}\}_{j\in[n]}$ and securely distribute each masked share to the corresponding server $S^{j'} \in S_{e_{i+1}}$. The new epoch server committee can proceed to refresh the masked token shares to obtain the updated vector to tokens labelled $\{\Delta^{j'}_{e_{i+1}}\}_{j'\in[n]}$.

The *refresh* of token shares can be achieved during the running of token share generation described above, using a *handover* process in the underlying redistribution phase of the chosen concrete DPSS scheme (recall Definition 58). For instance, secret shares are refreshed in the Shamir-based [110] DPSS scheme of [9] in such a way that shareholders from the current committee *mask* their polynomial P with some polynomial Q, such that no party in this committee learns shares for new polynomial $P' := P + Q$ given to the next shareholder committee, and vice versa.

Consequently, care needs to be taken in the choice of DPSS scheme so as to preserve security, especially if there is a crossover between the old and new server committees. In line with the proposed DPSS scheme from the authors of [9], an overlap of *one* server possessing the same share in both committees is not a security issue, since

the threshold of the scheme is not violated, however, if the threshold is violated then security may be compromised.

To illustrate this process in the context of a concrete DMUE scheme (Π_{DMUE}), using the techniques from the DPSS scheme from [9] to mask polynomials, we would define the following reconstruction phase. Informally, given a degree t polynomial P_{e_i} that determines the token shares for server committee S_{e_i} (in line with Shamir's secret sharing using polynomial interpolation from Section 2.4), the servers in S_{e_i} collectively produce a *masking* polynomial Q. This is used by the data owner to generate the new epoch polynomial $P_{e_{i+1}} := P_{e_i} + Q$. Crucial to security, no server in the old epoch e_i can determine shares of the polynomial $P_{e_{i+1}}$ for the new epoch e_{i+1} and vice versa for servers in the new committee $S_{e_{i+1}}$.

Observe, the following crossover committee problem can occur. If there is an overlap of t or more servers $S^j = S^{j'}$ for some $S^j \in S_{e_i}$, $S^{j'} \in S_{e_{i+1}}$, and the server shares are the *same* (derived from polynomials P_{e_i}, $P_{e_{i+1}}$ respectively) then the technique of masking that we described does not work since the threshold of servers can collude to determine polynomial Q manually. Therefore, we must make the following stipulation if the crossover of servers is above the threshold to ensure the security (Definition 56, Section 5.3) holds in Definition 61.

Remark 8. If a threshold t or more servers, $S^j = S^{j'}$ for $S^j \in S_{e_i}$ and $S^{j'} \in S_{e_{i+1}}$ respectively, overlap in two consecutive server committees then we necessitate distinct token shares ($\Delta^j_{e_i} \neq \Delta^{j'}_{e_{i+1}}$).

Alternatively, if we were to consider a concrete DMUE scheme in the decentralised setting (Blockchain technology), we could utilise techniques developed by the authors of [98]. In [98] they propose a Shamir-based DPSS scheme, with low communication complexity, that updates token shares with a masking method using asymmetric *bivariate polynomials*. Informally, the redistribution phase is based on efficient (t, n) bivariate 0-sharing and a *dimension-switching* technique in the handover process to *reduce* shares.

At a high level, a degree t polynomial $B(x, y)$ is generated by the data owner such that an update token $\Delta_{e_i} = B(0, 0)$ is shared with servers in committee S_{e_i}. During the handover to the next epoch server committee ($S_{e_{i+1}}$), committee S_{e_i} switches their sharing of Δ_{e_i} to a $2t$-degree version of $B(x, y)$ to *reduce* their shares. An important

result here for security is that an adversary would have to corrupt over $2t$ token shares across both epochs server committees. Masking of the reduced shares follows in the same manner as [9], except the masking polynomial generated is bivariate (Q(x,y)). Once the new epoch server committee $S_{e_{i+1}}$ have received their *reduced shares* of new update token $\Delta_{e_{i+1}}$ from the new $2t$-degree polynomial $B'(x, y) = B(x, y) + Q(x, y)$, they can switch dimensions back to a t-degree version of $B'(x, y)$ and receive their corresponding full token shares for $\Delta_{e_{i+1}}$.

5.6 Security Analysis

In this Section, we present the formal statements of security of the DMUE generic construction Π_{DMUE} from Definition 61. Namely, we demonstrate the ciphertext unlinkability (MUE-IND-CCA) and ciphertext integrity security notions, formalised in Definition 56 and Definition 57 respectively, are satisfied. Beforehand, we briefly discuss the necessary assumptions made in Theorem 6 to support the final proofs of security. The assumptions required are as follows.

Assumptions For the respective security definitions in Π_{DMUE} to hold we *assume* that the DPSS construction satisfies the definition of secrecy given in Definition 60. Secondly, we require the PKUE construction (that is, a single server DMUE scheme such that $t = n = 1$) to satisfy MUE-IND-CCA security and MUE-INT-CTXT security (Definitions 56 and 57 respectively). To be clear, security modelling with respect to a single server DMUE scheme inherits the description of lists and the oracles in Section 5.3 and Section 5.4, such that $k = t = n = 1$.

5.6.1 Proof of Security

In the following, we prove the statements of security regarding our DMUE construction. Namely, the ciphertext unlinkability notion (MUE-IND-CCA) formalised in Definition 56, and the ciphertext integrity notion (MUE-INT-CTXT) formalised in Definition 57.

Security Statements In general, we will separate our proofs into two cases: when an adversary corrupts less than the threshold number of token shares, *versus* an adversary that corrupts a threshold or more token shares. In each case, we can rely on the security of the underlying building blocks. Now, we formally prove that Π_{DMUE} satisfies ciphertext unlinkability and ciphertext integrity security notions, in that order.

Theorem 6. *Assume that* Π_{DPSS} *satisfies secrecy and suppose that* Π_{PKUE} *is a public-key updatable encryption scheme satisfying* MUE-IND-CCA *security for* $t = n = 1$*. Then* Π_{DMUE} *is a* MUE-IND-CCA *secure scheme.*

Proof. Following Definition 56, we want to show that there exists some negligible function negl under security parameter λ such that

$$\mathrm{Adv}^{\text{MUE-IND-CCA},b}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) \leq \mathsf{negl}(1^\lambda). \tag{5.1}$$

given the security experiment detailed in Section 5.3, Figure 5.4. To prove Equation 5.1, we must focus on two separate cases: first when an adversary $\mathcal{A}$ has corrupted $l < t$ token shares in the corresponding security game epoch $\tilde{e}$ and second when $\mathcal{A}$ has corrupted $l \geq t$ token shares. Following the assumptions in Theorem 6, we note that for either scenario we also assume the adversary's challenge message and ciphertext (m', C') were created in some epoch $e < \tilde{e}$ before the current epoch $\tilde{e}$, otherwise, the security experiment will output $\perp$.

Case $(\mathbf{l < t})$: Recall that *secrecy* is satisfied in the DPSS scheme Π_{DPSS}, the formal definition of which is detailed in Section 5.5. Consequently, an adversary has too few token shares from epoch $\tilde{e}$ to reconstruct the secret update token $\Delta_{\tilde{e}}$. In the case that the challenger randomly chose bit $b = 1$ (for $b = \{0, 1\}$) $\mathcal{A}$ cannot manually update their challenge ciphertext C' to ciphertext $C'_{\tilde{e}} := C^{(1)}$ due to the secrecy property. Moreover, if $\mathcal{A}$ queries oracle $\mathcal{O}_{\mathsf{Upd}}(C')$ to update the challenge ciphertext iteratively via epochs $\{e + 1, \ldots, \tilde{e}\}$, as detailed in Figure 5.2, $\mathcal{A}$ is still incapable of winning the experiment as the update oracle will add $\tilde{e}$ to the list of challenge-equal epochs $\mathcal{C}^*$ and winning conditions $(\mathcal{K} \cap \mathcal{C}^*) = \emptyset$ mean that $\perp$ is output.

Therefore, $\mathcal{A}$ is reduced to guessing bit b (in this case $b = 1$) which results in the advantage

$$\mathrm{Adv}^{\text{MUE-IND-CCA},1}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\text{MUE-IND-CCA},1}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1] - \tfrac{1}{2}| \leq \mathsf{negl}(1^\lambda).$$

If the challenger randomly chose bit $b = 0$, $\mathcal{A}$ would either have to query the epoch secret key corruption oracle to obtain $sk_{\tilde{e}}$ to manually decrypt the ciphertext $C^{(0)}$, or make calls to the decryption oracle. The assumed security of Π_{PKUE} is essential in this instance to prevent trivial wins. We note that both of the named oracles are detailed in Figure 5.3. The former scenario requires $\mathcal{A}$ query oracle $\mathcal{O}_{\mathsf{Corrupt\text{-}Key}}(\tilde{e})$ which results in output $\perp$ to prevent trivial wins. The latter scenario means $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{Dec}}(C^{(0)})$

which will result in output $\perp$ due to the decryption oracle conditions. Specifically, the first condition of the $\mathsf{isChallenge}$ predicate (Definition 23, Section 3.2) is satisfied, since $C^{(0)}$ is a challenge ciphertext and $\perp$ is output. Note that the output ($\perp$) does not inform the adversary whether or not the ciphertext is derived from fresh encryption in epoch $\tilde{e}$ or an update from a prior epoch.

Therefore, $\mathcal{A}$ is reduced to guessing bit b (in this case $b = 0$) which results in the advantage

$$\mathrm{Adv}^{\mathsf{MUE\text{-}IND\text{-}CCA},0}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}IND\text{-}CCA},0}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1] - \tfrac{1}{2}| \leq \mathsf{negl}(1^\lambda).$$

Consequently, Equation 5.1 holds when $l < r$.

Case ($\mathbf{l \geq t}$) : oracle $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}$ stipulates that the challenger needs to add the challenge epoch $\tilde{e}$ to list $\mathcal{T}$ (Figure 5.2). Crucially, epochs in $\mathcal{T}$ are incorporated into list $\mathcal{C}^*$ which captures all *challenge-equal* epochs. Thus, epoch $\tilde{e}$ belongs to $\mathcal{C}^*$ and winning conditions in our security experiment prevent trivial wins. That is, the intersection of sets ($\mathcal{K} \cap \mathcal{C}^*$) must be empty to prevent a trivial win from occurring. See the end of Section 5.3 for more depth on trivial wins.

If $t = n = 1$ we can rely on the assumed security of Π_{PKUE}, namely, Definition 56 is satisfied. Therefore, in the case of ($l \geq t$) and for either choice of $b = \{0, 1\}$, $\mathcal{A}$ is reduced to guessing bit b which results in the advantage

$$\mathrm{Adv}^{\mathsf{MUE\text{-}IND\text{-}CCA},b}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}IND\text{-}CCA},b}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1] - \tfrac{1}{2}| \leq \mathsf{negl}(1^\lambda).$$

Given the above, we can conclude that the Equation 5.1 is satisfied for any number (l) of corrupted tokens. □

Now we present a formal statement of security for ciphertext integrity.

Theorem 7. *Assume the* Π_{DPSS} *satisfies secrecy and suppose that* Π_{PKUE} *is a public-key updatable encryption scheme satisfying* MUE-INT-CTXT *security such that* $t = n = 1$. *Then* Π_{DMUE} *is a* MUE-INT-CTXT *secure scheme.*

Proof. Following Definition 57, we want to show that there exists some negligible function negl under security parameter λ such that

$$\mathrm{Adv}^{\mathsf{MUE\text{-}INT\text{-}CTXT}}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) \leq \mathsf{negl}(1^\lambda). \tag{5.2}$$

given the security experiment detailed in Section 5.4, Figure 5.5.

Once again, to prove Equation 5.2 we must focus on instances where adversary $\mathcal{A}$ has corrupted less than a threshold of token shares or at least a threshold of token shares. Following the assumptions in Theorem 7, we note in proving both possibilities we also assume the adversary's challenge ciphertext (C') was created in some epoch $e < e_{\mathsf{max}}$.

Case ($\mathbf{l < t}$) : Similarly to the ciphertext unlinkability proof, we rely on the assumption that *secrecy* is satisfied in the DPSS scheme Π_{DPSS}. Therefore, in this instance whereby an adversary has too few token shares for the final epoch e_{max}, they are incapable of reconstructing the secret update token $\Delta_{e_{\mathsf{max}}}$. Secrecy is important as it prevents $\mathcal{A}$ from manually reconstructing the update token $\Delta_{e_{\mathsf{max}}}$ and using it to update their challenge ciphertext C' to a valid ciphertext for epoch e_{max}.

Instead, $\mathcal{A}$ may query the update oracle $\mathcal{O}_{\mathsf{Upd}}(C')$, from Section 5.4, to update their challenge ciphertext iteratively from $\{e+1, \ldots, e_{\mathsf{max}}\}$, as detailed in Figure 5.2. The final epoch and updated ciphertext $(e_{\mathsf{max}}, C'_{e_{\mathsf{max}}})$ will be added to the list $\mathcal{L}^*$. This list is incorporated into the winning conditions of Figure 5.5, detailed in Section 5.4. Thus, the output will be $\perp$ and $\mathcal{A}$ remains incapable of winning the experiment.

Furthermore, $\mathcal{A}$ may attempt to corrupt the final epoch secret key $sk_{e_{\mathsf{max}}}$ from $\mathcal{O}_{\mathsf{Corrupt\text{-}Key}}$ to manually decrypt a ciphertext encrypted under $pk_{e_{\mathsf{max}}}$ to determine if the resulting message $m' = \perp$ or not. In the latter case, an adversary has found a valid ciphertext forgery, however, the corrupt key oracle ensures that epoch e_{max} has been added to list $\mathcal{K}$ which violates the winning conditions. Therefore, for scenario ($l < t$) we have

$$\mathrm{Adv}^{\mathsf{MUE\text{-}INT\text{-}CTXT}}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = \Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}INT\text{-}CTXT}}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(1^\lambda).$$

Case ($\mathbf{l \geq t}$) : oracle $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}$ stipulates that the challenger needs to add the final/challenge epoch e_{max} to list $\mathcal{T}$ (Figure 5.2). In turn, the final winning condition in our security experiment is not satisfied, namely, the intersection of sets $(\mathcal{K} \cap \mathcal{T}) \neq \emptyset$. Consequently, for scenario ($l \geq t$) we have

$$\mathrm{Adv}^{\mathsf{MUE\text{-}INT\text{-}CTXT}}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = \Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}INT\text{-}CTXT}}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(1^\lambda).$$

Lastly, if $t = n = 1$ we can rely on the assumed security of Π_{PKUE}, namely, Definition 57 is satisfied. Given all of the possibilities mentioned above, we can conclude that Equation 5.2 is satisfied for any number (l) of corrupted tokens. □

5.7 Summary and Outlook

In summary, our first contribution was a novel definition for dynamic multi-server updatable encryption (DMUE), designed to distribute the ciphertext update process across multiple servers to cope with the single point of failure problem inherent in standard PKUE.

Second, we modelled two new security notions for the DMUE primitive. First, a notion capturing ciphertext unlinkability against chosen ciphertext attacks (MUE-IND-CCA). In essence, unlinkability means that freshly generated and updated ciphertexts are indistinguishable. Second, a notion capturing integrity against ciphertext forgery attacks (MUE-INT-CTXT) which aims to prevent an adversary from forging a ciphertext that decrypts to a valid message. Intuitively, ciphertext integrity is a notion of security that guarantees the accuracy of the encrypted message.

Our third contribution was proposing a generic DMUE scheme constructed from single-server PKUE and dynamic proactive secret-sharing building blocks. We extend the final contribution further with a discussion regarding the practicalities of a concrete scheme. Lastly, we analyse the security of the generic DMUE construction, demonstrating that MUE-IND-CCA and MUE-INT-CTXT security is satisfied.

An exciting area of future work is to explore our DMUE use-case to store ciphertexts using Blockchain technology, especially given there is a potential synergy with the work of [100]. We believe it is worth exploring the gap between our contribution and the work of [100] who introduced the first Blockchain-based long-term time-stamping (BLTTS) scheme supporting cryptographic algorithm renewal on both the client (data owner) and server sides. Informally, time-stamping services in the decentralised setting are necessary when the time-stamping authority (TSA) is used to provide evidence of the existence of data at a given time. For future work, we believe a BLTTS scheme could be utilised to construct DMUE in the decentralised setting.

In general, we are of the opinion it is worthwhile to develop concrete DMUE schemes to analyse the efficiency, costs and security levels attained in comparison to single server PKUE. In light of our use of DPSS as a building block to construct a DMUE scheme, it would be interesting to interpret the security framework defined in this Chapter through a game-theory lens. That is, we would like to consider how we can model *rational servers*, possessing token shares derived from a rational secret sharing

scheme (RSS [69]), that are incentivised to mislead or deviate from behaving honestly. In doing so, we can further understand how practical it would be to apply DMUE in the real world.

Chapter 6

Fair and Sound Secret Sharing from Homomorphic Time-Lock Puzzles

Contents

This Chapter introduces a generic construction of a rational secret-sharing scheme for non-simultaneous communication. Our construction, labelled FRSS, is built to satisfy essential secret-sharing properties known as fairness and soundness. Crucial to attain both properties at the same time, we employ a time-delay mechanism called a homomorphic time-lock puzzle, and we can prove that this generic FRSS construction satisfies desirable privacy-based properties. Additionally, we propose an instantiation of FRSS which allows us to consider the efficiency of a concrete construction. Part of this work appears in [83], which is joint work with Elizabeth A. Quaglia, published and presented at ProvSec 2020.

6.1 Introduction

Recall from the Introduction that a key theme of this Thesis is *time*. The past three Chapters have been related to extending the functionality and security of the time-

based primitive Updatable Encryption (UE), and in Chapter 5 we achieved a generic construction of a multi-server variant of UE using *secret-sharing* as a building block. Secret sharing (SS) protocols are used to distribute a secret among a set of authorised parties such the parties can reconstruct the secret at a later time. Privacy preservation is the main security goal, however, additional properties are desirable in secret sharing such as *fairness* and *soundness*. Instead of time updates as in UE, *time delay* is a useful functionality in the context of secret sharing to attain certain properties.

Overview of this Chapter

First, in Section 6.2 we introduce *rational* secret sharing (RSS) which is a SS protocol assuming parties holding shares have strategies and behave in a way to benefit themself above other parties in the scheme. We will formally define an RSS, followed by definitions of the traditional SS properties of correctness and secrecy. Moreover, we formally introduce two important notions known as *fairness* and *soundness* which are the core focus of this Chapter.

