Masters Thesis

Secure Key Rotation in Cloud

Pabitra Mandal

This page intentionally left blank.

Secure Key Rotation in Cloud

Thesis submitted to the
Indian Statistical Institute, Kolkata
for partial fulfillment of requirements for the degree

of

Master of Technology in Cryptology & Security

by

Pabitra Mandal
Roll No. CrS2207

Under the guidance of

Capt. Ritesh Wahi
Deputy Director General
Weapons & Electronic Systems Engineering Establishment (WESEE)
Ministry of Defence, New Delhi

Co-supervisor

Lt. Cdr. Keval Krishan
Systems Manager
Weapons & Electronic Systems Engineering Establishment (WESEE)
Ministry of Defence, New Delhi

Dr. Mriganka Mandal
Assistant Professor
Cryptology & Security Research Unit (CSRU)
Indian Statistical Institute, Kolkata

Indian Statistical Institute, Kolkata

July 2024

This page intentionally left blank.

CERTIFICATE

This is to certify that the thesis entitled "**Secure Key Rotation in Cloud**", submitted by **Pabitra Mandal (CrS2207)** to **Indian Statistical Institute, Kolkata**, is a record of bonafide research work under my supervision and guidance, and I consider it worthy of consideration for the award of the degree of *Masters of Technology in Cryptology and Security* of the Institute.

Date:
Place:

Capt. Ritesh Wahi
Deputy Director General and HOD
Weapons and Electronic Systems Engineering
Establishment (WESEE), Ministry of Defense,
New Delhi

Date: 22/07/2024
Place: ISI Kolkata

Mriganka Mandal.

Dr. Mriganka Mandal
Assistant Professor
Cryptology and Security Research Unit
R. C. Bose Centre for Cryptology and Security
Indian Statistical Institute, Kolkata

Acknowledgements

The completion of this thesis would not have been possible without the support of several individuals. First and foremost, I would like to thank my primary supervisor, Capt. Ritesh Wahi, whose expertise, encouragement, and continuous guidance were instrumental in the completion of this work. Your insightful feedback and unwavering support helped shape the direction of this work and brought out the best in me.

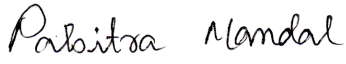
I am also deeply grateful to my co-supervisors, Lt. Cdr. Keval Krishan and Dr. Mriganka Mandal, for their valuable suggestions and for providing me with the resources and environment to conduct my work. Your profound knowledge and constructive criticism greatly enhanced the quality of this thesis.

I want to also express my gratitude to Dr. Pratay Mukherjee and Dr. Avik Chakrobarty, who provided invaluable emotional and mental support throughout this journey. Their encouragement and unwavering belief in me were instrumental in overcoming challenges and staying motivated.

I would also like to acknowledge my colleagues and friends who provided me with much-needed support and encouragement throughout this journey. Your moral support and the many discussions we had were invaluable.

Finally, I am forever indebted to my family for their unwavering love, patience, and understanding. Your constant encouragement and belief in me have been my greatest source of strength.

Thank you all for your invaluable contributions to this work.



Date: 21/07/2024
Place: Kolkata

Pabitra Mandal
Roll No.: CrS2207

Abstract

The way we store and manage data has changed dramatically with the rise of cloud storage. However, keeping our data safe for the long haul requires us to stay vigilant. According to NIST, one important way to do this is by regularly changing the encryption keys that protect our data.

A most naive approach to rotating key is to download all our data from the cloud, switch to a new key (decrypt all the data with the old key and then encrypt with the new key), and then upload everything again. However, for large amounts of data, this can be expensive and time-consuming.

The above-stated problem can be solved by using Updatable Encryption (UE). It's a clever method first introduced by Boneh et al [BLMR13], that makes key changes much easier, especially for cloud storage. With UE, we can send a small update token to the cloud, allowing it to switch our data from the old key to the new one without ever seeing what's inside. This will not only save the bandwidth to download and upload huge amounts of data, but it will also save the computation power required for decryption and encryption.

In this thesis, we'll dive into existing research to see how practical and effective UE is for securing data in the cloud. We hope to show how this innovative approach could make cloud storage safer and more efficient for everyone by looking at what others have discovered and testing things out ourselves.

Keywords: Key Rotation, Updatable Encryption, Cloud Storage, Cloud Storage Security.

Table of Contents

List of Figures

List of Tables

Abbreviations

AE Authenticated Encryption. 1, 6, 17

DEK Data Encryption Key. 9, 10

HSM Hardware Security Modules. 9–11, 15, 16, 19

KEK Key Encryption Key. 9

UAE Updatable Authenticated Encryption. 12

UE Updatable Encryption. v, 2, 13, 18, 19, 22, 24, 27

List of Symbols

$[n]$	Set of all natural numbers upto n.
$\langle \mathbf{s}, \mathbf{v} \rangle$	inner product of two vectors $\mathbf{s}$ and $\mathbf{v}$.
$\mathbb{N}$	The set of all natural numbers.
$\mathbb{R}$	The set of all real numbers.
$\mathbb{Z}$	The set of all integers.
$\mathbb{Z}_n$	Set of all integers modulo n.
$\mathcal{L}(B)$	Lattice generated by the basis B.
$\xleftarrow{R} S$	Sampled randomly from S.
$x \leftarrow \chi$	x is sampled from the distribution χ.
$x \leftarrow \mathcal{U}(S)$	x is sampled uniformly from the set S.

1

Introduction

Cloud computing is a successful paradigm virtually offering unlimited data storage and computational power. It's like renting space on someone else's computer to store and work with your data. Numerous enterprises, government bodies, and individuals are storing their data on the cloud instead of local devices because it's convenient and cheaper than managing your servers. However, public infrastructure has experienced frequent data breaches from the cloud. One potential solution is to enhance data security through the use of cryptographic tools.

When we talk about keeping data safe, encryption is crucial. Encryption is like putting your data in a locked box, and only you have the key to open it. But in the cloud, things are continually changing. Data gets moved around, and we must change the locks frequently to keep it safe. This is where key management comes in. It's like ensuring only the right people have the keys to the locked box.

1.1 Motivation and Background of the Problem

Systems use Authenticated Encryption (AE) techniques that offer strong message secrecy and ciphertext integrity to cryptographically safeguard data during storage. However, if data is kept for a long time, there is still a chance that users' secret keys could eventually be hacked.

Good key management methods require systems to allow periodic key rotation in order to handle this problem. This involves changing the relevant ciphertext and refreshing the secret key used to encrypt the data. Revocation of compromised old keys or data access is another function of key rotation.

Key rotation is possible for a local drive or server by decrypting and re-

encrypting with a new key because symmetric encryption processes are quick, parallelizable, and frequently have plenty of bandwidth. However, bandwidth becomes more expensive than compute when ciphertext storage is outsourced to a cloud storage provider (who may not be reliable). It can be too costly to download, re-encrypt, and upload the complete database at least once, even for little amounts of data. Key rotation through download, decrypt, re-encrypt, and reupload becomes nearly impossible due to this practical concern.

The systems used by Amazon and Google employ an envelope encryption approach, which is very useful but does not support full key rotation and has questionable security consequences in the event of key compromises.