Moving forwards, we provide a comprehensive background on SS protocols, in which we differentiate between two distinct strains of SS depending on the communication between parties holding shares. Namely, *simultaneous* versus *non-simultaneous* communication. Furthermore, we explore the current tools used in the literature to achieve certain security properties. In doing so, we motivate our work which studies non-simultaneous RSS protocols seeking to attain the properties of fairness and soundness using tools such as time delay and side information to do so.

In Section 6.3 we formally introduce a generic construction of a rational secret-sharing scheme, designed for asynchronous communication between rational players, which satisfies the important properties of soundness and fairness. Informally, soundness ensures that players participating in the scheme can check that a reconstructed secret is correct, and in doing so, they avoid being misled by deviant players. To ensure soundness, we will use the common technique of introducing *side information* called a *checking share*.

The property of fairness essentially means that no player learns of the secret during reconstruction before other players, even if they deviate. Given that we are focusing on non-simultaneous communication between players in an RSS, it is important to note that fairness is more challenging to achieve. Even more of a challenge, the

side-information used to ensure soundness can provide an additional way for players to reconstruct the secret before others. Therefore, in line with the literature, we will incorporate a time delay to ensure information is not realised until a certain time. Our novel choice of a time-delay tool is a homomorphic time-lock puzzle [97], which we will discuss in greater detail before the formal presentation of our generic construction.

In Section 6.4 we analyse the security of our generic construction of an RSS scheme. To prove our construction satisfies definitions of correctness, secrecy, fairness, and soundness, we begin this Section with preliminary game theoretical notions essential to proving our security statement. Categorically, we will discuss the concept of utility functions which captures the preferences players have concerning the outcomes of a game (in terms of real numbers). Moreover, we introduce a state of Nash equilibrium with respect to utility functions, and a definition essential to proving the soundness of an RSS construction in which players possess side information.

In Section 6.5, we present an instantiation of our generic construction using concrete building blocks. That is, our concrete scheme uses a multiplicative variant of the HTLP scheme proposed in [97]. Lastly, we conclude our work by discussing the efficiency of our concrete construction, and we demonstrate that our result is more efficient in comparison to the literature on RSS using time-delay functionalities.

6.2 Rational Secret Sharing

In this Section, we begin by recalling how secret sharing works. Next, we introduce the syntax used in this Chapter, recalling Section 2.4 which details threshold secret sharing. Further, we formally define a rational secret sharing (RSS) protocol plus relevant properties including correctness, secrecy, fairness and soundness. Lastly, we discuss the related work on RSS and the tools used in RSS literature to achieve fairness and soundness.

Threshold secret sharing (SS) schemes are an important primitive used in a variety of settings including multiparty computation [30, 37], attribute-based encryption [64, 121], and threshold cryptography [11, 43]. Recall, we formally detailed threshold secret sharing in Definition 19, Section 2.4. At a high level, secret sharing supports the splitting of a secret into shares such that the secret can be reconstructed by a threshold number of mutually distrustful parties. Crucially, knowledge of fewer than the threshold

number of shares reveals nothing about the secret [16, 110]. That is, a trusted dealer splits the secret into shares and distributes one to each authorised party. Parties then communicate and process their collective shares in a reconstruction phase. During the communication phase, parties broadcast their shares in one of two ways: *simultaneously* or *non-simultaneously*. That is, with or without synchronicity.

Properties of SS schemes are better understood and easier to guarantee in the simultaneous setting [40], however, such schemes are difficult to implement in practice. Hence, our attention was drawn to researching how non-simultaneous schemes can attain a desirable level of security.

Typically, non-simultaneous SS [56, 84, 85] protocols consist of *rounds*, whereby one round of the reconstruction phase simply translates to a capped period in which parties have the opportunity to communicate their share. Key to the design is that parties learn the secret is reconstructed when they regenerate some publicly known value in a so-called *revelation* round. It is assumed that the round before revelation is the one in which the secret can be reconstructed, allowing parties to identify when they will reconstruct the correct secret.

One of the biggest challenges in non-simultaneous construction design is to ensure the final party in the communication process is *incentivised* to follow the protocol. The approach taken to solve this problem is to re-frame the security modelling in terms of the assumptions regarding parties holding shares. Note that there are two somewhat independent research areas in secret sharing in terms of modelling shareholders:

- Traditional schemes that consider honest or malicious parties [10, 11, 35, 61, 70, 87, 88, 94, 106, 116].
- *Game-theoretical* SS schemes [5, 40, 56, 62, 69, 96] in which parties are modelled as rational.

For applications of non-simultaneous secret sharing, we believe it is more suitable to view parties as *rational players* in the game-theoretic sense. Rational secret sharing (RSS) [69] is a protocol considering the problem of secret sharing assuming players prefer to learn the secret over not learning it, and secondly, prefer that as few as possible other players learn the secret. Categorically, we focus on *game-theoretical schemes* to tackle the issue of incentivising parties.

6.2.1 Formal Definitions

In an RSS scheme, a player is considered to be rational if they have a preference for the outcome of the reconstruction phase and a player's strategy, is to maximise their payoff from the outcome of the game. Depending on the scheme, the strategies of players in P may be the same or different.

Syntax Throughout this Chapter, we assume an honest non-interactive dealer D, a set of $n \in \mathbb{N}$ rational players $P = \{P_1, \ldots, P_n\}$ communicating non-simultaneously, and each round of the reconstruction phase is bounded by the time hardness parameter $\mathcal{T}$. The security parameter is labelled $\lambda \in \mathbb{N}$. We use the following notation for a deviating strategy σ_i' for player P_i, to signify when a player behaves differently from how they are meant to. That is, they do not follow the protocol. In addition, P_{-i} represents all players in P excluding player P_i, and σ_{-i} signifies the honest strategies of this set of $(n-1)$ players, P_{-i}.

Note, in Definition 19 (Section 2.4) players only participate in the reconstruction phase, therefore, we formalise an RSS scheme by defining the reconstruction phase only in the following definition.

Definition 62 (Rational Secret Sharing Reconstruction Phase)**.** *Let us define an RSS scheme to be a tuple of three PPT algorithms* $\Pi_{RSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Recon})$*. The reconstruction phase* $\Gamma_{t,n}$*, in which* Recon *is run, is defined by* $\Gamma_{t,n} = (\Gamma, \overrightarrow{\sigma})$ *where* Γ *is the game to be played by players during the reconstruction phase and* $\overrightarrow{\sigma} = (\sigma_1, \cdots, \sigma_n)$ *denotes the strategy profile of the players in* P *prescribed by the dealer* D *during the sharing phase for that scheme.*

The outcome of the phase for all players is defined by the n-dimensional vector

$$\overrightarrow{\omega}((\Gamma, \overrightarrow{\sigma})_{t,n}) = (\omega_1, \ldots, \omega_n)$$

where ω_i *refers to the outcome of the phase for player* P_i*.*

The outcome ω_i alludes to whether player P_i learns the entirety of s, nothing of s, is misled into learning a fake secret s' or aborts the reconstruction phase altogether ($\perp$). It is important to note that the outcome of the phase depends on the strategy of the player.

Correctness and Secrecy Definitions for *correctness* and *secrecy* for an RSS scheme (Π_{RSS}) directly follows from the SS definitions presented in Section 2.4 (Definitions

20 and 21 respectively). In essence, the correctness of Π_{RSS} ensures that players reconstructing a secret from a threshold or more ($(k-1) \geq r$ for threshold r) of honestly generated shares will reconstruct the correct secret with overwhelming probability. Furthermore, an RSS scheme (Π_{RSS}) satisfies the secrecy property if players reconstruct a secret from less than a threshold ($(k-1) < r$ for threshold r) of honestly generated shares will reconstruct a value not equal to $\perp$ with negligible probability.

Fairness and Soundness Properties In the ensuing, we define fairness and soundness in the context of RSS literature [40]. One of the fundamental properties of secret sharing is *fairness* [117], which guarantees that no player has an advantage in the protocol over other players. More formally,

Definition 63 (Fairness)**.** *The reconstruction phase $\Gamma_{t,n}$ is* completely fair *if for every arbitrary alternative strategy σ_i' followed by player P_i for some $i \in [n]$, there exists a negligible function negl in the security parameter λ such that the following holds:*

$$\Pr[\omega_i(\Gamma, (\sigma_i', \sigma_{-i})) = s] \leq \Pr[\omega_{-i}(\Gamma, (\sigma_i', \sigma_{-i})) = s] + \mathsf{negl}(1^\lambda).$$

That is, the probability of player P_i learning the secret when they deviate from their prescribed strategy in phase $\Gamma_{t,n}$ (but all other players follow their prescribed strategies) is only ever negligibly more than the probability of the other players learning the secret too. Consequently, such a player has no real advantage in deviating from their strategy.

Another fundamental property of RSS is soundness. Simply put, the soundness of the reconstruction phase output means that the probability of players following the scheme outputting an incorrect secret when another player deviates from their strategy is negligible.

Definition 64 (Soundness)**.** *Reconstruction phase $\Gamma_{t,n}$ is* sound *if for every arbitrary alternative strategy σ_i' followed by player P_i for $i \in [n]$, there exists a negligible function negl in the security parameter λ such that the following holds:*

$$\Pr[\omega_{-i}(\Gamma, (\sigma_i', \sigma_{-i})) \notin \{s, \perp\}] \leq \mathsf{negl}(1^\lambda)$$

To re-emphasise, we believe that RSS is an excellent approach to capture more interesting scenarios and outcomes, such as how to incentivise players to participate honestly, and even how a scheme can penalise players for deviant play. Furthermore, modelling players as rational is not limited to the perhaps unrealistic assumption that

players always want to learn the secret above all else. Indeed, we will explore an emerging scenario in RSS that considers players with a preference to *mislead others* above learning the secret.

6.2.2 A Background in Secret Sharing

In RSS the possible outcomes of the game influence the players' strategies, as they seek to maximise their payoff. Security of the game requires the strategies of players to be in some form of equilibrium which motivates them to honestly communicate. Notably, achieving an equilibrium between players' strategies is the most natural way to demonstrate a fundamental property of SS schemes, called *fairness* (Definition 63) [46, 76].

A separate but equally important property in RSS literature is *soundness* [5, 40], which ensures players never reconstruct an *incorrect secret* except with negligible probability. In other words, honest players are guaranteed to output a correct value or a special abort symbol $\perp$. Soundness is becoming of emerging relevance in the non-simultaneous setting, especially in instances where rational players obtain a greater payoff from misleading other players compared to learning the secret.

Soundness (Definition 64) has been achieved in prior work [40] focusing on non-simultaneous communication as follows: before reconstruction begins, all players are given protocol-induced *side information* alongside their list of shares. They must assume that when a player aborts communication, the previous round was the revelation round. Even if a deviant player has aborted early, using this side information, honest players can check that they have the correct value after reconstruction. If not, they terminate the reconstruction altogether.

Unfortunately, achieving soundness this way can compromise fairness in the non-simultaneous setting since a deviant player can use the side-information to check whether they can abort early and learn the secret before honest players. That is, in a (t, n) threshold RSS scheme, the last player out of t-players to communicate can decide not to reveal their share and use all the other players' shares to reconstruct the secret. Thus, leaving the $(t-1)$ honest players with an insufficient number of shares to reconstruct the secret. The rational behaviour of all parties would therefore be to withhold their share.

To counteract this problem, and continue to ensure fairness, RSS literature suggests incorporating a *time-delay-based mechanism* to prevent deviant players from utilising the side information in the manner just described. As a consequence, players are incentivised to honestly participate in the reconstruction phase. We will use the two techniques mentioned, namely, protocol-induced side information and time delay, to achieve the properties of soundness and fairness (respectively) in an RSS construction. Before we formally define such a construction, we examine existing tools used, for the aforementioned techniques, in the literature.

Existing Methods to Achieve Fairness and Soundness In the following, we provide an extended examination regarding secret-sharing literature seeking to achieve soundness and fairness, separately focusing on the simultaneous and non-simultaneous settings.

(Simultaneous) To achieve fairness, [94] proposed that in addition to obtaining a share of the secret, each party possesses a check vector which they can use to verify the validity of other parties' shares, and a certificate vector, which is used to prove the validity of their share in the reconstruction phase. The dealer in the scheme chooses an indicator, which is a form of public information unrelated to the secret that must be reconstructed. In their scheme, the secret is hidden in a sequence of elements, such that the subsequent element of the sequence is the indicator and the rest of the elements are dummy secrets.

The authors of [70] continued the work of [94]. The scheme in [94] works under the assumption that all parties sharing is legitimate. In other words, their scheme only deals with external adversaries. Whereas [70] protects the secret from inside adversaries as well as unauthorised parties (outside adversaries) who are not legitimate shareholders.

In [88], a V-fair (t, n) SS scheme is proposed, where given n-parties, they have an equal probability of obtaining the secret, even if $V < (t/2)$ parties are dishonest. This is achieved by the dealer dividing the secret into multiple sub-secrets with different threshold values, and generating shares for each of the sub-secrets.

The authors of [35] showed that complete fairness cannot be achieved in *general*, without an honest majority. Intuitively, complete fairness means that an adversary can learn the output of secret reconstruction if, and only if, the honest parties learn the output too [4]. In the setting of secure two-party computation, if just one of the parties

is dishonest, there is no longer an honest majority, and so it was believed that no *non-trivial* function could be computed with complete fairness. However, the work of [61] demonstrated the existence of *some* non-trivial functions, based on cryptographic assumptions, which can be computed with complete fairness in the two-party setting.

In [61], the reconstruction phase is based on rounds, such that parties input a share of the secret into *some* function in every round and the round in which the party learns the secret depends on the value of their input (in contrast to standard protocols). If one party aborts after learning the secret in a round, and the second party has not yet received the function output, then the second party assumes the first learned the secret in the round they aborted and reconstructs the function output for that round independently.

The scheme of [116] extends the work in [61], achieving a more efficient scheme. At a high level, their reconstruction scheme provides complete fairness by hiding the secret in a sequence of secrets such that the validity of the shares can be verified and used to detect deviant parties.

In the simultaneous setting, both [62, 69] use some form of publicly known indicator and their respective protocols achieve fairness by demonstrating a form of Nash equilibrium [103] has been satisfied.

(Non-Simultaneous) The authors of [84] sought to ensure no information about the players' inputs is revealed until the round in which the secret is recovered, in which players are communicating non-simultaneously. They achieve this by introducing a new cryptographic tool called meaningful/meaningless encryption, where players are motivated to follow the scheme as they do not know whether a round of reconstruction is meaningful or not.

Following on from [84], [56] propose a scheme that does not require simultaneous broadcast channels or physical assumptions in both two-player and multi-player RSS scheme instantiations. The protocol follows a series of fake rounds, followed by a real round. That is, in the real round, every player learns the secret, and in the fake rounds, no information about the secret is revealed.

Players cannot know whether a round is real or fake. They identify the real round in the subsequent round, where they reconstruct public information in the form of a

flag/indicator (akin to the schemes of [70, 94] in the cryptographic model). Similarly, the authors of [85] use the same idea of players reconstructing an indicator.

The authors of [96] were the first to propose a fair RSS scheme that can tolerate *arbitrary* side-information, by proposing the use of *time-delay encryption* (TDE) [29, 99]. The basic idea of a TDE scheme is to encrypt a message such that it can only be decrypted after a specific amount of time has elapsed. We note that [96] works under the assumption that players prefer everyone to obtain the correct output over misleading others, therefore soundness is not an issue that needs to be addressed.

In more detail, the scheme in [96] employs a cryptographic memory-bound function[1] (CMBF) [1, 48] as a way to achieve *time delay* in the recovery of an encrypted sub-share of the secret. The fairness of their scheme is restored by setting the runtime of rounds of the secret sharing scheme to be less than the time it takes to decrypt the encrypted shares. Thus, there is no way for a deviant player to learn anything about the secret during a reconstruction round before they must decide whether to abort communication. In addition, proof of the sender's work in computing their message is sent.

Subsequently, [40] build upon the work of [96] by using *specific* protocol-induced side information to provide the first fair and sound RSS scheme in the non-simultaneous setting, additionally achieving independence from the preference (utility) of misleading.

Critically, all non-simultaneous RSS schemes mentioned suffer from a *computational bottleneck* as each player has to attempt to decrypt sub-share ciphertexts to reconstruct a share before proceeding to reconstruct the secret in the revelation round. We believe there is a more elegant approach to solving this problem if we employ a homomorphic functionality. Thus, we propose a time-delay mechanism called a *homomorphic time-lock puzzle* (HTLP) [97] which, when applied to an RSS construction, will enable the compact evaluation of encrypted sub-shares into just one ciphertext that needs decrypting to reveal the secret share for that round. Consequently, we can mitigate the computational bottleneck issue inherent to previous RSS schemes seeking to achieve soundness and fairness.

[1] A CMBF is a family of deterministic algorithms such that an efficiently generated key can decrypt the encrypted input, with a lower-bound on the number of memory-access steps to do so.

6.3 A Generic Construction of an FRSS Scheme

In this Section, we start by defining the tools required to build a generic construction of an RSS scheme satisfying fairness and soundness. Next, we formally present our construction, labelled an FRSS scheme, and additionally provide definitions of correctness and secrecy for the scheme.

To build a generic construction of a fair and sound RSS scheme (FRSS), we utilise a time-delay mechanism known as a homomorphic time-lock puzzle (HTLP). Informally, we aim to prevent a player, that chooses to deviate from their prescribed strategy and quit communication, from deriving the secret before the end of the round. If this occurs, then the other players should realise the deviant player has quit and output the result of the previous rounds' reconstruction.

That is, even if a player quits in a round and manages to learn the secret, the only case in which they can do so results in the honest players also learning the secret. Therefore, there is no advantage in a player deviating from their prescribed strategy.

High-Level Idea A *trusted dealer* splits the secret into shares and creates an additional share which is broadcast to all players, named the checking share. The rest of the shares are split into sub-shares, embedded into HTLPs and distributed to the corresponding players in such a way that the HTLP scheme can reconstruct the 'whole' share from them.

Intuitively, the checking share is used to verify the soundness of the secret that players reconstruct, and the delay provided by the HTLP scheme is used to guarantee fairness in the presence of a checking share for players communicating non-simultaneously. More specifically, the HTLP scheme embeds the sub-shares into puzzles that cannot be decrypted before a round of communication in the reconstruction phase has finished. Fairness is achieved by setting each round of communication to have an upper time bound less than the time to solve a puzzle. The formal definition of an HTLP is now presented.