Conversely, symmetric Updatable Encryption (UE) systems (proposed by Boneh et al., CRYPTO 2013 [BLMR13]) provide complete key rotation without necessitating decryption: ciphertexts generated using one key can be switched to another key with the aid of a re-encryption token. It is possible to transmit the token to storage providers without jeopardising security since, by design, even though it depends on both the old and new encryption keys, knowledge of the token should not reveal information about keys or plaintexts.

1.2 Problem Statement

This section outlines the challenges associated with key rotation in cloud storage utilizing envelope encryption [Goo22]. Following this, we delve into the practicalities and existing approaches to key rotation for such a deployment. Finally, by analyzing these existing solutions, we will identify their goals, limitations, and the specific areas we aim to address with our proposed secure key rotation protocol.

1.2.1 Data Model

A `PUT` request is used to store data in a key-value store, while a `GET` request is used to retrieve it in a typical cloud deployment. To distinguish them from cryptographic keys, we refer to key-value pair keys as *index*. Consequently, a `PUT` request that stores v in a database server requires as parameters index i and value v. In contrast, a `GET` request receives an index i and returns the value v that corresponds to it. Using `PUT` and `GET` requests with a plaintext index, respectively, the operations `SecurePut` and `SecureGet` are defined to save and retrieve encrypted data values.

1.2.1.1 Assumptions and Simplifications

Our supposition is a streamlined database backend with just one key-value store. Although we do not expect transactional processing assurances, we do expect the key-value store to lock data items until operations are finished in order to ensure correctness. In order to improve availability and scalability, most database backends replicate and exchange data across several machines. It is left for future work to address the application of our ideas to transactional processing systems and distributed systems.

1.2.2 Threat Model and Goals

To develop a robust solution for secure key rotation in cloud environments, we need to outline our threat model and goals. This ensures that our protocol effectively rotates keys within a cryptoperiod while having minimal impact on normal access services.

1.2.2.1 Threat Model

We consider that a malicious party could discover any old keys and see ciphertexts that have been updated with the most recent keys. Our objective is to offer post-compromise security, meaning that confidentiality cannot be violated even if an opponent obtains out-of-date keys and reads currently encrypted material. This method is most similar to Jarecki et al.'s security model [JKR19], with the exception that ours is stronger and permits data key compromises. It also aligns with the security models of other updatable encryption schemes [BEKS20, BLMR13, BDGJ20, EPRS17, FMM21, GP22, Jia20, KLR19, LT18, MPW22, Nis22][1].

We acknowledge that the suggested approach will differ depending on our security assumptions. Attacks known as denial-of-service (DoS) are not covered by our defences. In the event that an adversary gains access to the database and hinders update processes, our system might not be able to restrict the vulnerability window. Even in the event that the database deviates from the protocol, our goal is to preserve the secrecy of the ciphertexts in order to prevent further information leaking.

Furthermore, we take into account situations in which an adversary compromises the database and then alters or adds ciphertexts. Client computers must be

[1] Jaecki et al.'s security model guarantees that the HSM will not know the data key, but our solutions can also provide the same assurances by applying their techniques.

able to confirm the legitimacy of plaintexts in order to combat this.

1.2.2.2 Security Goal

Our goal is to update all ciphertexts to new keys by the end of the cryptoperiod, given the challenge of giving security assurances when an adversary has access to a ciphertext decryptable by a compromised key (known as trivial victories). By doing this, an adversary's window of opportunity for an attack is narrowed, improving post-compromise security. The frequency of updates made possible by higher key rotation rates reduces the amount of time that the database retains ciphertexts encrypted with compromised keys.

1.2.2.3 Correctness Goal

Another problem arising from concurrent operations is making sure that client updates (`SecurePut`) and ciphertext changes result in ciphertexts that can be successfully retrieved (`SecureGet`). Race conditions resulting from concurrent routine access and ciphertext updating activities may cause incorrect executions. Our goal is to create and demonstrate linearizability guarantees by altering regular access operations and ciphertext updates to remove these race situations. This guarantees a perfectly ordered sequence of events and permits normal access and ciphertext updating activities to run concurrently.

1.2.2.4 Performance Goal

The goal of our protocol is to enable regular access operations in addition to concurrent updates. Routine access operations should not be adversely affected by infrequent key rotations and ciphertext changes. Furthermore, the constant rotation of keys should not cause appreciable latency for ordinary access activities. Our system should be able to rotate big databases and be scalable enough to manage realistic workloads.

2

Preliminaries

2.1 Primitives of Basic Cryptography

We go over some fundamental cryptographic primitives that we use in our work in this part. We parameterize the security of these concepts with respect to advantage functions $\varepsilon : \mathbb{N} \to \mathbb{R}$, which bound the likelihood that an effective opponent will breach the primitive's security, in order to analyse the precise security of our creations.

Definition 2.1.1 (Negligible Function)**.** If for each positive polynomial $\mathtt{poly}(\cdot)$, there exists a positive integer $N_{\mathtt{poly}}$ such that, for all $n > N_{\mathtt{poly}}$,

$$|\mu(n)| < \frac{1}{\mathtt{poly}(n)},$$

then $\mu : \mathbb{N} \to \mathbb{R}$ is *negligible*.

Typically, a *negligible function* is indicated by $\mathtt{negl}(\cdot)$.

2.2 Authenticated Encryption

We revisit the concept of an encrypted method that is authenticated [BN00].

Definition 2.2.1 (Authenticated Encryption [BN00])**.** A pair of effective algorithms $\Pi_{AE} = (\mathtt{KeyGen}, \mathtt{Encrypt}, \mathtt{Decrypt})$ with the following syntax constitutes an authenticated encryption strategy for a message space $\mathcal{M}$.

- $\mathtt{KeyGen}(1^\lambda) \to k$: The key-generation algorithm returns a key k upon receiving a security parameter λ.

- $\texttt{Encrypt}(k, m) \rightarrow C$: The encryption method returns a ciphertext C on input of a key k and a message $m \in \mathcal{M}$.
- $\texttt{Decrypt}(k, C) \rightarrow m/\perp$: The decryption method returns a message m or $\perp$ upon receiving a key k and a ciphertext C as inputs.

Definition 2.2.2 (Correctness)**.** For every $\lambda \in \mathbb{N}$ and $m \in \mathcal{M}$, we claim that a AE scheme $\Pi_{AE} = (\texttt{KeyGen, Encrypt, Decrypt})$ is right if we have

$$\Pr[\texttt{Decrypt}(k, \texttt{Encrypt}(k, m)) = m] = 1,$$

where $k \leftarrow \texttt{KeyGen}(1^\lambda)$.