6.3.1 Tools Required

A time-lock puzzle (TLP) [108] is a primitive used to provide the feature of *time delay* in a cryptographic scheme. Informally, TLPs embed a secret into a puzzle such that it

cannot be decrypted until a certain amount of time T has elapsed. The characteristics of a TLP are,

- *Fast puzzle generation*: the time t required to generate a puzzle $\mathcal{Z}$ must be $t << \mathcal{T}$, for a given (time) hardness parameter $\mathcal{T}$.
- *Security against parallel algorithms*: the encapsulated secret s is disguised within the puzzle $\mathcal{Z}$ for circuits of depth $< \mathcal{T}$, regardless of the size of the circuit.

A *homomorphic* time-lock puzzle (HTLP) scheme evaluates puzzles homomorphically using some operation, without the evaluator knowing the secret shares encapsulated within the corresponding puzzles. The resulting puzzle output contains the homomorphic evaluation of the input puzzles, enabling a more efficient way for decryptors to obtain the final output solution, as they can solve just one puzzle rather than solving all of the puzzles individually like in a standard TLPs and then evaluating a final solution.

In more detail, HTLPs are augmented TLPs allowing anyone to evaluate a circuit C over sets of puzzles $(\mathcal{Z}_1, \ldots, \mathcal{Z}_n)$ *homomorphically* using operation Ψ. What Ψ depends on the application the HTLP is being used for, such as addition, multiplication or XOR. Essential to the purpose of an HTLP, the evaluator does not need to know the secret values $(s_1, \ldots, s_n)$ encapsulated within the corresponding puzzles. The resulting output (a puzzle $\mathcal{Z}$) contains the circuit output $C(s_1, \ldots, s_n)$, and the hardness parameter $\mathcal{T}$ does not depend on the size of the circuit C that was evaluated (this is called compactness). More formally,

Definition 65 (HTLP)**.** *Let $C = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a class of circuits and let secret space $\mathcal{S}_\lambda$ be a finite domain for security parameter λ. A homomorphic time-lock puzzle (HTLP) with respect to C, $\mathcal{S}_\lambda$, and time hardness parameter $\mathcal{T}$ is defined by a tuple of four PPT algorithms $\Pi_{HTLP} = (\mathsf{HP.Setup}, \mathsf{HP.Gen}, \mathsf{HP.Solve}, \mathsf{HP.Eval})$ as follows:*

1. $\mathsf{HP.Setup}(1^\lambda, \mathcal{T}) \xrightarrow{\$} pp$: is a probabilistic algorithm that takes as input security parameter 1^λ and hardness parameter $\mathcal{T}$ and outputs public parameters pp.
2. $\mathsf{HP.Gen}(pp, s) \xrightarrow{\$} \mathcal{Z}$: a probabilistic algorithm that takes as input the public parameters pp and a secret $s \in \mathcal{S}_\lambda$ and outputs a puzzle $\mathcal{Z}$.
3. $\mathsf{HP.Solve}(pp, \mathcal{Z}) \rightarrow s$: is a deterministic algorithm that takes as input public parameters pp and puzzle $\mathcal{Z}$, and outputs a solution s.

4. $\mathsf{HP.Eval}(pp, C, \Psi, \mathcal{Z}_1, \ldots, \mathcal{Z}_n) \xrightarrow{\$} \tilde{\mathcal{Z}}$: is a probabilistic algorithm taking as input a circuit C, public parameters, the homomorphic-operation Ψ and a set of n puzzles $(\mathcal{Z}_1, \ldots, \mathcal{Z}_n)$, and outputs a master puzzle $\tilde{\mathcal{Z}}$.

Correctness The following are formal definitions of correctness, security and compactness of an HTLP scheme in line with [97]. The following definition considers the case when the evaluation algorithm is executed only once.

Definition 66 (HTLP Correctness)**.** *Let Π_{HTLP} from Definition 65 satisfy correctness for the class of circuits C where $C = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ if $\forall\lambda \in \mathbb{N}$, all polynomials $\mathcal{T}$ in λ, all circuits $C \in \mathcal{C}_\lambda$ and respective inputs $(s_1, \ldots, s_n) \in \mathcal{S}^n$, all public parameters pp and for all puzzles $\mathcal{Z}_i$ in support of $\mathsf{HP.Gen}(pp, s_i)$, the following two conditions are satisfied:*

1. There exists a negligible function negl such that $\Pr[\mathsf{HP.Solve}(pp, \mathsf{HP.Eval}(C, pp, \Psi, \mathcal{Z}_1, \ldots, \mathcal{Z}_n)) \neq C(s_1, \ldots, s_n)] \leq \mathsf{negl}(1^\lambda)$.
2. There exists a fixed polynomial $p(\cdot)$ such that $\mathsf{HP.Solve}(pp, \mathcal{Z})$ runtime is bounded by $p(1^\lambda, \mathcal{T})$ for $\mathcal{Z} \xleftarrow{\$} \mathsf{HP.Eval}(C, pp, \Psi, \mathcal{Z}_1, \ldots, \mathcal{Z}_n)$.

Definition 67 (HTLP Security)**.** *Π_{HTLP} is secure with gap $\epsilon < 1$ if there exists a polynomial $\tilde{\mathcal{T}}(\cdot)$ such that for all polynomials $\mathcal{T}(\cdot) \geq \tilde{\mathcal{T}}(\cdot)$ and every polynomial-size adversary $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda\in\mathbb{N}}$ where the depth of $\mathcal{A}_2$ is bounded from above by $\mathcal{T}^\epsilon(1^\lambda)$, there exists a negligible function negl, such that for all $\lambda \in \mathbb{N}$ it holds that*

$$Pr\left[b \leftarrow \mathcal{A}_2(pp, \mathcal{Z}, \tau) : \begin{array}{l}(\tau, s_0, s_1) \leftarrow \mathcal{A}_1(1^\lambda),\\ pp \xleftarrow{\$} \mathsf{HP.Setup}(1^\lambda, \mathcal{T}(1^\lambda)),\\ b \xleftarrow{\$} \{0,1\},\\ \mathcal{Z} \xleftarrow{\$} \mathsf{HP.Gen}(pp, s_b)\end{array}\right] \leq \tfrac{1}{2} + \mathsf{negl}(1^\lambda)$$

and $(s_0, s_1) \in \mathcal{S}^2$.

Definition 68 (HTLP Compactness)**.** *Π_{HTLP} is compact for the class of circuits $C = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ if $\forall\lambda \in \mathbb{N}$, all polynomials $\mathcal{T}$ in λ, all circuits $C \in \mathcal{C}_\lambda$ and respective inputs $(s_1, \ldots, s_n) \in \mathcal{S}^n$, all public parameters pp and for all puzzles $\mathcal{Z}_i$ in support of $\mathsf{HP.Gen}(pp, s_i)$, the following two conditions are satisfied:*

1. There exists a fixed polynomial $p(\cdot)$ such that $|\mathcal{Z}| = p(1^\lambda, |C(s_1, \ldots, s_n)|)$, where $\mathcal{Z} \xleftarrow{\$} \mathsf{HP.Eval}(C, pp, \mathcal{Z}_1, \ldots \mathcal{Z}_n)$.

2. There exists a fixed polynomial $\tilde{p}(\cdot)$ such that the runtime $\mathsf{HP.Eval}(C, pp, \Psi, \mathcal{Z}_1, \ldots, \mathcal{Z}_n)$ is bounded by $\tilde{p}(1^\lambda, |C|)$.

6.3.2 Building the FRSS Scheme

Equipped with our chosen building blocks of an HTLP and threshold RSS scheme (given in Section 6.2, Definition 62), we now formally define our generic construction of an RSS scheme, entitled FRSS.

Definition 69 (FRSS Construction)**.** *Given security parameter λ, time hardness parameter $\mathcal{T}$, an efficiently samplable distribution of the set of secrets $\mathcal{S}_\lambda$ with operator Ψ, secret $s \in \mathcal{S}_\lambda$, efficiently samplable discrete distributions $\mathcal{G}, \mathcal{G}'$, we construct an FRSS scheme with reconstruction phase in the non-simultaneous setting as a tuple of three PPT algorithms $\Pi_{FRSS} = (\mathsf{Setup}', \mathsf{Share}', \mathsf{Recon}')$ from a secret sharing scheme $\Pi_{RSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Recon})$ and a HTLP scheme $\Pi_{HTLP} = (\mathsf{HP.Setup}, \mathsf{HP.Gen}, \mathsf{HP.Solve}, \mathsf{HP.Eval})$ as follows:*

- **Sharing Phase:** The honest dealer D takes as input the secret $s \in \mathcal{S}_\lambda$ and performs the following steps non-interactively:

  1. $\mathsf{Setup}'(1^\lambda, \mathcal{T}) \xrightarrow{\$} pp'$ a probabilistic algorithm on inputs $1^\lambda, \mathcal{T}$ in which the dealer runs:

     (a) $\mathsf{HP.Setup}(1^\lambda, \mathcal{T}) \xrightarrow{\$} pp_1$ which outputs public parameters pp_1.

     (b) $\mathsf{Setup}(1^\lambda, \mathcal{T}) \xrightarrow{\$} pp_2$ which outputs the public parameters pp_2. Additionally, let $r \xleftarrow{\$} \mathcal{G}$ be the sampled revelation value and d be a random value $d \xleftarrow{\$} \mathcal{G}'$.

     Outputs are sampled values r, d and public parameters $\{pp_1, pp_2\}$. Therefore, $pp' := \{r, d, pp_1, pp_2\}$.

  2. $\mathsf{Share}'(pp', s) \xrightarrow{\$} \{s_0, \{list_1, \ldots, list_n\}\}$: a probabilistic algorithm that takes as input the secret $s \in \mathcal{S}_\lambda$ and public parameters pp'. The output consists of a checking share s_0 and lists labelled $list_j$ for $j \in [n]$, each composed of m sub-puzzles for $m = r + d$.

(a) Run $\mathsf{Share}(pp_2, s) \xrightarrow{\$} \{s_0, \{s_1, \ldots, s_r\}\}$ a probabilistic algorithm with inputs the public parameters pp_2 and secret $s \in \mathcal{S}_\lambda$. The outputs are $(r+1)$ shares of the secret; the checking share s_0 and s_i for $i \in [r]$.

(b) $\mathcal{S}_\lambda \xrightarrow{\$} \{s_{r+1}, \ldots, s_m\}$, randomly sample d fake shares from $\mathcal{S}_\lambda$.

(c) For every $i \in [m]$, compute the list of sub-shares $\{s_{i,1}, \ldots, s_{i,n}\}$ such that $s_i = \underset{j \in [n]}{\Psi} s_{i,j}$.

(d) Run $\mathsf{HP.Gen}(pp_1, s_{i,j}) \xrightarrow{\$} \mathcal{Z}_{i,j}$ a probabilistic algorithm that takes as input sub-shares $s_{i,j}$ and public parameters pp_1, and outputs sub-puzzles $\mathcal{Z}_{i,j}$, $\forall i \in [m], \forall j \in [n]$.

(e) D distributes $\text{list}_j = \{\mathcal{Z}_{1,j}, \cdots, \mathcal{Z}_{r,j}, \mathcal{Z}_{r+1,j}, \cdots, \mathcal{Z}_{m,j}\}$ to the corresponding player P_j, for every $j \in [n]$.

3. The dealer distributes the following:

(a) D broadcasts $\{pp', s_0\}$ to all P the public parameters pp' and checking share s_0.

(b) D distributes list_j to P_j for every $j \in [n]$.

- **Reconstruction Phase:** All players in $P = \{P_1, \ldots, P_n\}$ independently take part in this phase.

1. Communication: We are in the kth round of communication, for some $1 < k \leq m$.

(a) P_j sends to all of P the sub-puzzle $\mathcal{Z}_{k,j}$ for every $j \in [n]$ non-simultaneously.

(b) At the end of round k (after time $\mathcal{T}$ has elapsed), along with their sub-puzzle, player P_j should have received $\{\mathcal{Z}_{k,1}, \ldots, \mathcal{Z}_{k,n}\}$ from all of P.

(c) Move to round $(k+1)$ of communication and round k of processing, unless fewer than $(n-1)$ sub-puzzles have been received. In this case, proceed to abort communication and move to the penultimate step of the processing phase with previously reconstructed shares $\{s_1, \ldots, s_{k-1}\}$.

2. Processing: We are in the round $(k-1)$ of processing, for some $1 < k \leq m$.[2] For any $j \in [n]$, P_j does the following:

[2]At least one round of communication is required before players can start processing.

(a) $\mathsf{HP.Eval}(pp_1, \mathcal{T}, \Psi, \mathcal{Z}_{k-1,1}, \cdots, \mathcal{Z}_{k-1,n}) \xrightarrow{\$} \mathcal{Z}_{k-1}$: Run the probabilistic algorithm $\mathsf{HP.Eval}$ with inputs the public parameters pp_1, hardness parameter $\mathcal{T}$, and the list of n sub-puzzles for the $(k-1)$th round, a player homomorphically evaluates sub-puzzles with operator Ψ to output share puzzle $\mathcal{Z}_{k-1}$.

(b) $\mathsf{HP.Solve}(pp_1, \mathcal{T}, \mathcal{Z}_{k-1}) \rightarrow s_{k-1}$: Run the deterministic algorithm $\mathsf{HP.Solve}$ that takes as input the public parameters pp_1; hardness parameter $\mathcal{T}$; and puzzle share $\mathcal{Z}_{k-1}$ and outputs secret share s_{k-1}. Output the round share s_{k-1} and move to reconstruct s.

(c) $\mathsf{Recon}'(pp', s_0, \{s_1, \ldots, s_{k-1}\}) \rightarrow \{s, \perp\}$: where the players run $\mathsf{Recon}(pp_2, \{s_1, \ldots, s_{k-1}\}) \rightarrow \{s, \perp\}$, a deterministic algorithm that inputs public parameters pp_2 and $(k-1)$ reconstructed shares of the secret $\{s_1, \ldots, s_{k-1}\}$. Player P_j uses checking share s_0 to confirm the soundness of their reconstructed value and outputs either the correct secret s or abort $\perp$. If this is the final round of processing, stop here. Else,

(d) If every player communicated in round k but P_j output $\perp$ in the previous step *and* every player $P_j \in P$ has $list_j \neq \emptyset$, players go to $(k+1)$th round of the reconstruction phase. Otherwise, the protocol aborts and outputs $\perp$.

Correctness and Secrecy Now, we formally define *correctness* and *secrecy* for an FRSS scheme (Π_{FRSS}), recalling the SS definitions presented in Section 2.4 (Definitions 20 and 21 respectively). Intuitively, the correctness of Π_{FRSS} means that players reconstructing a secret from a threshold or more ($(k-1) \geq r$ for threshold r) of honestly generated shares will reconstruct the correct secret with overwhelming probability.

Definition 70 (FRSS Correctness)**.** *Rational secret sharing scheme* Π_{FRSS} *is* correct *if* $\forall \lambda \in \mathbb{N}$ *and for all possible sets of* n *authorised players* $P = \{P_1, \ldots, P_n\}$*: given any secret* $s \in \mathcal{S}_\lambda$*, any* $r \xleftarrow{\$} \mathcal{G}$ *treated as the threshold,* $\mathsf{Setup}'(1^\lambda, \mathcal{T}) \xrightarrow{\$} pp'$ *and shares* $\{s_0, \{s_1, \ldots, s_m\}\}$ *generated in the process of running* Share'*, if round* $1 \leq r \leq (k-1)$ *is the final round in which* n *sub-puzzles have been shared, resulting in the set* $S' = \{s_1, \ldots, s_{k-1}\}$ *of shares evaluated during the processing phase, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that,*

$$Pr[\mathsf{Recon}'(pp', s_0, S') \neq s] \leq \mathsf{negl}(1^\lambda).$$

Informally, an FRSS scheme (Π_{FRSS}) satisfies the secrecy property if players reconstructing a secret from less than a threshold ($(k-1) < r$ for threshold r) of honestly generated shares will reconstruct a value not equal to $\perp$ with negligible probability. More formally,

Definition 71 (FRSS Secrecy)**.** *Rational secret sharing scheme Π_{FRSS} from Definition 62 satisfies* secrecy *if $\forall\lambda \in \mathbb{N}$ and for all possible sets of n authorised players $P = \{P_1, \ldots, P_n\}$: given any secret $s \in \mathcal{S}_\lambda$, any $r \xleftarrow{\$} \mathcal{G}$ treated as the threshold, $\mathsf{Setup}'(1^\lambda, \mathcal{T}) \xrightarrow{\$} pp'$ and shares $\{s_0, \{s_1, \ldots, s_m\}\}$ generated in the process of running Share', if round $1 \leq (k-1) < r$ is the last round in which n sub-puzzles have been shared, resulting in the set $S' = \{s_1, \ldots, s_{k-1}\}$ of shares evaluated during the processing phase, there exists a negligible function $\mathsf{negl}(\cdot)$ such that,*

$$Pr[\mathsf{Recon}'(pp', s_0, S') \neq \perp] \leq \mathsf{negl}(1^\lambda).$$

6.4 Security Analysis

In this Section, we analyse the security of our generic construction Π_{FRSS} from Definition 69. Recall from Section 6.1, RSS typically adopts game theory to assess the behaviour of participating rational players. Before our formal statement of security, we present a preliminary subsection detailing game theoretical notions relevant to RSS schemes, followed by definitions of necessary assumptions used to prove our statement of security.

6.4.1 Game Theory for Rational Secret Sharing

The key question in this Chapter is how we can ensure that players (despite any preferences they may have) are motivated to follow a strategy in the non-simultaneous setting. This is typically done by assuming that the *strategies* of players are in a state of *equilibrium*. This assumption is necessary to make certain that if every player $P_i \in P$ believes all other players in P are following their prescribed strategy in the phase, then they have nothing to gain in deviating from their strategy and are penalised in some way by deviating.

To demonstrate the properties of *fairness* and *soundness*, we view our FRSS construction, formalised in Definition 69, through a *game-theoretic lens*. Step-by-step

we will define the concept of utility functions, Nash equilibrium, and soundness with checking shares. Precisely, to prove our statement of security (Theorem 8) our generic construction Π_{FRSS} needs to ensure players' strategies are in a *computationally strict Nash equilibrium* when they additionally have access to *side information* related to the secret. Moreover, we must prove that the checking share used in our construction is sound protocol-induced side information. Observe, the notation and definitions presented in this subsection follow the framework of [5, 40, 96, 56].

Utility Functions To describe the possible outcomes of an RSS game, the preferences of players are represented by utility functions which are then used to define a Nash Equilibrium [103], the latter of which is necessary for our security statement to hold.

Definition 72 (Utility Functions)**.** *The set U_i for each player P_i is defined as the set of utility* values *resulting from the possible outcomes of the game, which are polynomial in the security parameter (λ) of the protocol. These values are determined by the outcomes $\overrightarrow{o}$ of the reconstruction phase, which depends on the strategies $\overrightarrow{\sigma}$ taken by the n rational parties. The set consists of the following, $U_i = \{U_i^{TN}, U_i^{TT}, U_i^{NN}, U_i^{NT}, U_i^{FN}, U_i^{NF}\}$. The parameters T, N, and F define the utility gained from the following actions of players (in which the players may be honest or dishonest):*

- *T signifies the case where a player learns the secret.*
- *N signifies the case where a player does not learn the secret.*
- *F signifies the case where a player learns a fake secret (mislead).*

For example, the utility U_i^{NF} is the value P_i gains from not learning the secret, whilst misleading the other $(n-1)$ parties into learning a fake (incorrect) secret. There are two preference relation scenarios which are of importance in defining our games for rational players, $\forall i \in [n]$:

1. $U_i^{TN} > U_i^{TT} > U_i^{NN} > U_i^{FN}$ *and* $U_i^{NF} \geq U_i^{TT}$;
2. $U_i^{TN} > U_i^{TT} > U_i^{NN} > U_i^{FN}$ *and* $U_i^{NF} < U_i^{TT}$.

Recall, the property of *fairness* is important to achieve (Definition 63) in RSS schemes such as our construction (Π_{FRSS}). One way to satisfy fairness in the game-theory setting is for the utility values gained from the outcome of the game to be independent of the game itself. More formally,

Definition 73 (Utility Independence)**.** *Let $\tilde{U} \in U$ be a set for a specific utility function, consisting of all the corresponding utility values for every player in P. Define the set of polynomial utility functions $U' = \{U_i^{TN}, U_i^{TT}, U_i^{NN}, U_i^{NT}, U_i^{FN}, U_i^{NF}\}_{i=1}^{n} \setminus \tilde{U}_{i=1}^{n}$, as the set excluding all $\tilde{U}_{i=1}^{n}$ values. A mechanism $(\Gamma, \overrightarrow{\sigma})$ is said to be $\tilde{U}$-*utility independent *if for all polynomial utility functions $\tilde{U}_{i=1}^{n}$, the elements in $U = U' \cup \tilde{U}_{i=1}^{n}$ satisfy a certain preference relationship $\mathcal{R}$. Therefore, $(\Gamma, \overrightarrow{\sigma})$ is a fair reconstruction mechanism for preference relationship $\mathcal{R}$ among the elements of U.*

Nash Equilibria An important consequence of an RSS attaining *utility independence* is that the scheme reaches a state of *Nash Equilibrium* [103]. Informally, a Nash equilibrium is a formalisation of what it means for players in a secret sharing scheme to follow their strategies. Let $\Gamma_{r,r+1} = (\Gamma, \overrightarrow{\sigma})$ be the scheme that players in P are following, where $\sigma_i \in \overrightarrow{\sigma}$ is a strategy for player P_i, for some $i \in [n]$. Thus, to demonstrate the provable security of Π_{FRSS}, we must show that players' strategies $\overrightarrow{\sigma}$ are in a *computationally strict Nash equilibrium* [40].

Definition 74 (Computationally Strict Nash Equilibrium)**.** *Given reconstruction phase $\Gamma_{r,r+1} = (\Gamma, \overrightarrow{\sigma})$, a strategy profile $\overrightarrow{\sigma}$ for Γ is said to be in a computationally strict Nash Equilibrium if for every $i \in [n]$ and every deviating strategy $\sigma_i' \neq \sigma_i$ that player P_i uses, it holds that $U_i(\sigma_i, \overrightarrow{\sigma_{-i}}) > U_i(\sigma_i', \overrightarrow{\sigma_{-i}})$.*

To rephrase, the utility (gain) of P_i following an alternative strategy σ_i' is less than the utility of P_i following their prescribed strategy σ_i from $\Gamma_{r,r+1}$, assuming all other players are following their prescribed strategy $\overrightarrow{\sigma_{-i}}$. Therefore, a rational player will adopt the strategy that results in them obtaining the highest utility value in the reconstruction phase. Definition 74 is used to ensure that players choose to follow their prescribed strategy, thus, ensuring that the property of *fairness* is achieved. We formalise a fair reconstruction in the context of utility functions as follows,

Definition 75 (Fair reconstruction Mechanism)**.** *Let $U = \{U_1, \ldots, U_n\}$ be the set of utility functions for players in P. The reconstruction phase $\Gamma_{r,r+1} = (\Gamma, \overrightarrow{\sigma})$ is fair for utility functions U if $\overrightarrow{\sigma}$ is a computational Nash equilibrium, and the probability that the outcome $\overrightarrow{\omega} = \perp$ for players in P when they follow $\overrightarrow{\sigma}$ is negligible.*

Soundness with Checking Shares

The notion of equivalent play (Nash Equilibrium) incorporates a scheme where players have access to any *side information* related to the secret, like the checking share that we defined in our generic FRSS construction from Definition 6.3.

Let Aux $= \{aux_i\}_{i\in[n]}$ define the (protocol-induced) side information related to the secret that players in P have access to during $\Gamma_{r,r+1}$. When deviant player P_i takes alternative strategy σ_i', then $\sigma_i' \overset{\text{Aux}}{\approx} \overrightarrow{\sigma}$ yields equivalent play if given the views of all other players $P_j \in P_{-i}$, including their side information aux_j for player P_j, no polynomial time algorithm can distinguish whether P_i is following prescribed strategy $\sigma_i \in \overrightarrow{\sigma}$ or alternative strategy σ_i'. More formally,

Definition 76 (Computationally Strict Nash Equilibrium with Protocol-Induced Side Information)**.** *Prescribed strategy $\overrightarrow{\sigma}$ in reconstruction phase $\Gamma_{r,r+1} = (\Gamma, \overrightarrow{\sigma})$ is a computational Nash equilibrium in the presence of protocol-induced side-information $Aux = \{aux_i\}_{i\in[n]}$ if it is a Nash equilibrium with protocol-induced side information such that, given security parameter λ, for every $P_i \in P$ and any PPT alternative strategy $\sigma_i' \overset{Aux}{\not\approx} \overrightarrow{\sigma}$, there exists a negligible function negl over the security parameter λ such that,*

$$U_i((\sigma_i', \sigma_{-i}), Aux) < U_i(\overrightarrow{\sigma}, Aux) + \mathsf{negl}(1^\lambda).$$

The authors of [96] propose giving each player in P some *arbitrary* auxiliary information or access to a membership oracle $O_{k,j}^{s_0}$. Either of these enables a player $P_j \in P$ to confirm whether the value $s' \in \mathcal{S}_\lambda$ they reconstructed in the kth round of reconstruction is the correct secret or not.

The work of [40] follows from [96] by introducing protocol-induced (i.e. chosen by the protocol designer) auxiliary information in the form of an extra share of the secret, known as the *checking share*. In line with the authors of [40], we first define what a membership oracle is, to use *specific* protocol-induced side information to ensure soundness.

Intuitively, a membership oracle should not reveal any information about the secret itself, meaning that no player can learn anything important about s by simply observing the oracle or querying it with arbitrary inputs. We present the following definitions from [40].

Definition 77 (Membership Oracle)**.** *Given the secret s and reconstructed value s', both in secret space $\mathcal{S}_\lambda$ and checking share s_0, we define a membership oracle $O^{s_0}_{k,j} : \mathcal{S}_\lambda \to \{0,1\}$ queried by player $P_j \in P$ in the kth round of the reconstruction phase as follows:*

$$O^{s_0}_{k,j}(s') = \begin{cases} 1 \text{ if } s' = s, \\ 0 \text{ otherwise.} \end{cases}$$

Note that the behaviour of the oracle may be dependent on the secret s. For example, it may output 1 on s and 0 on all other inputs, or it may output 1 on input s' if $f(s) = f(s')$ for some function f, where the function depends on the reconstruction process of the underlying SS scheme. Given this, in order to ensure the soundness of the reconstruction phase with reconstructed value $s' \in \mathcal{S}_\lambda$, the oracle must always output the correct decision on whether this is the secret or not. Similarly to [40], we want to ensure soundness, therefore we must define a sound membership oracle.

Definition 78 (Sound Membership Oracle)**.** *Given input $s' \in \mathcal{S}_\lambda$, a correct membership oracle $O^{s_0}_{k,j} : \mathcal{S}_\lambda \to \{0,1\}$ for player $P_j \in P$ in the kth round of reconstruction with access to checking share s_0 has the following properties:*

1. $Pr[O^{s_0}_{k,j}(s') = 1] \leq \mathsf{negl}(1^\lambda)$ *for any* $s' \neq s$;
2. $Pr[O^{s_0}_{k,j}(s') = 0] \leq \mathsf{negl}(1^\lambda)$ *for* $s' = s$.

for a negligible function $\mathsf{negl}(1^\lambda)$, over security parameter λ.

The next definition follows directly from the definition of a sound membership oracle.

Definition 79 (Protocol-Induced Membership Oracle)**.** *A sound membership oracle $O^{s_0}_{k,j}$ provided by the scheme $\Pi_{FRSS} = (\mathsf{Setup}', \mathsf{Share}', \mathsf{Recon}')$, given to player P_j for $j \in [n]$ for the kth round of the reconstruction phase is called a protocol-induced membership oracle.*

For our construction Π_{FRSS}, the sound membership oracle can be modelled as protocol-induced auxiliary information in the form of a checking share. Assuming players do not have a preference to mislead, let $(\Gamma_f, \overrightarrow{\sigma_f})$ be a fair reconstruction phase (Definition 63, Section 6.2) where players follow their prescribed strategies of communicating and reconstructing some function f to reconstruct the secret s. Note that function f is the honestly constructed function created by the dealer during the sharing phase,

dependent on the underlying SS scheme and its assumed security, in which players participating in the fair reconstruction phase $(\Gamma_f, \overrightarrow{\sigma_f})$ with a sufficient number of shares can reconstruct to obtain the secret.

In our construction, similar to the work of [40], we propose a fair reconstruction phase $(\Gamma_{f_k}, \overrightarrow{\sigma_{f_k}})$ for players $P_j \in P$ with k shares and access to protocol-induced auxiliary information in the form of a checking share s_0, assuming that some players may prefer to mislead other players into outputting an incorrect secret. Informally, this works as follows:

Let player P_i deviate with strategy σ_i' in the $(k+1)$th round and the non-deviant players P_{-i} are following strategies $\overrightarrow{\sigma_{f_k}}$ throughout the reconstruction phase. Strategy $\sigma_{f_k,j}$ for P_j instructs a player to follow their normal strategy $\sigma_{f,j}$ of reconstructing a value $s' \in \mathcal{S}_\lambda$ or outputting $\perp$ in the case that P_i quits communicating in the $(k+1)$th round. If a value $s' \in \mathcal{S}_\lambda$ is reconstructed from reconstructed function f_k, strategy $\sigma_{f_k,j}$ instructs P_j to check the soundness of value s' obtained in the kth round, by using the checking share s_0. Let $s_0 = (y_0, f(y_0))$ for some input y_0 computed by the dealer in the sharing phase; if $f_k(y_0) = f(y_0)$, then P_j concludes that $s' = s$, otherwise P_j concludes that $s' \neq s$.

6.4.2 Proof of Security

Theorem 8. *The generic construction of a non-simultaneous rational secret scheme (Definition 69)* $\Pi_{\mathsf{FRSS}} = (\mathsf{Setup}', \mathsf{Share}', \mathsf{Recon}')$ *satisfies correctness, secrecy, fairness and soundness in the presence of side information related to the secret, assuming the following properties:*

- *correctness, security, and compactness of the HTLP scheme,*
- *correctness and secrecy of the SS scheme,*
- *the checking share side information is correct, protocol-induced auxiliary information.*

Proof Overview Informally, we prove Theorem 8 by demonstrating that our construction satisfies correctness, and secrecy, achieves soundness in the non-simultaneous setting using protocol-induced side information and achieves fairness despite the presence of this side information by using an HTLP to provide a time-delay to the scheme.

Specifically, in our security analysis, we summarise the scenarios in which a deviant player attempts to mislead. In particular, we demonstrate that if a player aborts in a round k with respect to revelation round r, regardless of the round that k is, the outcome for all players is the same. Analysing the scenarios in which a player quits communicating aids the proofs of fairness and soundness, by providing an intuition to the outcome of the reconstruction phase.

Fairness of the scheme is proven as follows: we show that Definition 63, Section 6.2.1 is satisfied in our construction assuming the correctness and security of the HTLP scheme [97] (definitions of which are provided in Section 6.3.1), which is employed to implement a time-delay in the scheme. We use a reduction to break the correctness and security of the HTLP scheme, contradicting our assumptions, to show that there does not exist a deviant player with the ability to decrypt a puzzle in a time less than $\mathcal{T}$.

Furthermore, assuming the correctness and secrecy of the underlying SS scheme (Section 2.4), we show that the probability of a deviant player learning the secret, whilst other players do not, is negligible in the security parameter λ. We additionally show that the rational player's strategies $\overrightarrow{\sigma}$ are in a computationally strict Nash equilibrium (Section 6.4.1, Definition 76) following the proofs of [40, 96].

To prove soundness, we provide Section 6.4.1 preceding the analysis of Theorem 8, to define the side information used to achieve soundness. We closely follow the proof of [40] to define a membership oracle. Recall, this is an oracle queried by players in reconstruction to check the soundness of their reconstructed value [96]. Following [40], we claim and prove that the checking share in our construction can be used in place of a sound membership oracle (see Definition 78, Section 6.4.1), as a specific form of protocol-induced side information to ensure soundness. Finally, we prove that our generic construction achieves soundness with a checking share.

Proof. First, we formally prove that the correctness property is satisfied.

Claim 2. *The generic* Π_{FRSS} *scheme presented in Definition 69 satisfies correctness (Definition 70) assuming the underlying RSS and HTLP schemes (*Π_{RSS} *and* Π_{HTLP} *respectively) satisfy their respective correctness definitions.*

Proof. Correctness of Π_{FRSS} translates to demonstrating that there is a negligible probability of reconstructing a secret from honestly generated shares, which differs

from the correct secret. By design of our scheme, we, therefore, need to analyse the reconstruction phase, given that $(k-1)$ shares were honestly generated in the sharing phase for some $1 \leq r \leq k-1$ in which r is the threshold value. In more detail, We begin by proving the correctness of our construction Π_{FRSS}, which follows from the correctness of the SS and HTLP schemes underlying it. For all possible players in P, $\forall \lambda, \mathcal{T} \in \mathbb{N}$ and for all possible secrets $s \in \mathcal{S}_\lambda$; given $\mathsf{Setup}'(1^\lambda, \mathcal{T}) \rightarrow pp'$ and $\mathsf{Share}'(pp', s) \rightarrow \{s_0, \{list_1, \ldots, list_n\}\}$ run as in our construction (Section 6.5) using homomorphic operator Ψ, our scheme $\Pi_{\mathsf{FRSS}} = (\mathsf{Setup}', \mathsf{Share}', \mathsf{Recon}')$ is correct for some negligible function $\mathsf{negl}(\cdot)$ over the security parameter λ. Namely, at round r of the reconstruction phase, players reconstruct s using the first r reconstructed shares. Correctness holds except with negligible probability for the following reasons:

- The reconstruction phase starts with players evaluating sub-puzzles and then solving the output. Following the correctness of $\Pi_{\mathsf{HTLP}} = (\mathsf{HP.Setup}, \mathsf{HP.Gen}, \mathsf{HP.Eval}, \mathsf{HP.Solve})$, the HTLP scheme (recalled in Section 6.3.1, Definition 66) constructed using operator Ψ such that for $\forall i \in [r]$, and shares $s_i = \underset{j \in [n]}{\Psi} s_{i,j}$, will satisfy the ensuing equation for some negligible function $\mathsf{negl}'(\cdot)$,

$$\Pr[\mathsf{HP.Solve}(pp, \mathsf{HP.Eval}(pp_1, \mathcal{T}, \Psi, \mathcal{Z}_{i,1}, \ldots, \mathcal{Z}_{i,n})) \neq s_i] \leq \mathsf{negl}'(1^\lambda).$$

- Assuming shares $\{s_1, \ldots, s_{k-1}\}$ were constructed correctly as we just described, and following the assumed correctness of the underlying RSS scheme we have, for some negligible function $\mathsf{negl}''(\cdot)$,

$$\Pr[\mathsf{Recon}(pp_2, \{s_1, \ldots, s_{k-1}\}) \neq s] \leq \mathsf{negl}''(1^\lambda).$$

Therefore, the resulting equation follows assuming the existence of some negligible function $\mathsf{negl} = \mathsf{negl}' + \mathsf{negl}''$ over security parameter λ.

$$\Pr[\mathsf{Recon}'(pp', s_0, \{s_1, \ldots, s_{k-1}\}) \neq s] \leq \mathsf{negl}(1^\lambda).$$

Consequently, Definition 70 of correctness is satisfied. □

Second, we formally prove that the secrecy property is satisfied.

Claim 3. *The generic construction Π_{FRSS} presented in Definition 69 satisfies secrecy (Definition 71) assuming the underlying SS scheme (Π_{RSS}) satisfies a definition of secrecy.*

Proof. Secrecy of Π_{FRSS} translates to demonstrating that there is a negligible probability of reconstructing anything other than the failure symbol ($\perp$) given less than a threshold (r) of shares. Namely, reconstructing a meaningful value from set $S' = \{s_1, \ldots, s_{k-1}\}$ such that $1 \leq (k-1) < r$. By design of our scheme, we need to analyse the reconstruction phase (in which Recon' is run), given that shares were honestly generated in the sharing phase (Share').

In more detail, given the correct execution of n sub-puzzle evaluation which outputs a share for each round up to the final round $(k-1) < r$, the secrecy property of the underlying threshold RSS scheme holds since we have less than a threshold number of shares. That is, following Definition 71, Section 6.3 we have, for some negligible function $\mathsf{negl}(\cdot)$,

$$\Pr[\mathsf{Recon}(pp_2, \{s_1, \ldots, s_{k-1}\}) \neq \perp] \leq \mathsf{negl}(1^\lambda).$$

Therefore, the proceeding equation follows assuming the existence of some negligible function $\mathsf{negl}(\cdot)$ over security parameter λ.

$$\Pr[\mathsf{Recon}'(pp', s_0, S') \neq \perp] \leq \mathsf{negl}(1^\lambda).$$

Consequently, Definition 71 of secrecy is satisfied. □

Next, we analyse the fairness and soundness of our construction. We note that the outcome of the reconstruction phase $\Gamma_{r,r+1}$ is the same for every player in P irrespective of the round in which a deviant player quits in relation to the revelation round. To see this we explain what happens when deviant player P_i quits and the resulting outcome.