Definition 2.2.3 (Confidentiality)**.** For $\Pi_{AE} = (\texttt{KeyGen, Encrypt, Decrypt})$, let $\mathcal{M}$ be a message space. We state that $\varepsilon_{ae}^{\texttt{conf}}$-*confidentiality* is satisfied by Π_{AE}, if

$$\left| \Pr[\mathcal{A}^{O_{k,0}(\cdot,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{O_{k,1}(\cdot,\cdot)}(1^\lambda) = 1] \right| = \varepsilon_{ae}^{\texttt{conf}(\lambda)},$$

holds for all efficient adversaries $\mathcal{A}$. Where $k \leftarrow \texttt{KeyGen}(1^\lambda)$, and the definition of the oracle $O_{k,b}$ for $b \in \{0, 1\}$ is as follows:

- $O_{k,b}(m^{(0)}, m^{(1)})$: The oracle computes $C \leftarrow \texttt{Encrypt}(k, m^{(b)})$ and returns C on input two messages $m^{(0)}, m^{(1)} \in \mathcal{M}$.

In particular, if $\varepsilon_{ae}^{\texttt{conf}}(\lambda) = \texttt{negl}(\lambda)$, then Π_{AE} satisfies confidentiality.

Definition 2.2.4 (Integrity)**.** For a message space $\mathcal{M}$, let $\Pi_{AE} = (\texttt{KeyGen, Encrypt, Decrypt})$ be a AE scheme. We state that $\varepsilon_{ae}^{\texttt{int}}$-*integrity* is satisfied by Π_{AE} if for all efficient adversaries $\mathcal{A}$

$$\Pr[\mathcal{A}^{O_k(\cdot)}(1^\lambda) = C \wedge \texttt{Decrypt}(k, C) \neq \perp \wedge \notin \texttt{Table}] = \varepsilon_{ae}^{\texttt{int}}(\lambda),$$

holds. Where $k \xleftarrow{R} \texttt{KeyGen}(1^\lambda)$, and the definition of oracle O_k and $\texttt{Table}$ are as follows:

- $O_k(m) : C \leftarrow \texttt{Encrypt}(k, m)$ and $\texttt{Table} = \texttt{Table} \cup \{C\}$ are the computations made by the oracle upon receiving a message $m \in \mathcal{M}$. The oracle then returns C.

If $\varepsilon_{ae}^{\texttt{int}}(\lambda) = \texttt{negl}(\lambda)$, then Π_{AE} satisfies integrity.

Definition 2.2.5 (Ciphertext Pseudo-randomness)**.** For a message space $\mathcal{M}$, let $\Pi_{AE} = (\texttt{KeyGen, Encrypt, Decrypt})$ be a AE scheme. $\varepsilon_{ae}^{\texttt{rand}}$-*ciphertext pseudo-randomness* is said to be satisfied by Π_{AE} if

$$\left| \Pr[\mathcal{A}^{O_{k,0}(\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{O_{k,1}(\cdot)}(1^\lambda) = 1] \right| = \varepsilon_{ae}^{\texttt{rand}}(\lambda),$$

holds for all efficient adversaries $\mathcal{A}$. Where $k \leftarrow \texttt{KeyGen}(1^\lambda)$ and the definitions of the oracles $O_{k,b}$ for $b \in \{0,1\}$ and are as follows:

- $O_{k,0}(m)$: The oracle computes $C \leftarrow \texttt{Encrypt}(k,m)$ and returns C upon receiving a message $m \in \mathcal{M}$.
- $O_{k,1}(m)$: The oracle computes $C \leftarrow \texttt{Encrypt}(k,m)$ upon receiving a message $m \in \mathcal{M}$ and returns $C' \xleftarrow{R} \{0,1\}^{|C|}$.

If $\varepsilon_{ae}^{\texttt{rand}}(\lambda) = \texttt{negl}(\lambda)$, then Π_{AE} satisfies ciphertext pseudo-randomness.

2.3 ElGamal Cryptosystem

Using the ideas of Diffie-Hellman key exchange, the ElGamal cryptographic method is an asymmetric encryption algorithm. It was created in 1985 by Taher ElGamal and is widely used in digital signatures, encryption, and secure data transmission applications.

Construction 2.3.1. Assume that the system parameters $\mathbb{G}, g, p$ are known as CRS and that q is a λ-bit prime. This means that the DDH problem is hard with respect to λ. Here is a description of the scheme.

- $\texttt{KeyGen}(1^\lambda) : x \xleftarrow{R} \mathbb{Z}_q^*, y \leftarrow g^x$, return $(sk, pk) = (x, y)$
- $\texttt{Encrypt}(pk, m) : r \xleftarrow{R} \mathbb{Z}_q$, return $C \xleftarrow{R} (g^r, y^r m)$.
- $\texttt{Decrypt}(sk, C)$: parse $C = (C_1, C_2)$, return $m \leftarrow C_2 \cdot (C_1^x)^{-1}$.

3

Literature Survey

3.1 Envelope Encryption

Cloud databases may safely store sensitive data by encrypting it with cryptographic keys. Government and industry regulations [Bar20, PCI] mandate that these keys be kept and controlled inside cryptographic modules like Hardware Security Modules (HSM) to prevent adversaries from accessing them. The highest FIPS 140-2 security level is one of the worldwide physical security standards that HSMs follow [Nat19].

3.1.1 Challenges with HSMs

Compared to conventional machines, HSMs have limits when it comes to handling huge data sizes and scaling to high quantities. Encrypting many megabyte-sized messages on HSMs creates a major bottleneck due to its computational intensity. The PCI-DSS [PCI] and numerous cloud service providers, including Google [Goo22], Microsoft [Inc22], IBM [IBM22], and Amazon [AWS18], advise employing envelope encryption as a solution to this problem.

3.1.2 Envelope Encryption

With envelope encryption, messages in plaintext are first encrypted using a data key to create ciphertext, then the data key is subsequently encrypted using a root key to create a wrap[1]. Envelope encryption is made up of the wrap and ciphertext, and it can be safely kept in cloud databases. Because data keys are used to encrypt

[1]Key encryption keys (KEK) and data keys (DEK) are other names for data keys and root keys, respectively.

plaintext data, normal machines handle this task, mitigating the HSM bottleneck. Instead of multiple huge plaintexts, HSMs just need to encrypt small data keys using root keys. The demand on HSMs is dramatically reduced because data keys and wraps are much less (less than 100 bytes) than usual plaintext data (kilobytes to megabytes). Additionally, this method guarantees that HSMs do not need to keep a large number of unique data keys, as they have limited key storage capacity.

3.1.3 Operational Architecture

The typical envelope encryption setup involves three key components:

1. **Client Machine:** Responsible for generating DEKs, encrypting data, and interfacing with the HSM to manage wraps.
2. **Hardware Security Module (HSM):** Ensures secure storage and management of the root key. Handles the wrapping and unwrapping of DEKs.
3. **Database Server:** Stores the encrypted data (ciphertext) and the wraps.

3.1.4 Operational Process

`SecurePut` and `SecureGet` are standard operations used for storing and retrieving envelope-encrypted data.