We will only look at a player who is the last to communicate in a round. If a player quits before every other player has sent their sub-share for that round, then they will not be able to reconstruct the corresponding share for the round that they quit in. In this case, the deviant player is in no better a position than other players.[3]

Suppose P_i follows the reconstruction phase with their prescribed strategy σ_i (that is, to share the corresponding sub-puzzle) for the first k rounds, then follows the alternative strategy σ_i' (quit communicating) in the $(k+1)$th round.

[3]Notice that the non-deviant players will also be unable to reconstruct the share for that round.

The other players discounting P_i are labelled P_{-i} and are assumed to be following the same strategy, prescribed by the dealer, of communicating. This is represented by the $(n-1)$ vector of strategies σ_{-i}. To restore fairness, P_{-i} must abort communication after the $(k+1)$th round has finished (bounded above by time $\mathcal{T}$) when they realise that P_i has not sent their sub-puzzle $\mathcal{Z}_{k+1,i}$, concluding that the previous round was the revelation round r. P_{-i} are not able to homomorphically evaluate $(k+1)$ sub-puzzles, since P_i did not communicate $\mathcal{Z}_{k+1,i}$, so they cannot reconstruct the $(k+1)$th share.

As a consequence, P_{-i} move to reconstruct the secret from the k previously reconstructed shares $\{s_1, \ldots, s_k\}$ by running Recon'. The checking shares s_0, essential to ensuring soundness, is used before P_{-i} outputs a result to confirm whether their outcome from reconstruction is the true secret.

As we have mentioned, the outcomes $\overrightarrow{\omega} = (\omega_1, \ldots, \omega_n)$ for every player depending on the round that P_i quits relative to revelation value r. We denote the outcome of deviant player P_i as ω_i and the outcome for the other $(n-1)$ non-deviant players P_{-i} as ω_{-i} in the following cases:

Case (k+1)<r : As P_i is last to communicate, they will be able to reconstruct the $(k+1)$th puzzle $\mathcal{Z}_{k+1}$ of share s_{k+1} using the sub-puzzles communicated by all other players in the $(k+1)$th round. Essential to ensuring fairness, intuitively, the time delay is ensured by the correctness of the HTLP scheme (see Definition 66, Section 6.3.1).

Using the HTLP time-delay scheme in our construction prevents the deviant player P_i from solving $\mathcal{Z}_{k+1}$ (by running $\mathsf{HP.Solve}$) before time $\mathcal{T}$ has elapsed and other players begin the reconstruction process. Thus, P_i has no time to reconstruct the secret before players P_{-i} begin reconstructing the secret. Indeed, by the property of compactness in the HTLP scheme, the players P_{-i} can evaluate the sub-puzzles to obtain the corresponding share before the end of the reconstruction phase round.

Furthermore, the total number of shares that P_i possesses remains insufficient $(< r)$ to reconstruct the secret. As a consequence, P_i will be identified as a cheater when P_{-i} use the checking share to verify their reconstruction solution from shares $\{s_1, \ldots, s_k\}$, as they too have an insufficient number of shares to reconstruct s. The output of the phase will be $\perp$. **Outcome for all players:** $\overrightarrow{\omega} = \perp$.

Note that P_i could attempt to reconstruct the secret with previously reconstructed shares plus the checking share s_0. Whilst this means that P_i potentially has r shares

(as is the case if they quit when $k+1=r-1$) and can reconstruct the correct secret, the value of r remains unknown and the deviant player no longer has the checking share to verify that they have reconstructed the correct secret. Even if P_i adopts this strategy, the best P_i can do is correctly guess the value of r. Additionally, players P_{-i} will reconstruct an incorrect value and identify P_i as a cheater.

Case (k+1)=r : Whilst P_i has the correct number of shares to reconstruct s (they do not know this at the time), by the design of our construction there is an upper bound of time $\mathcal{T}$ for the length of a round in the phase, as in the first case, so P_i cannot solve $\mathcal{Z}_{k+1}$ within this time limit of round $(k+1)$ to reconstruct the share s_{k+1}. Furthermore, other players will have started reconstruction of the secret as follows.

When the $(k+1)$th round of communication has finished, P_{-i} will abort after not receiving sub-puzzle $\mathcal{Z}_{k+1,i}$ from P_i, moving to output the result from the reconstruction of the secret with the previous $k=(r-1)$ derived shares. As the other players have an insufficient number of shares to reconstruct s, the checking share will demonstrate that their output is incorrect, so $\omega_{-i}=\perp$, identifying P_i as a cheater in the process. P_i will not be able to reconstruct the secret s before players P_{-i} have output $\perp$. **Outcome for all players:** $\overrightarrow{\omega}=\perp$.

Case (k+1)>r : The non-deviant players P_{-i} will have previously reconstructed k shares from the phase. If $k=r$, P_{-i} has the precise number of shares to reconstruct s and can verify their output using checking share s_0, that is $\omega_{-i}=s$. Despite P_i quitting in the $(k+1)$th round, all players will be able to reconstruct the secret s when $k=r$. Our construction uses an $(r, r+1)$ secret sharing scheme such that one of the $(r+1)$ shares is a checking share that cannot be used to reconstruct the secret if soundness must be ensured. P_i can only deviate from their strategy up until the $(r+1)$th round of communication. **Outcome for all players:** $\overrightarrow{\omega}=s$.

Remark 9. Even if deviant players decide to send randomly generated sub-puzzles in the communication phase, the soundness of our scheme ensures that their deviance would be detected by honest players in the processing phase of reconstruction. This is because honest players possess the checking share, which will confirm that they have reconstructed an incorrect secret. As a result, the output of reconstruction from such an attack would be $\perp$.

Remark 10. For simplicity, we consider the case of one deviating player, however, we note that the construction tolerates up to $(r-1)$ deviant players cooperating.

Now that we have explained the various outcomes of the reconstruction phase, we show that fairness and soundness of our construction are ensured.

Fairness Following Definition 63 in Section 6.3, we want to show that there exists some negligible function negl under security parameter λ such that

$$\Pr[\omega_i(\Gamma, (\sigma_i', \sigma_{-i})) = s] \leq \Pr[\omega_{-i}(\Gamma, (\sigma_i', \sigma_{-i})) = s] + \mathsf{negl}(1^\lambda). \tag{6.1}$$

Let us note that there are two, mutually exclusive scenarios in which P_i learns s when deviating:

- When P_i takes strategy σ_i' in round $(k+1) = (r+1)$, as in the third case. Define this scenario as $\texttt{Event}_1$, which has probability $\Pr[\omega_i(\Gamma, (\sigma_i', \sigma_{-i})) = s] = \Pr[\omega_{-i}(\Gamma, (\sigma_i', \sigma_{-i})) = s]$, *or*
- When P_i takes strategy σ_i' in the round $(k+1) = r$, and P_{-i} has an insufficient number of shares to reconstruct s, by the secrecy of the underlying SS scheme. Define $\texttt{Event}_2$ to be the scenario in which P_i learns s but P_{-i} does not learn s.

  Firstly, the only way for P_i to learn s in this event is to evaluate the rth round sub-puzzles to obtain puzzle share $\mathcal{Z}_r$, solve $\mathcal{Z}_r$ to obtain share s_r and then reconstruct s from the r shares. The correctness of the HTLP states that the runtime for solving the puzzle is bounded by a fixed, positive polynomial $p(1^\lambda, \mathcal{T})$ and the security of the HTLP means that the solution of the puzzles is hidden for all players that run in (parallel) time $\mathcal{T}^\epsilon(\cdot) < \mathcal{T}(\cdot)$, for some $\epsilon < 1$. The definition of security follows the standard cryptographic security notion of indistinguishable CPA security. Suppose P_i can do this, meaning that there exists some algorithm able to solve a puzzle in time $\mathcal{T}^\epsilon(1^\lambda)$. We can use this in a reduction to break the correctness and security of the HTLP scheme, contradicting these assumptions in our construction.

  As a consequence, the probability that P_i solves $\mathcal{Z}_r$ to reconstruct the share s_r in time $\mathcal{T}^\epsilon$ is $\Pr[\mathsf{HP.PSolve}(pp_1, \mathcal{T}^\epsilon, \mathcal{Z}_r) \to s_r] \leq 1/2 + \mathsf{negl}(1^\lambda)$ for some negligible

function negl over the security parameter of the HTLP. In addition, assuming the correctness of the underlying SS scheme, we have

$$\begin{aligned}
&\Pr[\omega_i(\Gamma, (\sigma'_i, \sigma_{-i})) = s] \\
&= \Pr[(\mathsf{HP.PSolve}(pp_1, \mathcal{T}^{\epsilon}, \mathcal{Z}_r) \to s_r) \cap (\mathsf{Recon}(pp, \{s_1, \ldots, s_r\}) \neq \perp]) \\
&\leq (1/2 + \mathsf{negl}(1^{\lambda})) \cdot \mathsf{negl}'(1^{\lambda}) \leq \mathsf{negl}''(1^{\lambda}),
\end{aligned}$$

for some negligible function negl''.

Secondly, assuming the secrecy of the SS scheme, for players P_{-i} with $k < r$ shares, the probability $\Pr[\mathsf{Recon}(pp, \{s_1, \ldots, s_k\}) \neq \perp] \leq \mathsf{negl}'(1^{\lambda})$. That is, $\Pr[\mathsf{Recon}(pp, \{s_1, \ldots, s_k\}) = \perp] \in [1 - \mathsf{negl}'(1^{\lambda}), 1]$, and so $\Pr[\omega_{-i}(\Gamma, (\sigma'_i, \sigma_{-i})) = \perp] \in [1 - \mathsf{negl}'(1^{\lambda}), 1]$. Given this fact,

$$\begin{aligned}
\Pr[\mathtt{Event}_2] &= \Pr[(\omega_i(\Gamma, (\sigma'_i, \sigma_{-i})) = s) \cap (\omega_{-i}(\Gamma, (\sigma'_i, \sigma_{-i})) = \perp)] \\
&\leq \mathsf{negl}''(1^{\lambda}) \cdot 1 = \mathsf{negl}''(1^{\lambda}).
\end{aligned}$$

Given the two disjoint scenarios in which P_i can reconstruct and learn s, we have;

$$\begin{aligned}
\Pr[\omega_i(\Gamma, (\sigma'_i, \sigma_{-i})) = s] &= \Pr[\mathtt{Event}_1 \cup \mathtt{Event}_2] = \Pr[\mathtt{Event}_1] + \Pr[\mathtt{Event}_2] \\
&= \Pr[\omega_{-i}(\Gamma, (\sigma'_i, \sigma_{-i})) = s] + \Pr[\mathtt{Event}_2] \\
&\leq \Pr[\omega_{-i}(\Gamma, (\sigma'_i, \sigma_{-i})) = s] + \mathsf{negl}''(1^{\lambda}).
\end{aligned}$$

Therefore Equation 6.1 is satisfied.

Soundness Following Definition 64 from Section 6.3, we want to show that there exists some negligible function negl under security parameter λ such that

$$\Pr[\omega_{-i}(\Gamma, (\sigma'_i, \sigma_{-i})) \notin \{s, \perp\}] \leq \mathsf{negl}(1^{\lambda}) \tag{6.2}$$

Recall Section 6.4.1 which presented preliminary definitions related to soundness from a checking share. Proving soundness is a two-step approach, the first step being to demonstrate that the checking share in our construction can be used in place of the sound membership oracle of Definition 78, as protocol-induced auxiliary-information to ensure soundness.

Claim 4. *Let* $(\Gamma_{f_k}, \overrightarrow{\sigma_{f_k}})$ *be the reconstruction mechanism* $(\Gamma_f, \overrightarrow{\sigma_f})$ *with* $P_j \in P$ *possessing* k *shares, checking share* s_0*, and reconstructed value* $s' \in \mathcal{S}_\lambda$*. Let* $s_0 = (y_0, f(y_0))$ *for some input* y_0 *computed by the dealer in the sharing phase; if* $f_k(y_0) = f(y_0)$*, then* P_j *concludes that* $s' = s$*, otherwise* P_j *concludes that* $s' \neq s$*.*

Proof. By Definition 78, we can assume that the following conditions hold:

1. $\Pr[f_k(y_0) = f(y_0)] \leq \mathsf{negl}(1^\lambda)$ for $s' \neq s$.
2. $\Pr[f_k(y_0) \neq f(y_0)] \leq \mathsf{negl}(1^\lambda)$ for $s' = s$.

Function f is honestly constructed by the dealer during the sharing phase such that the secret can be recovered from it. Due to the security of the underlying SS scheme, which we assume in Theorem 8, the function f must be one-to-one. This uniqueness property means that the function is unique and given some reconstructed value $s' \neq s$, the probability of f and reconstructed function f_k being equal for a specific input is negligible. So the first inequality holds.

The second inequality holds by the correctness of our construction, proved in Theorem 8. This says that the players will always output s when they have reconstructed $s' = s$, except with negligible probability. Thus the checking share follows Definition 78 and can be used to provide soundness in the reconstruction phase $(\Gamma_{f_k}, \overrightarrow{\sigma_{f_k}})$. □

Following Claim 4, we can now proceed to the second stage of proving soundness in the following.

Claim 5 (Soundness with a Checking Share)**.** *Let* $\Gamma_{r,r+1} = (\Gamma_{f_k}, \overrightarrow{\sigma_{f_k}})$ *be the (r, r+1) fair secret reconstruction phase of our construction (Section 6.5), assuming player* $P_j \in P$ *has* k *shares, access to protocol-induced auxiliary information in the form of a checking share* $s_0 = (y_0, f(y_0))$*, defined as above such that function* f *is determined by the dealer to reconstruct the secret. Then, the reconstruction phase is sound.*

Proof. By the assumptions of Theorem 8, the reconstruction phase of our construction is fair despite every player having access to protocol-induced auxiliary information in the form of a checking share s_0 and satisfies correctness.

Suppose that deviant player P_i follows an alternative strategy σ_i' which sees player P_i follow their normal strategy σ_i for the first k rounds, and then deviate in the round $(k+1)$ by quitting communication.

Regardless of the round that P_i deviates in with respect to the value r, assuming players have access to the checking share of the (r, r+1) such that Claim 4 holds, the fair reconstruction phase $(\Gamma_{f_k}, \overrightarrow{\sigma_{f_k}})$ for rational players satisfies soundness. More precisely, given an value $s' \in \mathcal{S}_\lambda$, suppose P_i either reconstructs $s' \notin \{s, \perp\}$ with r shares or $k < r$ shares:

$$\Pr[\omega_{-i}(\Gamma_{f_k}, (\sigma'_i, \sigma_{-i,f_k})) \notin \{s, \perp\}] = \Pr[(\omega_{-i}(\Gamma_{f_k}, (\sigma'_i, \sigma_{-i,f_k})) = s')]$$
$$= \Pr[(\mathsf{Recon}'(pp', s_0, \{s_1, \ldots, s_r\}) = s') \cup (\mathsf{Recon}'(pp', s_0, \{s_1, \ldots, s_k\}) = s')]$$
$$= \Pr[\mathsf{Recon}'(pp', s_0, \{s_1, \ldots, s_r\}) \neq s] + \Pr[\mathsf{Recon}'(pp', s_0, \{s_1, \ldots, s_k\}) = s']$$
$$\leq \mathsf{negl}(1^\lambda) + \Pr[\mathsf{Recon}'(pp', s_0, \{s_1, \ldots, s_k\}) = s'] = \mathsf{negl}(1^\lambda) + \Pr[f_k(y_0) = f(y_0)]$$
$$\leq \mathsf{negl}(1^\lambda) + \mathsf{negl}'(1^\lambda) \leq \mathsf{negl}''(1^\lambda),$$

for some negligible function negl'', where negligible function negl' in the penultimate inequality comes from the fact that $\Pr[f_k(y_0) = f(y_0)] \leq \mathsf{negl}'(1^\lambda)$ when $s' \neq s$. Therefore Equation 6.2 is satisfied. □

□

6.5 A Concrete FRSS Construction

In this Section, we propose a concrete construction of an FRSS scheme. First, we present the building blocks used before formalising our construction.

6.5.1 Building Blocks

In our construction, we will use a *multiplicative* homomorphic time-lock puzzle (Definition 65). For construction Π_{FRSS} to satisfy Definition 69, the secret sharing scheme used as a building block must be compatible with an MHTLP. Thus, we use a homomorphic multiplicative threshold secret sharing scheme, formally presented in the ensuing.

Let $P = \{P_1, \ldots, P_n\}$ be a group of n players, D be the honest dealer, $\mathcal{S}_\lambda$ the set of secrets under security parameter λ. Assume the share for P_i, $i \in [n]$ is selected from the set $\mathcal{S}_i$[4]. A (t, n) threshold scheme Π_{SS} consists of D running setup and sharing algorithms and taking a secret input from $\mathcal{S}_\lambda$, mapping $\mathcal{S}_\lambda \to \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ to assign shares to every player in P of the form (x_i, s_i) for a $(t-1)$-degree polynomial

$$f(x) = s + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$$

[4]We drop the λ in the set of shares for simplicity of notation.

such that $\{a_1, \ldots, a_{t-1}\} \xleftarrow{\$} \mathcal{S}_\lambda$ and $\{x_1, \ldots, x_n\} \in \mathcal{S}_\lambda$ are distinct inputs, with outputs $s_i \in \mathcal{S}_i$ corresponding to x_i, $\forall i \in [n]$. The reconstruction algorithm sees a subset of the players, $A \subseteq P$ either return the secret or the phase of reconstruction fails.

In *multiplicative* homomorphic threshold secret sharing [119] over groups [44, 42], the secret space S_λ is a finite group with respect to the operation $\otimes$. For any t distinct players P_i, secret space subset $\mathcal{S}' \subseteq \mathcal{S}_\lambda$ such that $|\mathcal{S}'| = t$ and $\mathcal{S}' = \{s_{i_1}, \ldots, s_{i_t}\}$, there exists a family of function for all $l \in [t]$ such that

$$f_{i_l, \mathcal{S}'} : \mathcal{S}_{i_l} \to \mathcal{S}$$

with $\{i_1, \ldots, i_l\}$ publicly ordered with the following property.

For any secret $s \in \mathcal{S}_\lambda$ and shares $s_{i_1}, \ldots, s_{i_t}$ that have been distributed to P_i by the dealer D on input s, the secret can be expressed as follows,

$$s = f_{i_1, \mathcal{S}'}(s_{i_1}) \otimes \ldots \otimes f_{i_t, \mathcal{S}'}(s_{i_t})$$

such that, provided the share set $\mathcal{S}_i$ is a group, the function $f_{i, \mathcal{S}'} : \mathcal{S}_i \to \mathcal{S}_\lambda$ is a group homomorphism for $\forall i, \mathcal{S}'$; then $\forall l \in [t]$, define

$$f_{i_l, \mathcal{S}'}(s_{i_l}) = f(x_{i_l}) \prod_{j \in \mathcal{S}', j \neq i_l} \frac{-x_j}{(x_{i_l} - x_j)},$$

assuming that $(x_{i_l} - x_j) \in \mathcal{S}_\lambda$ with a multiplicative inverse. In other words, a unit of $\mathcal{S}_\lambda$.

6.5.2 Instantiation

Our final contribution is to provide a concrete, fair, and sound RSS construction. To achieve this we use a specific variant of Shamir's SS scheme (see above and Section 2.4) and a multiplicative HTLP scheme.

We chose to define a *concrete* scheme so we can demonstrate the improved efficiency of the scheme versus if a standard TLP had been used. So far, we have detailed an HTLP scheme (Definition 65) and provided our generic construction with a homomorphic operation Ψ in Definition 69. For the concrete instantiation, we will make use of a multiplicative HTLP (MHTLP) which is multiplicatively homomorphic over the ring $(\mathbb{J}_N, \cdot)$ using the operator $\otimes$ (see Section 2.3).

High Level Idea We instantiate our construction as follows:

- A multiplicative homomorphic threshold secret sharing scheme $\Pi_{\mathsf{SS}} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Recon})$ (Section 2.4), for a secret space $\mathcal{S}_\lambda = \mathbb{J}_N$ over a finite group with respect to multiplication, defined as in [119, 44, 42],
- A MHTLP scheme $\Pi_{\mathsf{MHTLP}} = (\mathsf{MHP.Setup}, \mathsf{MHP.Gen}, \mathsf{MHP.Eval}, \mathsf{MHP.Solve})$ (Section 6.5.1), which is multiplicatively homomorphic over a ring $(\mathbb{J}_N, \cdot)$.

The multiplicative operator $\otimes$ enables the dealer to split the ith share, for some $i \in [m]$, of the secret into n sub-shares in the following way,

$$s_{i,n} = s_i \cdot \left(\prod_{j=1}^{n-1} s_{i,j} \right)^{-1}$$

enabling players to homomorphically evaluate sub-puzzles by running $\mathsf{MHP.Eval}$, and $\mathsf{MHP.Solve}$ on the master puzzle output from evaluation to obtain the correctly reconstructed share for the ith round. To ensure the soundness of the concrete instantiation, the dealer distributes a checking share s_0 to all players. This is computed as $s_0 = f(y_0) \pmod N$, for some polynomial f determined in the setup phase of the scheme from a multiplicative homomorphic threshold SS scheme.

Our Concrete Construction To instantiate an FRSS construction with a multiplicative homomorphic operator, we use a concrete MHTLP scheme and a concrete SS scheme, described in the preceding subsection. To do so, we consider an honest dealer D, n rational players $P = \{P_1, \ldots, P_n\}$ and efficiently samplable distribution $\mathcal{S}_\lambda = \mathbb{J}_N$ (see Section 2.3 for further details[5]) under security parameter λ for secret $s \in \mathbb{J}_N$. Here, we define $N = p \cdot q$ to be a strong RSA prime (See Section 2.3, Assumption 9, [73]), time hardness parameter $\mathcal{T}$, and efficiently samplable discrete distributions $\mathcal{G}, \mathcal{G}'$ [6]. The multiplicative homomorphic SS scheme is an $(r, r+1)$ threshold scheme such that, for value $r \xleftarrow{\$} \mathcal{G} = \{1, \ldots, N^2\}$, given $r+1$ shares of s, a threshold of r shares is sufficient to reconstruct s. Observe, in the sharing phase, the honest dealer D takes as input the secret $s \in \mathbb{J}_N$. We now present the formal definition.

Definition 80 (Concrete FRSS Construction)**.** *Given security parameter λ, time hardness parameter $\mathcal{T}$, an efficiently samplable distribution of the set of secrets $\mathcal{S}_\lambda = \mathbb{J}_N$*

[5] Recall, $\mathbb{J}_N$ is the cyclic group of elements of $\mathbb{Z}_N^*$ with Jacobi symbol $+1$.

[6] We suggest geometric distributions over $\mathbb{N}$, with the parameter dependent on players outcome preferences (see Section 6.4.1), denoted β. Let β be the probability of the first success in a repeated Bernoulli trial, that is, repeatedly tossing a biased coin until the first head appears [61].

with the operator $\otimes$, secret $s \in \mathcal{S}_\lambda$, efficiently samplable discrete distributions $\mathcal{G}, \mathcal{G}'$, we construct a concrete FRSS scheme with reconstruction phase in the non-simultaneous setting as a tuple of three PPT algorithms $\Pi_{\mathsf{FRSS}} = (\mathsf{Setup}', \mathsf{Share}', \mathsf{Recon}')$. Construction Π_{FRSS} is built from a multiplicative homomorphic secret sharing scheme $\Pi_{\mathsf{SS}} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Recon})$ and an MHTLP scheme $\Pi_{\mathsf{MHTLP}} = (\mathsf{MHP.Setup}, \mathsf{MHP.Gen}, \mathsf{MHP.Solve}, \mathsf{MHP.Eval})$, and we define the scheme as follows:

1. $\mathsf{Setup}'(1^\lambda, \mathcal{T}) \xrightarrow{\$} pp'$: given the inputs $1^\lambda, \mathcal{T}$, the dealer performs the following steps:

   (a) $\mathsf{MHP.Setup}(1^\lambda, \mathcal{T}) \xrightarrow{\$} pp_1$: Uniformly sample $\tilde{g} \xleftarrow{\$} \mathbb{Z}_N^*$ and set $g := -\tilde{g}^2 (\text{mod } N)$ such that $g \in \mathbb{J}_N$, where g is the generator of $\mathbb{J}_N$. Compute $h := g^{2^\mathcal{T}}$.[7] Define $pp_1 := (\mathcal{T}, N, g, h)$.

   (b) $\mathsf{Setup}(1^\lambda, \mathcal{T}) \xrightarrow{\$} pp_2$: Define $pp_2 := \{p, \{y_0, \ldots, y_r\}\}$ with prime $p > \{s, n\}$ and with $\{y_0, \ldots, y_r\} \xleftarrow{\$} \mathbb{J}_N$.

   Additionally, let $r \xleftarrow{\$} \mathcal{G}$ be the sampled revelation value r and $d \xleftarrow{\$} \mathcal{G}'$ a random value, and output r, d and public parameters $pp' := \{pp_1, pp_2\}$.

2. $\mathsf{Share}'(pp', s) \xrightarrow{\$} \{s_0, \{list_1, \ldots, list_n\}\}$: takes as input the secret $s \in \mathbb{J}_N$ and public parameters pp'. The output consists of a checking share s_0 and lists labelled $list_j$ for $j \in [n]$, each composed of m sub-puzzles for $m = r + d$.

   (a) $\mathsf{Share}(pp_2, s) \xrightarrow{\$} \{s_0, \{s_1, \ldots, s_r\}\}$: inputs are the public parameters pp_2 and secret $s \in \mathbb{J}_N$. The dealer outputs $(r+1)$ shares s_i, determined as follows: D chooses a random r degree polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_r x^r$ where $a_0 = s$ and $\{a_1, \ldots, a_r\} \xleftarrow{\$} \mathbb{J}_\mathbb{N}$. The dealer then determines shares s_i for $i \in \{0, \ldots, r\}$, computed as $s_i = f(y_i) \text{ (mod } N)$ such that $s_i \in \mathbb{J}_N$.

   (b) $\{s_{r+1}, \ldots, s_m\} \xleftarrow{\$} \mathbb{J}_N$, randomly sample d fake shares from secret space $\mathbb{J}_N$.

   (c) In our construction, shares $\{s_1, \ldots, s_r\}$ are split into n sub-shares so that they can be distributed to P (along with other, fake sub-shares). The checking share s_0 is publicly broadcast and used to check soundness. Distributing sub-shares to P enables just one share to be reconstructed per round, gradually releasing the secret to players in the reconstruction phase.

   How are sub-shares created by the dealer during the setup phase?

[7] The dealer can optimise h by reducing the exponent modulo $\phi(N)/2$ first, as suggested in [97].

Here, D creates n sub-shares for each of the m shares they have derived as follows: for a fixed $i \in [m]$, the dealer samples sub-puzzles $s_{i,j} \xleftarrow{\$} \mathbb{QR}_N$ for $j \in \{1, n-1\}$ and defines the nth sub-share to be

$$s_{i,n} = s_i \cdot \left(\prod_{j=1}^{n-1} s_{i,j}\right)^{-1}$$

for every $i \in [m]$. The ith share is defined as

$$s_i = \prod_{j \in [n]} s_{i,j} \pmod N. \ (*)$$

(d) Next, D generates sub-puzzles $\mathcal{Z}_{i,j}$ to embed the sub-shares as follows. It runs $\mathsf{MHP.Gen}(pp_1, s_{i,j})$ on input public parameters pp_1 and sub-shares $s_{i,j}$: $\forall i \in [m]$, $\forall j \in [n]$, uniformly sample $r_{i,j} \xleftarrow{\$} \{1, \cdots, N^2\}$, and generate the elements $u_{i,j} := g^{r_{i,j}} \pmod N$, $v_{i,j} := h^{r_{i,j}} \cdot s_{i,j} \pmod N$.
Define the n sub-puzzles for the ith share to be $\mathcal{Z}_{i,j} := (u_{i,j}, v_{i,j}) \in \mathbb{J}_N^2$.
The dealer outputs sub-puzzles $\mathcal{Z}_{i,j}$, $\forall i \in [m], \forall j \in [n]$.

(e) D distributes $\text{list}_j = \{\mathcal{Z}_{1,j}, \cdots, \mathcal{Z}_{r,j}, \mathcal{Z}_{r+1,j}, \cdots, \mathcal{Z}_{m,j}\}$ to the corresponding player P_j, for every $j \in [n]$.

3. The dealer distributes the following:

(a) D broadcasts $\{pp', s_0\}$ to all P the public parameters pp' and checking share $s_0 = f(y_0) \pmod N$, determined in the second step above.

(b) D distributes list_j to P_j for every $j \in [n]$.

In the reconstruction phase, whilst sending sub-puzzles for communication phase k, for some $1 < k \leq m$, players simultaneously process the set of n sub-puzzles that they obtained in the previous round. In the processing phase, assume that players are in the round $(k-1)$ of processing. For any $j \in [n]$, P_j does the following:

1. $\mathsf{MHP.Eval}(pp_1, \mathcal{T}, \otimes, \mathcal{Z}_{k-1,1}, \cdots, \mathcal{Z}_{k-1,n}) \rightarrow \mathcal{Z}_{k-1}$: input public parameters pp_1, hardness parameter $\mathcal{T}$, and the list of n sub-puzzles for the $(k-1)$th round. Compute $u_{k-1} := \Pi_{j=1}^{n} u_{k-1,j} \pmod N$, $v_{k-1} := \Pi_{j=1}^{n} v_{k-1,j} \pmod N$ and output the homomorphic puzzle $\mathcal{Z}_{k-1} := (u_{k-1}, v_{k-1})$ of share s_{k-1}.

2. $\mathsf{MHP.Solve}(pp_1, \mathcal{T}, \mathcal{Z}_{k-1}) \rightarrow s_{k-1}$: on input, the public parameters pp_1, hardness parameter $\mathcal{T}$, and puzzle share $\mathcal{Z}_{k-1}$, compute $w_{k-1} := u_{k-1}^{2^{\mathcal{T}}} \pmod{\text{N}}$ by sequential squaring. Output $s_{k-1} := v_{k-1}/w_{k-1}$ as the solution to the puzzle. Move to reconstruct s from shares $\{s_1, \ldots, s_{k-1}\}$.

The dealer in a SS scheme will have created sub-shares using $(*)$ for the concrete instantiation of our construction using an MHTLP. As a consequence, following the correctness of the MHTLP construction [97], $s_{k-1} = \Pi_{j=n} s_{i,j}$, which is precisely the $(k-1)$th share.

3. $\mathsf{Recon}'(pp', s_0, \{s_1, \ldots, s_{k-1}\}) \rightarrow \{s, \perp\}$: from the secret sharing scheme, run $\mathsf{Recon}(pp_2, \{s_1, \ldots, s_{k-1}\}) \rightarrow \{s, \perp\}$ with inputs of the public parameters pp_2 and $(k-1)$ reconstructed shares of the secret $\{s_1, \ldots, s_{k-1}\}$.

   Outputting the secret is done firstly by players using polynomial interpolation (see Section 6.5.1):

$$f'(x) = \prod_{i \in [k-1]} \left[f(y_i) \cdot \prod_{l \in [k-1], l \neq i} \frac{(x - x_l)}{(x_i - x_l)} \right] = a'_0 + a'_1 x + a'_2 x^2 + \ldots + a'_{k-1} x^{k-1}$$

   using the shares $\{s_1, \ldots, s_{k-1}\}$ such that $a'_0 = s'$.

4. P_j uses checking share s_0 to see if $f'(s_0) = f(s_0)$. If this equality holds, it must be the case that $(k-1) = r$ and so $s' = s$ is the output. If $(k-1) \neq r$, the equality will not hold, which means that $s' \neq s$. The output may be $\perp$ depending on the following.

5. If P_j outputs $\perp$, but no player quits in round k of communication and every player $P_j \in P$ has $list_j \neq \emptyset$, then players go to $(k+1)$th round of the reconstruction phase. Otherwise, the output $\perp$ is the outcome of the reconstruction game.

The concrete instantiation of our scheme relies on standard cryptographic and number-theoretic assumptions regarding the building blocks it is based on. Namely, Shamir's SS [110] and the MHTLP scheme from [97]. More specifically, letting the modulus N be a strong RSA modulus; if the sequential squaring (Definition 10 in Section 2.3) and Decisional Diffie-Hellman (Definition 4 in Section 2.3) assumptions hold over $\mathbb{J}_N$, then the MHTLP scheme of [97] is proven secure.

In the proceeding, we will conclude by discussing the efficiency of our concrete construction.

6.5.3 Efficiency Considerations

In general, our concrete construction (Π_{FRSS}) is less efficient than a standard SS scheme (Definition 19, Section 2.4) like Shamir's scheme [110]. This is due to the inclusion of an MHTLP building block, which increases the computational burden on the players participating in reconstruction as they need to evaluate and solve a puzzle in each round. Nevertheless, we believe the time-delay feature of a TLP is essential to satisfying fairness in schemes seeking to also satisfy soundness, and this is especially important for rational SS schemes in which both properties are highly desirable.

However, in the context of RSS schemes utilising time-delay mechanisms, our scheme is the first to use a time delay with a homomorphic property (recount Definition 65 of an HTLP in Section 6.3). The consequence of this fact is that Π_{FRSS} is generally more efficient in direct comparison to other SS schemes that incorporate a time delay. To provide context, we will start by comparing the efficiency of Π_{FRSS} built from a *HTLP vs. TLP*. Proceeding with this, we will compare the efficiency of our instantiation to a closely aligned RSS scheme, proposed by the authors of [40], which utilises a different type of time delay.

HTLPs vs. TLPs The homomorphic property of an HTLP scheme means that solving a puzzle, the most computationally expensive step for the players, need only be run once rather than n times in the processing phase of our scheme. Given puzzles have a time hardness parameter of $\mathcal{T}$, the computational cost of running $\mathsf{HP.Solve}$ is $\Omega(2^{\mathcal{T}})$-steps [8].

Indeed, if we were to use a standard TLP in the processing phase of our scheme, each player would independently have to solve each of the n sub-puzzles using $\mathsf{P.Solve}$, and then evaluate the n sub-shares to obtain the share for that round. Conversely, by using a HTLP in our scheme, players must run $\mathsf{HP.Eval}$ *once* over the n sub-puzzles, outputting a master puzzle, and proceed to run $\mathsf{HP.Solve}$ *once* on this master puzzle to obtain the corresponding share. Thus, HTLPs are more efficient by a linear factor of n, where n corresponds to the number of players participating in the reconstruction phase.

We highlight the importance of the homomorphic property in the HTLP scheme, which is key to satisfying the property of compactness (Section 6.3.1, Definition 68). This

[8] The computational complexity of the puzzle-solving algorithm $\mathsf{HP.Solve}$ is the same as solving a standard TLP scheme using algorithm $\mathsf{P.Solve}$.

means that the runtime of homomorphically evaluating puzzles is bounded above by a fixed polynomial that only depends on the security parameter λ and not the time hardness parameter $\mathcal{T}$. Otherwise, the trivial solution would be indeed to use a standard TLP scheme.

In more detail, we use a multiplicative-homomorphic TLP rather than a standard TLP to reduce the computational overhead for players by a linear factor. Recall, the *compactness* property of the MHTLP scheme (Definition 68, Section 6.3.1) ensures, informally, that the length of evaluated puzzle $\mathcal{Z}_i$ of share s_i, only depends on the security parameter λ.

As a consequence, in the context of our construction, it is more efficient for players to solve the evaluated puzzle $\mathcal{Z}_i$ than it is to solve individual sub-puzzles $\mathcal{Z}_{i,j}$ for a fixed $i \in [m]$, $\forall j \in [n]$, obtain sub-shares $s_{i,j}$ and using $(*)$ as a function to evaluate the share s_i. More succinctly, it is more efficient for the players to evaluate the function of embedded sub-shares than it is for them to evaluate the function of the decrypted sub-puzzles.

To be precise, Π_{FRSS} requires exactly one run of $\mathsf{MHP.Eval}$, which translates to n multiplications. This is followed by one run of $\mathsf{MHP.Solve}$ with complexity $\Omega(2^{\mathcal{T}})$. If we used a plain TLP in the instantiation instead, assuming the same parameters, we require n runs of $\mathsf{P.Solve}$ of complexity $\Omega(2^{\mathcal{T}})$, followed by n runs of $\mathsf{P.Eval}$, which means n multiplications.

Comparison to [40] It is important to highlight our scheme closely follows the work of [40]. The authors of [40] achieve utility (preference of outcome) independence by designing a scheme that encrypts shares (computed using Shamir's SS scheme) using *memory-bounded functions* (MBFs) and further splits the encrypted shares into sub-shares, distributed to players.