`SecurePut` Operation

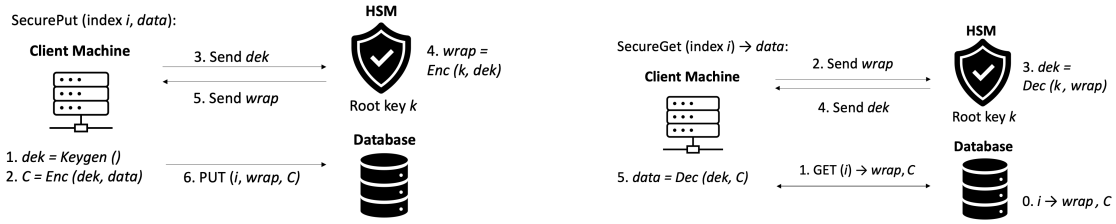
To perform a `SecurePut` operation for storing a plaintext data value at index i (Figure 3.1b), the following steps are executed:

- A new data key (`dek`) is generated by the client computer.
- The `dek` is used to encrypt the plaintext data into a ciphertext.
- The `dek` is encrypted using the root key by the client communicating with the HSM, producing a `wrap`.
- The ciphertext and `wrap` are subsequently kept in the database.

`SecureGet` Operation

To perform a `SecureGet` operation for retrieving data at index i (Figure 3.1b), the following steps are executed in reverse:

- The client retrieves the wrap and ciphertext from the database.
- In order to retrieve the `dek`, the client asks the HSM to decrypt the `wrap` using the root key.
- The ciphertext is then decrypted using the `dek`, revealing the original plaintext.



(a) `SecurePut` (b) `SecureGet`

Figure 3.1: `SecurePut` and `SecureGet` Operations

3.1.5 Advantages of Envelope Encryption

- **Reduced HSM Workload:** By encrypting only small-sized DEKs rather than large volumes of data, the HSM's computational load is significantly reduced. This addresses the scalability and performance limitations of HSMs.
- **Enhanced Security:** Storing the root key within an HSM ensures that even if adversaries access the cloud database, they cannot decrypt the data without the root key. Client-side encryption adds an extra layer of security, ensuring that even if the database server is compromised, the data remains secure as the server does not have access to the DEKs.
- **Compliance with Standards:** HSMs are certified to meet stringent security standards, such as FIPS 140-2, which guarantees a high level of physical security and integrity for key management.

By leveraging envelope encryption, organizations can effectively mitigate the risks associated with data breaches and ensure compliance with regulatory standards, all while maintaining efficient performance and scalability in.

3.2 Updatable Encryption Syntax

Definition 3.2.1 (Updatable Authenticated Encryption)**.** For a message space $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda\in\mathbb{N}}$ *updatable authenticated encryption* (UAE) scheme is a collection of efficient algorithms $\Pi_{\texttt{UAE}} = \texttt{KeyGen, Encrypt, Decrypt, ReKeyGen, ReEncrypt}$ using the syntax as follows:

- $\texttt{KeyGen}(1^\lambda) \to k$: The secret key k is produced via the key generation method given a security parameter λ.
- $\texttt{Encrypt}(k, m) \to (\hat{C}, C)$: A ciphertext header $\hat{C}$ and a ciphertext body C are produced by the encryption algorithm upon receipt of a key k and a message $m \in \mathcal{M}_\lambda$.
- $\texttt{ReKeyGen}(k_1, k_2, \hat{C}) \to \Delta_{1,2,\hat{C}}/\ \perp$: The re-encryption key generation technique produces an update token $\Delta_{1,2,\hat{C}}$ or returns $\perp$ given two keys k_1, k_2 and a ciphertext header $\hat{C}$.
- $\texttt{ReEncrypt}(\Delta, (\hat{C}, C)) \to (\hat{C}', C')$: The re-encryption algorithm either returns $\perp$ or generates a new ciphertext $(\hat{C}', C)$ upon receiving an update token Δ and a ciphertext $(\hat{C}, C)$ as input.
- $\texttt{Decrypt}(k, (\hat{C}, C)) \to m/\ \perp$: A message m or $\perp$ is returned by the decryption algorithm upon receiving a key k and a ciphertext $(\hat{C}, C)$.

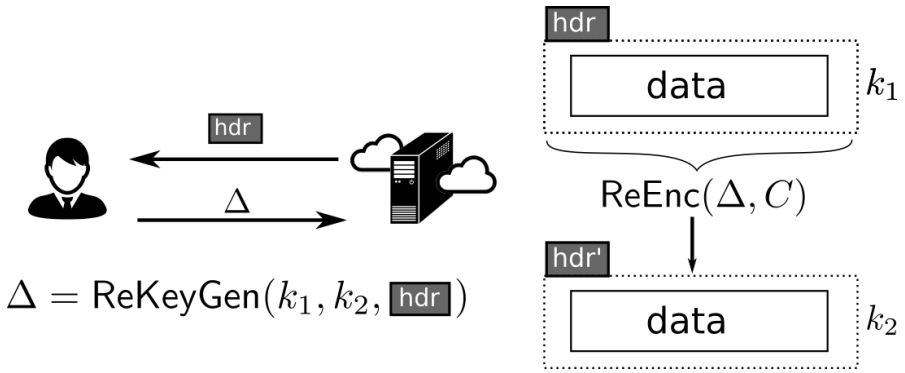


Figure 3.2: Ciphertext-dependent Updatable Encryption

3.2.1 Ciphertext-dependence

As previously mentioned, in order to produce a rekey token, updatable AE schemes require the ciphertext header $\hat{C}$. Schemes with $\hat{C}$ empty, or ϵ, are also taken into

account. We concentrate on methods for which $\hat{C} = \epsilon$ or a non-empty string is consistently produced by encryption. Schemes classified as *ciphertext-dependent* are those classified as *ciphertext-independent*. $\hat{C}$ is left out of notation for ciphertext-independent schemes; for example, C for the ciphertext instead of $(\hat{C}, C)$ and $\Delta_{i,j}$ instead of $\Delta_{i,j,\hat{C}}$.

Definition 3.2.2 (Compactness)**.** For a message space $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda\in\mathbb{N}}$, we state that an UE scheme $\Pi_{\texttt{UAE}} = (\texttt{KeyGen, Encrypt, ReKeyGen, ReEncrypt, Decrypt})$ is *compact* if there exist polynomials $f_1(\cdot)$ and $f_2(\cdot)$ such that the lengths of the ciphertext header and token satisfy:

$$|\hat{C}| \leq f_1(\lambda), \quad |\Delta_{1,2,\hat{C}}| \leq f_2(\lambda),$$

for any $\lambda \in \mathbb{N}$ and message $m \in \mathcal{M}_\lambda$, where $k_1, k_2 \leftarrow \texttt{KeyGen}(1^\lambda), (\hat{C}, C) \leftarrow \texttt{Encrypt}(k_1, m)$, and $\Delta_{1,2,\hat{C}} \leftarrow \texttt{ReKeyGen}(k_1, k_2, \hat{C})$. Therefore, the message length has no bearing on the length of the update token or the ciphertext header.

The correctness condition for a UE scheme is defined in a standard manner.