During processing, players independently evaluate the encrypted sub-shares to obtain the encrypted share, decrypt and then reconstruct the polynomial to obtain the secret. They use a *specific* form of side-information, called a checking share, which is an actual share of the secret that players can use to confirm they have reconstructed the correct secret, thus achieving soundness. In other words, the construction of [40] requires players to independently decrypt each share before they proceed to the secret reconstruction using the Shamir SS scheme.

We believe our scheme is more efficient due to a distinction in design stemming from the homomorphic property of an HTLP. First and foremost, following the preceding discussion, the advantage of using HTLPs is an efficiency improvement for the honest players evaluating puzzles in comparison to using standard TLPs or a time-delay encryption mechanism. Recall those traditional time-delay methods can create a so-called 'computational bottleneck'.

Moreover, the [40] scheme involves linearly evaluating sub-shares encrypted using memory-bound functions for the time-delay to ensure fairness of the scheme and reconstructing the secret using Shamir's SS scheme. In contrast, our generic construction uses *CPU-bound HTLPs* to ensure a time delay in rounds of the scheme. In our opinion, basing the time-delay primitive on CPU-bound functions as opposed to memory-bound functions captures a more realistic, inexpensive way to implement a SS scheme construction.

In other words, a justification for using MBFs in [40, 96] is that disparities in the computational power of players can cause unfairness when using standard TLPs for the time delay. However, with reasonable assumptions on the CPU power of players, this disparity is not significant. That is, we argue the efficiency improvement in using an HTLP mitigates any disparities in CPU power between players. To see this, evaluating several puzzles homomorphically, and then solving just *one* puzzle requires fewer computational steps than solving individual puzzles and evaluating a function over the outputs. Further, it holds that processors are faster than memory and scale better; even more so, fast memory is considerably more expensive. We conjecture that it is easier in practice to raise the computational requirements of a player than it is memory accesses, up to a point, as adding more processors to a computer is more accessible than making memory access faster.

6.6 Summary and Outlook

In this Chapter, we proposed a generic construction for a fair and sound rational secret-sharing scheme in the non-simultaneous setting of communication, built from homomorphic time-lock puzzles and any correct threshold secret-sharing scheme. Further, we proposed a concrete scheme, whose security relied on standard assumptions, that was built from multiplicative secret sharing and multiplicative HTLPs. In defining a concrete FRSS scheme we were able to analyse efficiency and make comparisons to

existing RSS literature incorporating a time-delay mechanism. Lastly, we showed that our generic construction of an FRSS provably satisfied the properties of correctness, secrecy, fairness, and soundness.

Advances in HTLPs Recently, similar areas of research to our work in this Chapter (related to our paper [83]) include contributions from the authors of [115, 95] whose research focuses on the practicalities of time-based primitives like HTLPs, as well as verifiable delay functions (VDFs) [23], and timed commitments [23, 78].

In particular, the authors of [115] propose a decentralised service called *OpenSquare* which outsources the computation of repeated modular squaring, inherent to HTLPs, via smart contracts. Their motivation is to address the practicalities of HTLPs for large-scale application adoption. Specific concerns that the authors address are the computational effort in performing the squaring operation and the prediction of the time hardness parameter $\mathcal{T}$ in an HTLP scheme so that security holds in the real world. Furthermore, the authors of [95] look at *practical HTLPs* in the sense of applications and *verifiability* that is missing in current HTLP schemes. One of their proposed solutions is a protocol allowing a puzzle solver to convince others of the solution's correctness or invalidity of the puzzle, to save unnecessary additional puzzle solvers performing the computation.

Future Work In our opinion, an obvious avenue of future work for this Chapter is to implement and optimise our instantiation of an FRSS construction. In line with our previous Chapters (4 and 5), we also think it is important for the security of real-world applications to consider an untrusted dealer of secret shares.

Moreover, we believe it is natural to extend the definition of utility functions (see Section 6.4.1), and consequently, the definition of fairness of our FRSS scheme to incorporate parameters of time. This idea is supported by the recent work by the authors of [55] who investigated the relationship between the time versus monetary gain of rational decision-making in the game-theoretical setting. They demonstrated that individuals have similar preferences with respect to either type of gain resulting from the outcome unless the gain in time or money is significant. In the context of rational secret sharing, we leave this desirable extension for future work.

In addition, we think it is worthwhile to integrate the ideas and contributions from follow-up work [78, 115, 95]. In particular, we think it is of interest to understand

how to incorporate the ideas proposed in [95] in which a so-called puzzle solver can convince others of the correct output. This property may be usefully applied in our work to attain soundness using alternative, more efficient methods to protocol-induced side information. Moreover, the outsourced HTLP puzzle solution (OpenSquare) [115] traverses research into HTLP and UE primitives featured in this Thesis. Thus, we believe it worthwhile to consider how UE and time-delay primitives can be combined to suit the application of outsourced solutions like OpenSquare.

Recall the contributions from Chapter 5, in which we proposed a generic dynamic multi-server UE (DMUE) construction built from secret sharing. We think it is beneficial to UE literature to explore the gap between the contributions in this Chapter and DMUE. In particular, it would be interesting to explore whether we can build DMUE using an FRSS construction. In doing so, we can model DMUE security using game theory to demonstrate the satisfaction of desirable properties of fairness and soundness when viewing servers are rational players.

Chapter 7

Concluding Remarks

In this Thesis, we explored the relationship between time and communication in the context of the updatable encryption primitive and secret sharing protocols. First, we lifted updatable encryption to the public key setting and considered different variations of this primitive depending on the application, with an emphasis on security modelling. In the second part of this Thesis, we utilised time delay as a tool to satisfy certain properties in a rational secret-sharing scheme. We will briefly recount each Chapter, followed by a discussion on the Chapters related to the PKUE primitive, and conclude by remarking on future work.

In Chapter 3 we introduced a public-key definition (PKUE) of the updatable encryption primitive [24, 52, 92] followed by a new security framework for the public key setting. We defined a standard notion of security in UE literature known as *ciphertext unlinkability*, which intuitively captured the advantage an adversary has in determining the origin of the ciphertext, be it that it has been produced via fresh encryption or through the update process. Our second security definition was a new notion called *epoch confidentiality* formalised to encompass the leakage of metadata, such as the number of key rotations of a ciphertext, which directly translates to the age of a ciphertext.

In Chapter 4 we extended our understanding of the PKUE primitive in the context of *public key infrastructures* (PKI). Of primary concern, the *key generation centre* (KGC) that generates the public and secret key pairs associated with an epoch could behave dishonestly, thus, we sought a way to reduce trust in the KGC. The solution we settled on was to incorporate the *certificateless public key encryption* primitive (CL-PKE), introduced by [3], as the underlying encryption scheme used to build a PKUE scheme. The result was a new certificateless public-key updatable encryption primitive labelled

CLUE, followed by a security framework and provably secure concrete construction built from key-homomorphic pseudorandom functions (KH-PRFs) [24].

In Chapter 5 we chose to explore the assumed trust in the server performing PKUE ciphertext updates. We believe a PKUE scheme would lack resilience if a lone server was relied upon to update ciphertexts over a long period. Thus, we were motivated to define a multi-server PKUE scheme. The primitive we settled on, labelled DMUE, further extended this idea to support *dynamic* changes in the servers participating, allowing for a change in a committee of servers at the start of each epoch. In addition to providing a multi-server security framework capturing confidentiality and integrity notions, we formalised a *generic* DMUE construction. Intuitively, the construction was built from single-server PKUE and a *dynamic-proactive secret sharing* (DPSS) protocol, such that a *threshold* number of servers participate to reconstruct the master token for that epoch using *token shares*.

In Chapter 6 our attention turned to the role of time in communication between participants of a *rational secret sharing* (RSS) protocol to satisfy two desirable properties. Concretely, we were motivated to define a more efficient construction of an RSS scheme that satisfied *fairness* and *soundness* when players communicate non-simultaneously. Inspired by secret sharing literature [96, 40], we utilised the use of a *time-delay* mechanism to mitigate the unfairness in an RSS scheme when *side information* is being used to ensure soundness. We settled on a *homomorphic time lock puzzle* (HTLP) [97] time delay mechanism to build a generic construction labelled FRSS, with the homomorphic property of the HTLP being a new and essential feature to prevent the computational bottleneck inherent in other time-delay mechanisms.

Discussions and Conclusions We wish to clarify the running themes and distinctions of our PKUE contributions. To be clear, Chapters 4 and 5 build upon the foundational Chapter 3 with respect to public-key updatable encryption (PKUE), but there are clear adaptations and omissions regarding security notions as the reader progresses through this Thesis. Specifically, the notion of epoch confidentiality from Chapter 3 is not modelled for CLUE or DMUE. This is an intentional decision as we focused on mitigating issues outside of the leakage of metadata in the latter two Chapters. Specifically, the problems created when epoch secret keys are held in escrow and a single point of failure respectively. Whilst our focus turned to new concerns, we observe that it should be possible to achieve instances of CLUE and DMUE satisfying epoch confidentiality, but this has been reserved for future work.

Moreover, the contrast in security modelling for Chapters 3 and 4 versus Chapter 5 is due to the choice of probabilistic versus deterministic ciphertext updates respectively. In particular, the latter choice meant we could model a strictly stronger ciphertext confidentiality notion compared to the previous Chapters, and more importantly, model a new ciphertext integrity notion which is impossible to capture when ciphertexts are re-randomised during the update process. We highlight that it is possible to design deterministic PKUE and CLUE should a given application require ciphertext integrity in future. Nevertheless, it was a purposeful decision to diversify the Definitions and security modelling of PKUE with bi-directional key and ciphertext updates, to highlight the PKUE primitives' security capabilities and suit the many different scenarios for which it can be applied.

Furthermore, we observe that there has been a recent trend in UE literature to move towards designing no-directional tokens which are tokens independent from the epoch secret keys [104, 112, 31]. As a result, the need for complex models to capture inferable information in our PKUE setting can be removed. Alas, the no-directional setting is in its infancy relative to bi-directional UE and only recently have constructions been proposed from more standard cryptographic building blocks. In addition, there is an ongoing challenge in no-directional literature to capture a notion of ciphertext confidentiality stronger than chosen plaintext security. Given our emphasis on attaining the strongest level of ciphertext confidentiality possible in the chosen setting of PKUE, we decided to continue research in the bi-directional setting but we intend to consider no-directional updates outside of the confines of this Thesis. For instance, it may be desirable to omit epoch confidentiality to achieve more efficient PKUE schemes with no-directional, probabilistic updates.

Future Work Globally, we are transitioning from cloud computing to the "ubiquitous computing" phase of the digital revolution. As such, the threat landscape of communication and storage of sensitive information is evolving. As a consequence, new cryptographic schemes are required to continue guaranteeing desirable levels of security balanced with the need for application-specific solutions. By way of illustration, internet-of-things (IoT) devices, vehicle-to-vehicle communication, and smart cities are all scenarios currently being researched by the cryptographic community, with each case necessitating the design of new protocols to support their complex use-case architectures. We believe that privacy-enhancing technologies and time in communication are essential components of such frameworks, for instance, there need

to be efficient, lightweight mechanisms for updating cryptographic keys in group key management protocols after a new device has entered or left a network.

In the context of this Thesis, improving efficiency has continued to be an open challenge for both updatable encryption and secret sharing primitives which we discussed in-depth in the summary section of each Chapter. Moving forward, we believe an increased effort into building and implementing more efficient time-based cryptographic schemes is the direction future research should take in this field to be meaningful in light of the current research mentioned above.

References

[1] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *ACM Transactions on Internet Technology*, 5:299–327, 2005.

[2] S. Al-Riyami and K. Paterson. Cbe from cl-pke: A generic construction and efficient schemes. In *Lecture Notes in Computer Science*, volume 3386, pages 398–415. International Workshop on Public Key Cryptography - PKC 2005, Springer, 2005.

[3] S.S Al-Riyami and K.G Paterson. Certificateless public key cryptography. In *Lecture Notes in Computer Science*, volume 2894, pages 452–473. Advances in Cryptology, ASIACRYPT 2003, Springer, 2003.

[4] G. Asharov. Towards characterizing complete fairness in secure two-party computation. In Y. Lindell, editor, *Lecture Notes in Computer Science*, volume 8349, pages 291–316. Theory of Cryptography Conference, TCC 2014, Springer, 2014.

[5] G. Asharov and Y. Lindell. Utility dependence in correct and fair rational secret sharing. In S. Halevi, editor, *Lecture Notes in Computer Science*, volume 5677, pages 559–576. Annual International Cryptology Conference, CRYPTO 2009, Springer, 2009.

[6] G. Ateniese, K. Benson, and S. Hohenberger. Key-private proxy re-encryption. In M. Fischlin, editor, *Lecture Notes in Computer Science*, volume 5473, pages 279–294. Cryptographers' Track at the RSA Conference- CT-RSA 2009, Springer, 2009.

[7] J. Baek, R. Safavi-Naini, and W. Susilo. Certificateless public key encryption without pairing. In *Lecture Notes in Computer Science*, volume 3650, pages 134–148. International conference on information security - ISC 2005, Springer, 2005.

[8] L. Ballard, M. Green, B. De Medeiros, and F. Monrose. Correlation-resistant storage via keyword-searchable encryption. *IACR Cryptol. ePrint Arch.*, 2005:417, 2005.

[9] J. Baron, K. El Defrawy, J. Lampkins, and R. Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In T. Malkin, V. Kolesnikov, A. Lewko, and M. Polychronakis, editors, *Lecture Notes in Computer Science*, volume 9092, pages 23–41. International Conference on Applied Cryptography and Network Security, ACNS 2015, Springer, 2015.

[10] A. Beimel. *Secure schemes for secret sharing and key distribution.* PhD thesis, Technion-Israel Institute of technology, Faculty of computer science, 1996.

[11] A. Beimel. Secret-sharing schemes: A survey. In Y.M. Chee et al., editors, *Lecture Notes in Computer Science*, volume 6639, pages 11–46. International Conference on Coding and Cryptology, Springer, 2011.

[12] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In *Lecture Notes in Computer Science*, volume 2248, pages 566–582. Advances in Cryptology- ASIACRYPT 2001, Springer, 2001.

[13] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Lecture Notes in Computer Science*, volume 1807, pages 259–274. Advances in Cryptology — EUROCRYPT 2000, Springer, 2000.

[14] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73. Association for Computing Machinery, 1993.

[15] F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin. Can a blockchain keep a secret? *IACR Cryptology ePrint Archive*, 2020:464, 2020.

[16] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the AFIPS National Computer Conference, NCC 1979*, volume 48, pages 313–318. International Workshop on Managing Requirements Knowledge (MARK), IEEE, 1979.

[17] M. Blaze and M. Bleumer, G.and Strauss. Divertible protocols and atomic proxy cryptography. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 127–144. Springer, 1998.

[18] D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In J.L. Cachin, C. Camenisch, editor, *Lecture Notes in Computer Science*, volume 3027, pages 223–238. Advances in Cryptology-EUROCRYPT 2004, Springer, 2004.

[19] D. Boneh, S. Eskandarian, S. Kim, and M. Shih. Improving speed and security in updatable encryption schemes. In S. Moriai and H. Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, volume 12493. Lecture Notes in Computer Science, Springer, 2020.

[20] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In J. Kilian, editor, *Lecture Notes in Computer Science*, volume 2139, pages 213–229. Advances in Cryptology- CRYPTO 2001, Springer, 2001.

[21] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan. Key homomorphic prfs and their applications. Cryptology ePrint Archive, Report 2015/220, 2015. https://eprint.iacr.org/2015/220.

[22] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In C. Boyd, editor, *Lecture Notes in Computer Science*, volume 2248, pages 514–532. Advances in Cryptology- ASIACRYPT 2001, Springer, 2001.

[23] D. Boneh and M. Naor. Timed commitments. In *Lecture Notes in Computer Science*, volume 1880, pages 236–254. Advances in Cryptology, CRYPTO 2000, Springer, 2000.

[24] K. Boneh, D.and Lewi, H. Montgomery, and A. Raghunathan. Key homomorphic prfs and their applications. In R. Canetti and Garay J.A., editors, *Lecture Notes in Computer Science*, volume 8042, pages 410–428. Advances in Cryptology, CRYPTO 2013, Springer, 2013.

[25] C. Boyd, Davies. G.T., K. Gjøsteen, and Y. Jiang. Fast and secure updatable encryption†. Technical report, Cryptology ePrint Archive, Report 2019/1457, 2019. https://eprint.iacr.org/2019/1457.pdf, 2020.

[26] J. Camenisch, M. Drijvers, T. Gagliardoni, A. Lehmann, and G. Neven. The wonderful world of global random oracles. In *Lecture Notes in Computer Science*, volume 10820, pages 280–312. Advances in Cryptology - EUROCRYPT 2018, Springer, 2018.

[27] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

[28] R. Canetti, H. Krawczyk, and J.B. Nielsen. Relaxing chosen-ciphertext security. In *Lecture Notes in Computer Science*, volume 2729, pages 565–582. Advances in Cryptology- CRYPTO 2003, Springer, 2003.

[29] J. Cathalo, B. Libert, and J. Quisquater. Efficient and non-interactive timed-release encryption. In S. Qing, W. Mao, J. Lopez, and G. Wang, editors, *Lecture Notes in Computer Science*, volume 3783, pages 291–303. International Conference on Information and Communications Security, ICICS 2005, Springer, 2005.

[30] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In C. Pomerance, editor, *Lecture Notes in Computer Science*, volume 293, pages 11–19. Advances in Cryptology- CRYPTO 1987, Springer, 1988.

[31] H. Chen, S. Fu, and K. Liang. No-directional and backward-leak uni-directional updatable encryption are equivalent. In *Lecture Notes in Computer Science*, volume 13554, pages 387–407. European Symposium on Research in Computer Security - ESORICS 2022, Springer, 2022.

[32] L. Chen, Y. Li, and Q. Tang. Cca updatable encryption against malicious re-encryption attacks. In *Lecture Notes in Computer Science*, volume 12493, pages 590–620. Advances in Cryptology - ASIACRYPT 2020, Springer, 2020.

[33] X. Chen, Y. Liu, Y. Li, and C. Lin. Threshold proxy re-encryption and its application in blockchain. In X. Sun, Z. Pan, and E. Bertino, editors, *Lecture Notes in Computer Science*, volume 11066, pages 16–25. Cloud Computing and Security - ICCCS 2018, Springer, 2018.

[34] V. Cini, S. Ramacher, D. Slamanig, C. Striecks, and E. Tairi. Updatable signatures and message authentication codes. In J.A. Garay, editor, *Lecture Notes in Computer Science*, volume 12710, pages 691–723. Public Key Cryptography, PKC 2021, Springer, 2021.

[35] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, page 364–369. STOC 1986, Association for Computing Machinery, 1986.

[36] PCI Security Standards Council. Data security standard (pci dss v4.0). Technical report, https://www.pcisecuritystandards.org/, 2022.

[37] R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In B. Preneel, editor, *Lecture Notes in Computer Science*, volume 1807, pages 316–334. Advances in Cryptology-EUROCRYPT 2000, Springer, 2000.

[38] R. Cramer, I. Damgård, and J.B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Lecture Notes in Computer Science*, volume 2045, pages 280–300. Advances in Cryptology — EUROCRYPT 2001, Springer, 2001.

[39] A. Davidson, A. Deo, E. Lee, and K. Martin. Strong post-compromise secure proxy re-encryption. In *Australasian Conference on Information Security and Privacy- ACISP 2019*, volume 11547, pages 58–77. Lecture Notes in Computer Science, Springer, 2019.

[40] S. J. De and Asim K. Pal. Achieving correctness in fair rational secret sharing. In M. Abdalla, Cristina N. R., and R. Dahab, editors, *Lecture Notes in Computer Science*, volume 8257, pages 139–161. International Conference on Cryptology and Network Security, CANS 2013, Springer, 2013.

[41] A.W Dent. A survey of certificateless encryption schemes and security models. *International Journal of Information Security*, 7(5):349–377, 2008.

[42] Y. Desmedt, G. Di Crescenzo, and M. Burmester. Multiplicative non-abelian sharing schemes and their application to threshold cryptography. In J. Pieprzyk and R. Safavi-Naini, editors, *Lecture Notes in Computer Science*, volume 917, pages 19–32. Advances in Cryptology, ASIACRYPT 1994, Springer, 1994.

[43] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In J. Feigenbaum, editor, *Lecture Notes in Computer Science*, volume 576, pages 457–469. Advances in Cryptology- CRYPTO 1991, Springer, 1991.

[44] Y. G. Desmedt and Y. Frankel. Homomorphic zero-knowledge threshold schemes over any finite abelian group. *SIAM journal on Discrete Mathematics*, 7(4):667–679, 1994.

[45] Y. Dodis, H. Karthikeyan, and D. Wichs. Updatable public key encryption in the standard model. In *Lecture Notes in Computer Science*, volume 13044, pages 254–285. Theory of Cryptography Conference - TCC 2021, Springer, 2021.

[46] Y. Dodis and T. Rabin. Cryptography and game theory. *Algorithmic Game Theory*, pages 181–207, 2007.

[47] N. Döttling and D. Schröder. Efficient pseudorandom functions via on-the-fly adaptation. In *Lecture Notes in Computer Science*, volume 9215, pages 329–350. Annual Cryptology Conference, Springer, 2015.

[48] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In D. Boneh, editor, *Lecture Notes in Computer Science*, volume 2729, pages 426–444. Advances in Cryptology, CRYPTO 2003, Springer, 2003.

[49] E. Eaton, D. Jao, C. Komlo, and Y. Mokrani. Towards post-quantum key-updatable public-key encryption via supersingular isogenies. In *Lecture Notes in Computer Science*, volume 13203, pages 461–482. Selected Areas in Cryptography: 28th International Conference - SAC 2022, Springer, 2022.

[50] A. Escala and J. Groth. Fine-tuning groth-sahai proofs. In *Lecture Notes in Computer Science*, volume 8383, pages 630–649. Public-Key Cryptography- PKC 2014, Springer, 2014.

[51] D. Evans, V. Kolesnikov, M. Rosulek, et al. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.

[52] A. Everspaugh, K. Paterson, T. Ristenpart, and S. Scott. Key rotation for authenticated encryption. In J. Katz and H. Shacham, editors, *Lecture Note in Computer Science*, volume 10403, pages 98–129. Advances in Cryptology-CRYPTO 2017, 2017.

[53] A. Fabrega, U. Maurer, and M. Mularczyk. A fresh approach to updatable symmetric encryption. *Cryptology ePrint Archive*, 2021.

[54] A. Faonio, D. Hofheinz, and L. Russo. Almost tightly-secure re-randomizable and replayable cca-secure public key encryption. *Cryptology ePrint Archive*, 2023.

[55] A. Festjens, S. Bruyneel, E. Diecidue, and S. Dewitte. Time-based versus money-based decision making under risk: An experimental investigation. *Journal of Economic Psychology*, 50:52–72, 2015.

[56] G. Fuchsbauer, J. Katz, and D. Naccache. Efficient rational secret sharing in standard communication networks. In D. Micciancio, editor, *Lecture Notes in Computer Science*, volume 5978, pages 419–436. Theory of Cryptography Conference, TCC 2010, Springer, 2010.

[57] E. Fujisaki and T. Okamoto. How to enhance the security of public-key encryption at minimum cost. In *Lecture Notes in Computer Science*, volume 1560, pages 53–68. International Workshop on Public Key Cryptography - PKC 1999, Springer, 1999.

[58] S.D. Galbraith, K.G. Paterson, and N.P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[59] Y. J. Galteland and J. Pan. Backward-leak uni-directional updatable encryption from (homomorphic) public key encryption. *Cryptology ePrint Archive*, 2022.

[60] C. Gentry. Certificate-based encryption and the certificate revocation problem. In *Lecture Notes in Computer Science*, volume 2656, pages 272–293. Advances in Cryptology - EUROCRYPT 2003, Springer, 2003.

[61] S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. *Journal of the ACM (JACM)*, 58(6):1–37, 2011.

[62] S. D. Gordon and J. Katz. Rational secret sharing, revisited. In *Lecture Notes in Computer Science*, volume 4116, pages 229–241. International Conference on Security and Cryptography for Networks, SCN 2006, Springer, 2006.

[63] V. Goyal. Reducing trust in the pkg in identity based cryptosystems. In *Lecture Notes in Computer Science*, volume 4622, pages 430–447. Advances in Cryptology: 27th Annual International Cryptology Conference - CRYPTO 2007, Springer, 2007.

[64] V. Goyal, O. Pandey, and B. Sahai, A.and Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, page 89–98. CCS 2006, Association for Computing Machinery, 2006.

[65] J. Groth. Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems. In *Lecture Notes in Computer Science*, volume 2951, pages 152–170. TCC, Springer, 2004.

[66] J. Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *Lecture Notes in Computer Science*, volume 4284, pages 444–459. Advantages in Cryptology- ASIACRYPT 2006, Springer, 2006.

[67] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In N. Smart, editor, *Lecture Notes in Computer Science*, volume 4965, pages 415–432. Advances in Cryptology- EUROCRYPT 2008, Springer, 2008.

[68] Z. Guo, H.and Zhang, J. Zhang, and C. Chen. Towards a secure certificateless proxy re-encryption scheme. In *Lecture Notes in Computer Science*, volume 8209, pages 330–346. International Conference on Provable Security - ProvSec 2013, Springer, 2013.

[69] J. Halpern and V. Teague. Rational secret sharing and multiparty computation: Extended abstract. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, page 623–632. STOC 2004, Association for Computing Machinery, 2004.

[70] L. Harn, C. Lin, and Y. Li. Fair secret reconstruction in (t, n) secret sharing. *Journal of Information Security and Applications*, 23:1–7, 2015.

[71] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Lecture Notes in Computer Science*, volume 963, pages 339–352. Annual International Cryptology Conference, Springer, 1995.

[72] Dennis Hofheinz and Tibor Jager. Tightly secure signatures and public-key encryption. In *Lecture Notes in Computer Science*, volume 7417, pages 590–607. Advances in Cryptology- CRYPTO 2012, Springer, 2012.

[73] S. Hohenberger and B. Waters. Synchronized aggregate signatures from the rsa assumption. In *Lecture Notes in Computer Science*, volume 10821, pages 197–229. Advances in Crytology, EUROCRYPT 2018, Springer, 2018.

[74] S. Jarecki, H. Krawczyk, and J. Resch. Updatable oblivious key management for storage systems. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 379–393. CCS 2019, Association for Computing Machinery, 2019.

[75] Y Jiang. The direction of updatable encryption does not matter much. In *Lecture Notes in Computer Science*, volume 12493, pages 529–558. Advances in Cryptology - ASIACRYPT 2020, Springer, 2020.

[76] J. Katz. Bridging game theory and cryptography: Recent results and future directions. In R. Canetti, editor, *Lecture Notes in Computer Science*, volume 4948, pages 251–272. Theory of Cryptography Conference, TCC 2008, Springer, 2008.

[77] J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[78] J. Katz, J. Loss, and J. Xu. On the security of time-lock puzzles and timed commitments. In *Lecture Notes in Computer Science*, volume 12552, pages 390–413. Theory of Cryptography - TCC 2020, Springer, 2020.

[79] E. Kiltz, J. Pan, and H. Wee. Structure-preserving signatures from standard assumptions, revisited. In *Lecture Notes in Computer Science*, volume 9216, pages 275–295. Advances in Cryptology- CRYPT0 2015, Springer, 2015.

[80] M. Klooß, A. Lehmann, and A. Rupp. (r) cca secure updatable encryption with integrity protection. In Y. Ishai and V. Rijmen, editors, *Lecture Notes in Computer Science*, volume 11476, pages 68–99. Advances in Cryptology-EUROCRYPT 2019, Springer, 2019.

[81] J. Knapp and E. A. Quaglia. Epoch confidentiality in updatable encryption. In *Lecture Notes in Computer Science*, volume 13600, pages 60–67. International Conference on Provable Security - ProvSec 2022, Springer, 2022.

[82] J. Knapp and E. A. Quaglia. Clue: Certificateless updatable encryption. *Italian Conference on Cybersecurity 2023 - ITASEC 2023*, 2023.

[83] J. Knapp and E.A. Quaglia. Fair and sound secret sharing from homomorphic time-lock puzzles. Cryptology ePrint Archive, Report 2020/1078, 2020. https://eprint.iacr.org/2020/1078.

[84] G. Kol and M. Naor. Cryptography and game theory: Designing protocols for exchanging information. In R. Canetti, editor, *Lecture Notes in Computer Science*, volume 4948, pages 320–339. Theory of Cryptography Conference, TCC 2008, Springer, 2008.

[85] G. Kol and M. Naor. Games for exchanging information. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, STOC 2008*, page 423–432. Association for Computing Machinery, 2008.

[86] I. Komargodski and A. Paskin-Cherniavsky. Evolving secret sharing: dynamic thresholds and robustness. In Y. Kalai and L. Reyzin, editors, *Lecture Notes in Computer Science*, volume 10678, pages 379–393. Theory of Cryptography Conference- TCC 2017, Springer, 2017.

[87] H. Krawczyk. Secret sharing made short. In *Lecture Notes in Computer Science*, volume 773, pages 136–146. Advances in Cryptology, CRYPTO 1993, Springer, 1993.

[88] C-S Laih and Y-C Lee. V-fairness (t, n) secret sharing scheme. *IEE Proceedings-Computers and Digital Techniques*, 144(4):245–248, 1997.

[89] E. Lee. Improved security notions for proxy re-encryption to enforce access control. In T. Lange and O. Dunkelman, editors, *Lecture Notes in Computer Science*, volume 11368, pages 66–85. International Conference on Cryptology and Information Security in Latin America- LATINCRYPT 2017, Springer, 2017.

[90] K. Lee. Self-updatable encryption with short public parameters and its extensions. *Designs, Codes and Cryptography*, 79(1):121–161, 2016.

[91] K. Lee, S. G. Choi, D. H. Lee, J. H. Park, and M. Yung. Self-updatable encryption: Time constrained access control with hidden attributes and better efficiency. In K. Sako and P. Sarkar, editors, *Lecture Notes in Computer Science*, volume 8269, pages 235–254. Advances in Cryptology- ASIACRYPT 2013, Springer, 2013.

[92] A. Lehmann and B. Tackmann. Updatable encryption with post-compromise security. In J. Nielsen and V. Rijmen, editors, *Lecture Notes in Computer Science*, volume 10822, pages 685–716. Advances in Cryptology, EUROCRYPT 2018, Springer, 2018.

[93] B. Libert and J. Quisquater. On constructing certificateless cryptosystems from identity based encryption. In *Lecture Notes in Computer Science*, volume 3958, pages 474–490. International Workshop on Public Key Cryptography - PKC 2006, Springer, 2006.

[94] H-Y Lin and L. Harn. Fair reconstruction of a secret. *Information Processing Letters*, 55(1):45–47, 1995.

[95] Y. Liu, Q. Wang, and S. Yiu. Towards practical homomorphic time-lock puzzles: Applicability and verifiability. In *Lecture Notes in Computer Science*, volume 13554, pages 424–443. 27th European Symposium on Research in Computer Security - ESORICS 2022, Springer, 2022.

[96] A. Lysyanskaya and A. Segal. Rational secret sharing with side information in point-to-point networks via time-delayed encryption. *IACR Cryptology ePrint Archive*, 2010:540, 2010.

[97] G. Malavolta and S. A. K. Thyagarajan. Homomorphic time-lock puzzles and applications. In A. Boldyreva and D. Micciancio, editors, *Lecture Notes in Computer Science*, volume 11692, pages 620–649. Annual International Cryptology Conference, CRYPTO 2019, Springer, 2019.

[98] S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song. Churp: dynamic-committee proactive secret sharing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2369–2386. CCS 2019, Association for Computing Machinery, 2019.

[99] T. C. May. Time-release crypto. In *Manuscript*, 1993.

[100] L. Meng and L. Chen. A blockchain-based long-term time-stamping scheme. In *Lecture Notes in Computer Science*, volume 13554, pages 3–24. Computer Security - ESORICS 2022, Springer, 2022.

[101] M. Naor, B. Pinkas, and O. Reingold. Distributed pseudo-random functions and kdcs. In *Lecture Notes in Computer Science*, volume 1592, pages 327–346. Advances in Cryptology - EUROCRYPT 1999, Springer, 1999.

[102] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437. Association for Computing Machinery, ACM, 1990.

[103] J. Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[104] R. Nishimaki. The direction of updatable encryption does matter. In *Lecture Notes in Computer Science*, volume 13178, pages 194–224. Public-Key Cryptography - PKC 2022, Springer, 2022.

[105] K.G. Paterson and E.A. Quaglia. Time-specific encryption. In J.A. Garay and R. De Prisco, editors, *Lecture Notes in Computer Science*, volume 6280, pages 1–16. International Conference on Security and Cryptography for Networks- SCN 2010, Springer, 2010.

[106] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Lecture Notes in Computer Science*, volume 576, pages 129–140. Advances in Cryptology, CRYPTO 1991, Springer, 1991.

[107] D. Pointcheval and O. Sanders. Short randomizable signatures. In *Lecture Notes in Computer Science*, volume 9610, pages 111–126. Topics in Cryptology - CT-RSA 2016, Springer, 2016.

[108] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. *Technical Report MIT/LCS/TR-684*, 1996.

[109] D. Schultz, B. Liskov, and M. Liskov. Mpss: Mobile proactive secret sharing. *ACM Trans. Inf. Syst. Secur.*, 13, 2010.

[110] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[111] A. Shamir. Identity-based cryptosystems and signature schemes. In *Lecture Notes in Computer Science*, volume 196, pages 47–53. Advances in Cryptology - CRYPTO 1984, Springer, 1984.

[112] D. Slamanig and C. Striecks. Puncture'em all: Updatable encryption with no-directional key updates and expiring ciphertexts. *Cryptology ePrint Archive*, 2021.

[113] A. Srinivasan and C.P. Rangan. Certificateless proxy re-encryption without pairing: revisited. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, pages 41–52. SCC 2015, Association for Computing Machinery, 2015.

[114] Y. Sun, W. Susilo, F. Zhang, and A. Fu. Cca-secure revocable identity-based encryption with ciphertext evolution in the cloud. *IEEE Access*, 6:56977–56983, 2018.

[115] S. A. K. Thyagarajan, T. Gong, A. Bhat, A. Kate, and D. Schröder. Opensquare: Decentralized repeated modular squaring service. In *Association for Computing Machinery*, pages 3447–3464. Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security - CCS 2021, ACM, 2021.

[116] Y. Tian, J. Ma, C. Peng, and J. Zhu. Secret sharing scheme with fairness. In *10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 494–500. IEEE, 2011.

[117] M. Tompa and H. Woll. How to share a secret with cheaters. *Journal of Cryptology*, 1(3):133–138, 1989.

[118] O. Uzunkol and M. S. Kiraz. Still wrong use of pairings in cryptography. *Applied Mathematics and Computation*, 333:467–479, 2018.

[119] H. Wang, K.Y. Lam, G.Z Xiao, and H. Zhao. On multiplicative secret sharing schemes. In E.P. Dawson, A. Clark, and C. Boyd, editors, *Lecture Notes on Computer Science*, volume 1841, pages 342–351. Information Security and Privacy, ACISP 2000, Springer, 2000.

[120] Y. Wang, R. Chen, X. Huang, J. Ning, and M. Wang, B.and Yung. Identity-based encryption for fair anonymity applications: Defining, implementing, and applying rerandomizable rcca-secure ibe. *Cryptology ePrint Archive*, 2021.

[121] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In D. Catalano, N. Fazio, R. Gennaro, and Nicolosi A., editors, *Lecture Notes in Computer Science*, volume 6571, pages 53–70. International Workshop on Public Key Cryptography- PKC 2011, Springer, 2011.

[122] L. Xu, X. Wu, and X. Zhang. Cl-pre: A certificateless proxy re-encryption scheme for secure data sharing with public cloud. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, page 87–88. ASIA-CCS 2012, Association for Computing Machinery, 2012.

[123] K. Yang, J. Xu, and Z. Zhang. Certificateless proxy re-encryption without pairings. In *Lecture Notes in Computer Science*, volume 8565, pages 67–88. International Conference on Information Security and Cryptology - ICISC 2013, Springer, 2013.

[124] P. Yang, Z. Cao, and X. Dong. Threshold proxy re-signature. *Journal of Systems Science and Complexity*, 24(4):816–824, 2011.

[125] F. Zhang, R. Safavi-Naini, and W. Susilo. An efficient signature scheme from bilinear pairings and its applications. In *Lecture Notes in Computer Science*, volume 2947, pages 277–290. International Workshop on Public Key Cryptography - PKC 2004, Springer, 2004.

[126] Z. Zhang, D. S. Wong, J. Xu, and D. Feng. Certificateless public-key signature: security model and efficient construction. In *Lecture Notes in Computer Science*, volume 3989, pages 293–308. Applied Cryptography and Network Security- ACNS 2006, Springer, 2006.