Definition 3.2.3 (Correctness)**.** For a message space $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda\in\mathbb{N}}$, we state that an UE scheme $\Pi_{\texttt{UAE}} = (\texttt{KeyGen, Encrypt, ReKeyGen, ReEncrypt, Decrypt})$ is *correct* if for any $\lambda \in \mathbb{N}$, $N \in \mathbb{N}$, and $m \in \mathcal{M}_\lambda$, the following holds:

$$Pr[\texttt{Decrypt}(k_N, (\hat{C}_N, C_N)) = m] = 1,$$

where $k_1, \cdots, k_N \leftarrow \texttt{KeyGen}(1^\lambda), (\hat{C}_1, C_1) \leftarrow \texttt{Encrypt}(k_1, m)$, and

$$(\hat{C}_{i+1}, C_{i+1}) \leftarrow \texttt{ReEncrypt}(\texttt{ReKeyGen}(k_i, k_{i+1}, \hat{C}_i), (\hat{C}_i, C_i)).$$

4

Key Rotation Protocol

4.1 Key Rotation in Practice

Large amounts of unencrypted data can occur in real-world situations. Updating a database containing all of the encrypted content (ciphertexts) directly becomes computationally costly and can result in large expenses. Google states, for example, that it stores data in encrypted chunks that are typically between 256 KB and 8 MB in size [Goo20]. As a result, root key rotations and associated key wrap updates are frequently given priority by existing cloud storage providers [AWS18, Goo22, IBM22, Inc22], whereas data key rotations and ciphertext updates are completely ignored. This strategy is based on the fact that updating small key wraps has a far lower overhead than re-encrypting large volumes of data, which can be rather expensive.

4.1.1 Update Process

Here, we list the steps that make up the updating process:

- The HSM generates a new root key.
- The database contains individual `wrap`s.
- The old root key is used to decrypt each recovered `wrap`, and the new root key is used to encrypt it.
- Lastly, the old `wrap`s are uploaded back into the database to be replaced by the freshly created ones.

4.1.2 Security Consideration

However, this strategy of omitting data key rotations has a significant weakness: the data keys are fixed, resulting in a lack of post-compromise security. In such a scenario, compromised root keys or data keys would still render the ciphertexts vulnerable. Therefore, our goal is to establish a secure key rotation protocol that rotates both the data key and root key, thereby prompting the complete update of ciphertexts and limiting the size of the vulnerability window.

4.2 Naive Ciphertext Updates

It's simple to use this naive method of rotating both the root and data keys: the client downloads the previous ciphertexts, re-encrypts, and uploads the current ciphertext.

4.2.1 Update Process

In figure 4.1 the steps to rotate keys in this approach are described as follows:

- The wrap $\texttt{wrap}_1$ and ciphertext C_1 are retrieved by the update machine from the database together with the root key ID $\texttt{keyID}_1$ and accompanying version number v_1.
- The update machine generates a new data key $\texttt{dek}_2$.
- To decrypt $\texttt{wrap}_1$ into the previous data key $\texttt{dek}_1$, the update machine talks with the HSM.
- The update machine then encrypts $\texttt{dek}_2$ into the new wrap $\texttt{wrap}_2$.
- The update machine decrypts C_1 into message data m.
- The update machine encrypts m with $\texttt{dek}_2$ into ciphertext C_2.
- Finally, the new wrap $\texttt{wrap}_2$ and ciphertext C_2 are automatically `PUT` in the database if the version number v_1 of this ciphertext has not changed and not been deleted. Otherwise, the ciphertext update is aborted. (Step 13 shows v_1 renamed to v_2 to disambiguate the old version number from a possibly modified version number.)

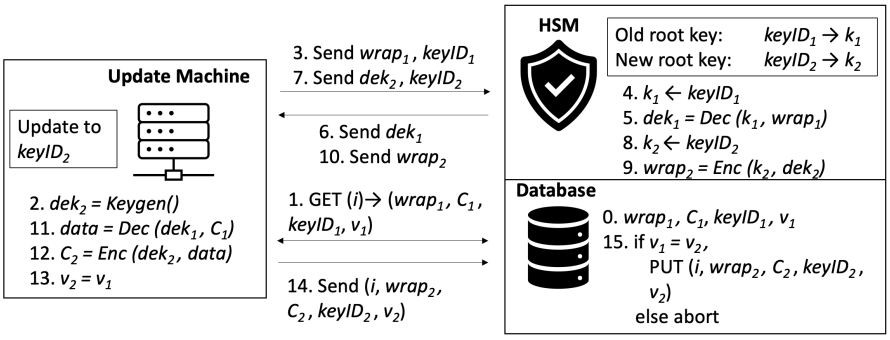


Figure 4.1: Naive Ciphertext Updates

4.2.2 Security Advantages

Naive ciphertext updates offer several security benefits over current practices, which typically only update wraps:

- **Post-compromise Security:** A whole new root and data key pair are used to encrypt the updated ciphertext. Therefore, compromising earlier keys gives an adversary no more ability to decipher the plaintext data.
- **Integrity:** Furthermore, it uses a AE, like `AES-GCM`, to ensure integrity.

4.2.3 Operational Benefits

- It does not introduce additional latency by utilising a new encryption scheme because it retains standard encryption schemes for `SecurePut` and `SecureGet` routine access operations.
- Because they don't necessitate major changes to the source code of the client machine, they are also simple to incorporate into current practice.

4.2.4 Performance Considerations

This approach's main problem is that it might have a detrimental effect on how well normal access procedures function. The update procedure uses a significant amount of the database's network capacity to download and upload multiple big ciphertexts. Concurrent regular access activities of `SecurePut` and `SecureGet` share this bandwidth. As a result, network capacity may end up being the primary bottleneck, limiting how frequently simple updates and regular access procedures may be carried out concurrently.

4.3 An Updatable Encryption Approach

This section outlines an UE approach designed to minimize network overheads associated with updating ciphertexts. The strategy makes use of the finding that switching from old to new data keys requires the greatest amount of resources when it comes to upgrading envelope-encrypted ciphertexts. The protocol operates as follows:

4.3.1 Root Key Rotation

- Root keys are rotated naively due to the minimal cost of downloading and uploading a key wrap.
- Data is encrypted and data keys are rotated using an UE scheme.

4.3.2 Generalization and Efficiency

- The approach is designed to accommodate any type of updatable encryption scheme, whether it relies on ciphertext-dependency or is ciphertext-independent.
- Rather than transmitting the entire ciphertext, it employs an update token that facilitates the database in transitioning from an old data key to a new one.
- To streamline operations and minimize network latency, the protocol integrates the update token with a new wrap, which is necessary for updating regardless.
- For ciphertext-dependent schemes that necessitate retrieving a compact header $\hat{C}$, the method combines this header with the existing wrap.

4.3.3 Detailed Process

By replacing the plaintext data encryption in `SecurePut` and `SecureGet` with an updatable encryption (UE) scheme, this technique improves the efficiency of ciphertext updates. Figure 4.2 depicts the process of updating ciphertexts, where an entry at index i is transitioned to a new root key identified by $\texttt{keyID}_2$ using UE.

- For index i, the update machine retrieves the wrap $wrap_1$, version number v_1, root key ID $\texttt{keyID}_1$, and ciphertext header C_{header} (for ciphertext-dependent schemes) from the database and creates a new data key $\texttt{dek}_2$.
- The update machine interacts with the HSM and uses $\texttt{keyID}_1$ to retrieve $\texttt{dek}_1$ by decrypting $\texttt{wrap}_1$ and uses $\texttt{keyID}_2$ to encrypt $\texttt{dek}_2$ to a new wrap $\texttt{wrap}_2$.
- An update token $\Delta = \texttt{UE.TokenGen}(\texttt{dek}_1, \texttt{dek}_2, C_{header})$ is generated.
- Lastly, the update machine transmits the version number v_1 (renamed to v_2), index i, token Δ, new wrap $\texttt{wrap}_2$, new root key ID $\texttt{keyID}_2$, and new wrap $\texttt{wrap}_2$ to the database.
- The database generates the new ciphertext $C_2 = \texttt{UE.Update}(\Delta, C_1)$ and atomically updates the database with the new ciphertext C_2, new wrap $\texttt{wrap}_2$, and new root key ID $\texttt{keyID}_2$ if the version number v_1 has not changed.

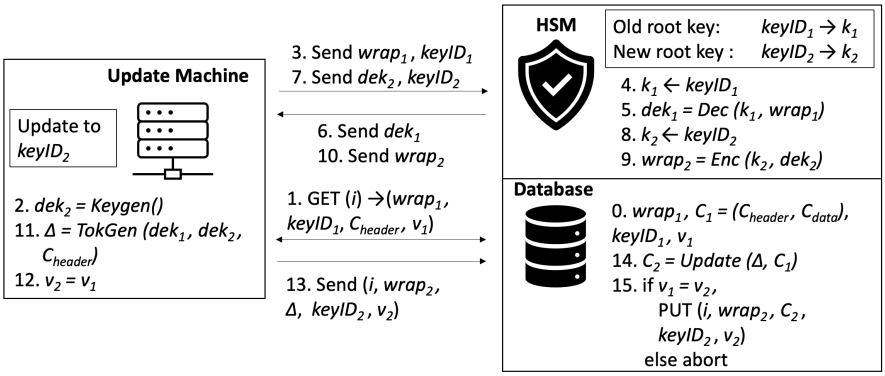


Figure 4.2: An Updatable Encryption Approach

4.3.4 Security and Performance

- This method provides security and integrity guarantees post-compromise, as UE schemes are designed to offer both, assuming certain conditions for post-compromise security are met.
- The method reduces the update bandwidth overhead significantly, from the full ciphertext size to a few bytes.
- The method reduces the update bandwidth overhead significantly, from the full ciphertext size to a few bytes.This technique lessens the trade-off between the size of the vulnerability window and normal access performance by eliminating bandwidth as a significant constraint.

4.3.5 Concurrency Impact

- Even with its benefits, using updatable encryption directly can still have a detrimental effect on how well regular access operations function.
- Since many databases synchronise operations, obtaining exclusive locks during the computationally demanding ciphertext update step can prevent concurrent `GET` and `PUT` operations. The item Prolonged update processes have the potential to monopolise the single thread of operation execution, preventing subsequent standard access processes from starting until the update is finished. The item Thus, increasing the frequency of updates may have a negative impact on how well regular access operations operate.

5

Our Work

5.1 Updatable Encryption using ElGamal (RISE)

RISE, introduced by Lehmann et al. [LT18], adapts the traditional proxy re-encryption concept of ElGamal, initially presented by Blaze et al. [BBS98], for a secret-key environment. This method on its own does not ensure complete security since some parts of the ciphertext remain unaltered. To mitigate this issue, the scheme takes advantage of ElGamal's ability to re-randomize ciphertexts using only the public key. By incorporating the epoch's public key element into the token, and re-randomizing the ciphertext during updates, the scheme introduces probabilistic updates. This is a significant distinction from the work of Everspaugh et al. [EPRS17].

Although employing public-key techniques in a secret-key updatable encryption setting might seem redundant, previous constructions also depend on key-homomorphic PRFs, which are based on similar techniques. In contrast, directly utilizing the group structure without intermediary abstractions allows for re-randomization and achieves complete IND-UPD security.

Regarding computational efficiency, this scheme's encryption process is as efficient as those by Boneh et al. [BLMR13] and Everspaugh et al.'s ReCrypt scheme [EPRS17]. Additionally, the operations for generating update tokens and updating ciphertexts are more efficient than those in [EPRS17], thanks to the ciphertext-independent setting.

Construction 5.1.1 (RISE)**.** Let $\mathbb{G}, g, p$ represent the system parameters available as CRS, ensuring the difficulty of the DDH problem relative to λ, where q is a prime number of λ bits. The scheme is outlined as follows.

- $\texttt{KeyGen}(e) : k_e \xleftarrow{R} \mathbb{Z}_q^*$
- $\texttt{Encrypt}(k_e, m) : y = g^{k_e}, r \xleftarrow{R} \mathbb{Z}_q$, return $C_e \xleftarrow{R} (y^r, g^r m)$.
- $\texttt{Decrypt}(k_e, C_e)$: parse $C_e = (C_1, C_2)$, return $m \leftarrow C_2 \cdot C_1^{-1/k_e}$.
- $\texttt{ReKeyGen}(k_e, k_{e+1}) : \Delta_{e+1} \xleftarrow{R} (k_{e+1}/k_e, g^{k_{e+1}})$, return Δ_{e+1}.
- $\texttt{ReEncrypt}(\Delta_{e+1}, C_e)$: parse $\Delta_{e+1} = (\Delta, y')$ and $C_e = (C_1, C_2)$, $r' \xleftarrow{R} \mathbb{Z}_q$, $C_1' \xleftarrow{R} C_1^{\Delta} \cdot y'^{r'}, C_2' \xleftarrow{R} C_2 \cdot g^{r'}$, return $C_{e+1} \leftarrow (C_1', C_2')$.

A strongly secure (IND-ENC+IND-UPD) UE has been demonstrated by Lehmann et al. for their updatable RISE encryption scheme. Additionally, under its encryption algorithm Encrypt and decryption algorithm Decrypt, RISE is homomorphic. In particular, given two messages m and m', $\texttt{Encrypt}(k_e, m) = (C_1, C_2)$ and $\texttt{Encrypt}(k_e, m') = (C_1', C_2')$ are their corresponding RISE ciphertexts. Then, $\texttt{Encrypt}(k_e, m) \cdot \texttt{Encrypt}(k_e, m') = (C_1 \cdot C_1', C_2 \cdot C_2')$ is the result of computing their product.

$$\texttt{Decrypt}(\texttt{Encrypt}(k_e, m) \cdot \texttt{Encrypt}(k_e, m')) = m \cdot m'$$

is satisfied by the decryption algorithm of RISE.

5.1.1 Security Consideration

While RISE has been shown to be secure under the DDH assumption, IND-ENC and IND-UPD are not [LT18], there is a security lack for our desired thread model. Note that the update token is generated as $\Delta_{e+1} = (k_{e+1}/k_e, g^{k_{e+1}})$. Now suppose the adversary learns the outdated key k_e, then he can easily retrieve the new key by observing the update token as $k_{e+1} = \Delta \cdot k_e$, where $\Delta = k_{e+1}/k_e$.

Thus the knowledge of an outdated key makes the whole system vulnerable. To avoid this shortcoming we are proposing two improved versions of RISE.

Construction 5.1.2 (RISE+)**.** Let $\mathbb{G}, g, p$ represent the system parameters available as CRS, ensuring the difficulty of the DDH problem relative to λ, where q is a prime number of λ bits. The scheme is outlined as follows.

- $\texttt{KeyGen}(e) : k_e \xleftarrow{R} \mathbb{Z}_q^*$
- $\texttt{Encrypt}(k_e, m) : y = g^{k_e}, r \xleftarrow{R} \mathbb{Z}_q$, return $C_e \xleftarrow{R} (y^r, g^r m)$.
- $\texttt{Decrypt}(k_e, C_e)$: parse $C_e = (C_1, C_2)$, return $m \leftarrow C_2 \cdot C_1^{-1/k_e}$.

- $\texttt{ReKeyGen}(k_e, k_{e+1}, C_1) : \Delta_{e+1} \xleftarrow{R} (C_1^{k_{e+1}/k_e}, g^{k_{e+1}})$, return Δ_{e+1}.
- $\texttt{ReEncrypt}(\Delta_{e+1}, C_e)$: parse $\Delta_{e+1} = (\Delta, y')$ and $C_e = (C_1, C_2)$, $r' \xleftarrow{R} \mathbb{Z}_q$, $C_1' \xleftarrow{R} \Delta \cdot y'^{r'}, C_2' \xleftarrow{R} C_2 \cdot g^{r'}$, return $C_{e+1} \leftarrow (C_1', C_2')$.

Note that now the scheme becomes *ciphertext-dependent*. So the client has to generate an update token for each ciphertext, affecting the system's performance and routine access.

Currently, the randomness for re-encryption is generated on the server side. We can generate the randomness for re-encryption on the client side as well. This thing done in the following construction:

Construction 5.1.3 (RISE++)**.** Let $\mathbb{G}, g, p$ represent the system parameters available as CRS, ensuring the difficulty of the DDH problem relative to λ, where q is a prime number of λ bits. The scheme is outlined as follows.

- $\texttt{KeyGen}(e) : k_e \xleftarrow{R} \mathbb{Z}_q^*$
- $\texttt{Encrypt}(k_e, m) : y = g^{k_e}, r \xleftarrow{R} \mathbb{Z}_q$, return $C_e \xleftarrow{R} (y^r, g^r m)$.
- $\texttt{Decrypt}(k_e, C_e)$: parse $C_e = (C_1, C_2)$, return $m \leftarrow C_2 \cdot C_1^{-1/k_e}$.
- $\texttt{ReKeyGen}(k_e, k_{e+1}, C_1) : r' \xleftarrow{R} \mathbb{Z}_q, y' = g^{k_{e+1}}, \Delta_{e+1} \xleftarrow{R} (g^{r'}, C_1^{k_{e+1}/k_e} \cdot y'^{r'})$, return Δ_{e+1}.
- $\texttt{ReEncrypt}(\Delta_{e+1}, C_e)$: parse $\Delta_{e+1} = (\Delta, C_1')$ and $C_e = (C_1, C_2), C_2' \xleftarrow{R} C_2 \cdot \Delta$, return $C_{e+1} \leftarrow (C_1', C_2')$.

5.1.2 Security Analysis

Both the proposed schemes are not only based on RISE but also have the almost same structure as RISE, except for the ciphertext dependency. So we can believe that these schemes are also secure till the original RISE scheme is secure. Besides that now even if the adversary learns the outdated keys, he can not recover the new key from the update token.

Note that if the adversary gets access to the old ciphertext and corresponding key, he can trivially know the underlying plaintext and it is a case of trivial win. But the good news is that he can not retrieve the updated key, so the newly generated ciphertexts are safe from the adversary.

5.2 Proposed Schemes based on ElGamal

We have noticed that in RISE the confidentiality for the plaintext depends only on the randomness introduced during encryption or re-encryption. The key is not directly used to hide the message, it is only used to retrieve the message. But in the original ElGamal algorithm, the key has a direct role in hiding the message. So we are now going to construct UE purely based on ElGamal Encryption.

Construction 5.2.1. Let $\mathbb{G}, g, p$ represent the system parameters available as CRS, ensuring the difficulty of the DDH problem relative to λ, where q is a prime number of λ bits. The scheme is outlined as follows.

- $\texttt{KeyGen}(e) : k_e \xleftarrow{R} \mathbb{Z}_q^*$
- $\texttt{Encrypt}(k_e, m) : y = g^{k_e}, r \xleftarrow{R} \mathbb{Z}_q$, return $C_e \xleftarrow{R} (g^r, y^r m)$.
- $\texttt{Decrypt}(k_e, C_e)$: parse $C_e = (C_1, C_2)$, return $m \leftarrow C_2 \cdot C_1^{-k_e}$.
- $\texttt{ReKeyGen}(k_e, k_{e+1}) : \Delta_{e+1} \xleftarrow{R} (k_{e+1} - k_e, g^{k_{e+1}})$, return Δ_{e+1}.
- $\texttt{ReEncrypt}(\Delta_{e+1}, C_e)$: parse $\Delta_{e+1} = (\Delta, y')$ and $C_e = (C_1, C_2)$, $r' \xleftarrow{R} \mathbb{Z}_q$, $C_1' \xleftarrow{R} C_1 \cdot g^{r'}, C_2' \xleftarrow{R} C_2 \cdot C_1^{\Delta} \cdot y'^{r'}$, return $C_{e+1} \leftarrow (C_1', C_2')$.

In our experimental implementation, this scheme is being used currently.

5.2.1 Security Consideration

As like the original RISE, this scheme also suffers from a security notion for our threat model. An adversary with the knowledge of outdated key and update token can retrieve the new key by calculating $k_{e+1} = \Delta + k_e$ and completely break the system.

To mitigate such things we can improve the scheme as follows:

Construction 5.2.2. Let $\mathbb{G}, g, p$ represent the system parameters available as CRS, ensuring the difficulty of the DDH problem relative to λ, where q is a prime number of λ bits. The scheme is outlined as follows.

- $\texttt{KeyGen}(e) : k_e \xleftarrow{R} \mathbb{Z}_q^*$
- $\texttt{Encrypt}(k_e, m) : y = g^{k_e}, r \xleftarrow{R} \mathbb{Z}_q$, return $C_e \xleftarrow{R} (g^r, y^r m)$.
- $\texttt{Decrypt}(k_e, C_e)$: parse $C_e = (C_1, C_2)$, return $m \leftarrow C_2 \cdot C_1^{-k_e}$.

- $\texttt{ReKeyGen}(k_e, k_{e+1}, C_1) : y' = g^{k_{e+1}}, \Delta_{e+1} \xleftarrow{R} (C_1^{k_{e+1}-k_e}, y')$, return Δ_{e+1}.
- $\texttt{ReEncrypt}(\Delta_{e+1}, C_e)$: parse $\Delta_{e+1} = (\Delta, y')$ and $C_e = (C_1, C_2), r' \xleftarrow{R} \mathbb{Z}_q, C_1' = C_1 \cdot g^{r'} C_2' \xleftarrow{R} C_2 \cdot \Delta \cdot y'^{r'}$, return $C_{e+1} \leftarrow (C_1', C_2')$.

In this scheme, the randomness generation is done on the server side. To generate the randomness in client side we can go through the following:

Construction 5.2.3. Let $\mathbb{G}, g, p$ represent the system parameters available as CRS, ensuring the difficulty of the DDH problem relative to λ, where q is a prime number of λ bits. The scheme is outlined as follows.

- $\texttt{KeyGen}(e) : k_e \xleftarrow{R} \mathbb{Z}_q^*$
- $\texttt{Encrypt}(k_e, m) : y = g^{k_e}, r \xleftarrow{R} \mathbb{Z}_q$, return $C_e \xleftarrow{R} (g^r, y^r m)$.
- $\texttt{Decrypt}(k_e, C_e)$: parse $C_e = (C_1, C_2)$, return $m \leftarrow C_2 \cdot C_1^{-k_e}$.
- $\texttt{ReKeyGen}(k_e, k_{e+1}, C_1) : r' \xleftarrow{R} \mathbb{Z}_q, y' = g^{k_{e+1}}, \Delta_{e+1} \xleftarrow{R} (C_1^{-k_e} \cdot y'^{r'}, g^{r'})$, return Δ_{e+1}.
- $\texttt{ReEncrypt}(\Delta_{e+1}, C_e)$: parse $\Delta_{e+1} = (\Delta, C_1')$ and $C_e = (C_1, C_2), C_2' \xleftarrow{R} C_2 \cdot \Delta$, return $C_{e+1} \leftarrow (C_1', C_2')$.

5.2.2 Security Notions

We have not analyzed the security of the proposed scheme till now. But as it is based on ElGamal and Lehmann et al. proved their ElGamal based scheme `RISE` to be secure, we can expect that it will also be secure under the DDH hardness assumption. But till it is not proven we are not claiming anything.

6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we have explored the concept of Secure Key Rotation in Cloud environments with a focus on Updatable Encryption (UE). The primary motivation was to address the inefficiencies and security concerns associated with traditional key rotation methods.

Our findings indicate that updatable encryption offers a promising solution for secure and efficient key rotation. Specifically, it reduces the need for extensive data transfer and re-encryption, thereby minimizing operational overhead and enhancing security by not exposing data during the key rotation process. We implemented an UE scheme based on ElGamal, demonstrating its practicality and effectiveness in real-world cloud environments.

6.2 Future Work

While our research has laid a strong foundation for secure key rotation using updatable encryption, several avenues for future work remain:

- **Enhanced Security Analysis**: Conduct a more exhaustive security analysis under various attack models to further validate the robustness of the UE scheme.
- **Performance Optimization**: Explore optimization techniques to further improve the efficiency of the UE process, particularly in large-scale cloud environments.

- **Implementation in Diverse Cryptographic Schemes**: Investigate the application of UE in other cryptographic schemes.
- **Real-World Deployment**: Work towards deploying the scheme in a real-world cloud infrastructure to evaluate its performance and security in a production environment.
- **User Experience and Usability Studies**: Assess the usability and impact on end-users, focusing on ease of integration and management within existing cloud security frameworks.
- **Adaptation to Emerging Technologies**: Consider the adaptation of UE schemes to emerging technologies such as quantum computing, ensuring future proof security solutions.
- **Mapping to Elliptic Curve Group**: Using elliptic curves such as `NIST P-256` or `P-521`, we can instantiate the groups.

Bibliography

[AWS18] Inc. Amazon Web Services. Aws key management service cryptographic details, 2018. https://d1.awsstatic.com/whitepapers/KMS-Cryptographic-Details.pdf.

[Bar20] Elaine Barker. Recommendation for key management, part 1: General, 2020. https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-57pt1r5.pdf.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, pages 127–144, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[BDGJ20] Colin Boyd, Gareth T. Davies, Kristian Gjøsteen, and Yao Jiang. Fast and secure updatable encryption. In *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part I*, page 464–493, Berlin, Heidelberg, 2020. Springer-Verlag.

[BEKS20] Dan Boneh, Saba Eskandarian, Sam Kim, and Maurice Shih. Improving speed and security in updatable encryption schemes. Cryptology ePrint Archive, Paper 2020/222, 2020. https://eprint.iacr.org/2020/222.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *Proc. of Crypto*, volume 8043 of *LNCS*, pages 410–428, 2013. https://crypto.stanford.edu/~dabo/pubs/papers/homprf.pdf.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. Cryptology ePrint Archive, Paper 2000/025, 2000. https://eprint.iacr.org/2000/025.

[EPRS17] Adam Everspaugh, Kenneth Paterson, Thomas Ristenpart, and Sam Scott. Key rotation for authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 98–129, Cham, 2017. Springer International Publishing.

[FMM21] Andrés Fabrega, Ueli Maurer, and Marta Mularczyk. A fresh approach to updatable symmetric encryption. Cryptology ePrint Archive, Paper 2021/559, 2021. https://eprint.iacr.org/2021/559.

[Goo20] Google Cloud. Encryption at rest in google cloud, 2020. https://services.google.com/fh/files/misc/security_whitepapers_march2018.pdf.

[Goo22] Google Cloud. Envelope encryption, 2022. https://cloud.google.com/kms/docs/envelope-encryption.

[GP22] Yao Jiang Galteland and Jiaxin Pan. Backward-leak uni-directional updatable encryption from (homomorphic) public key encryption. Cryptology ePrint Archive, Paper 2022/324, 2022. https://eprint.iacr.org/2022/324.

[IBM22] IBM Cloud. Key protect, 2022. https://cloud.ibm.com/catalog/services/key-protect.

[Inc22] Microsoft Inc. Best practices for secrets management in key vault, 2022. https://docs.microsoft.com/en-us/azure/key-vault/secrets/secrets-best-practices.

[Jia20] Yao Jiang. The direction of updatable encryption does not matter much. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 529–558, Cham, 2020. Springer International Publishing.

[JKR19] Stanislaw Jarecki, Hugo Krawczyk, and Jason Resch. Updatable oblivious key management for storage systems. Cryptology ePrint Archive, Paper 2019/1275, 2019. https://eprint.iacr.org/2019/1275.

[KLR19] Michael Klooß, Anja Lehmann, and Andy Rupp. (r)cca secure updatable encryption with integrity protection. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 68–99, Cham, 2019. Springer International Publishing.

[LT18] Anja Lehmann and Björn Tackmann. Updatable encryption with post-compromise security. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 685–716, Cham, 2018. Springer International Publishing.

[MPW22] Peihan Miao, Sikhar Patranabis, and Gaven Watson. Unidirectional updatable encryption and proxy re-encryption from ddh. Cryptology ePrint Archive, Paper 2022/311, 2022. https://eprint.iacr.org/2022/311.

[Nat19] National Institute of Standards and Technology. FIPS Security Requirements for Cryptographic Modules, 2019. Special Publication (SP) 800-140.

[Nis22] Ryo Nishimaki. The direction of updatable encryption does matter. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography – PKC 2022*, pages 194–224, Cham, 2022. Springer International Publishing.

[PCI] PCI Security Standards Council 2022. Payment Card Industry (PCI) Data Security Standard. v4.0.