Low Trust Cloud Data Storage and Sharing

JONAS THURESSON
MARTIN ÖRNDAHL
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Low Trust Cloud Data Storage and Sharing

Jonas Thuresson, dat13jth@student.lu.se
Martin Örndahl, dic13mor@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisors:
Viktor Andersson, Axis Communications AB
Paul Stankovski, Lund University

Examiner: Thomas Johansson, Lund University

June 25, 2018

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

The many benefits of storing data in the cloud have driven more and more parties to outsource their storage needs. But, as data leaves the owner's domain it is exposed to the risk of unauthorised access, because now, it is the cloud service provider who is responsible for access control.

This thesis studies applicable techniques that allows the owner to apply encryption to the data, without losing distribution possibilities when it is uploaded to cloud services. Techniques that ensure that the cloud service provider can never see the plaintext data and thus decrease the risk of it being exposed. Two ways of achieving this cryptographically is proxy re-encryption (PRE) and attribute-based encryption (ABE). These two types of asymmetric primitives can generate a one-to-many relationship between public and private keys. This allows several parties to decrypt data protected under the same public key. However, both approaches suffer from drawbacks related to key revocation and in some cases performance.

Our research is mainly focused on overcoming these two issues. We also want to evaluate whether encryption and distribution are two properties that can be achieved in contexts with high performance requirements. By modifying, implementing and evaluating PRE and ABE in combination with symmetric encryption we show that both techniques can obtain both encryption and distribution with high performance. The issue with key revocation is using time stamps which make keys invalid beyond a certain time. Consequently, this means that the key revocation process is not perfect as keys are not revoked instantly.

Keywords: proxy re-encryption, attribute-based encryption, secure cloud storage, key revocation, file sharing

Acknowledgements

First and foremost we would like to thank Axis Communications AB for the opportunity to write this thesis. They provided us with resources needed and a great workplace. Special thanks to our supervisor at Axis, Viktor Andersson and manager Guillermo Sagastume for supporting our work and providing technical guidance. Finally, we would also like to thank our supervisor at LTH, Paul Stankovski for his valuable academical and technical support needed to complete this thesis.

Popular science summary

Most of us use cloud services to store and share our personal files. But have you ever reflected over whether it is you or the cloud service provider that is in control of your data? The only real protection you have that nothing malicious is done with your files, is your trust in the provider. That they manage your data the way they are supposed to. You might be okay with this, after all, it is just some personal photos and your favourite recipes. But what happens when the data in question is, for example, highly sensitive surveillance footage?

During the last decade, our way of storing data material has changed vastly. We have gone from storing data locally on our phone and computer, to having everything backed up in the cloud. The same goes for corporations, it is much easier to store data in the cloud than having to build your own infrastructure for the same purpose. However, storing data outside of the own domain is not without risks. Once data is uploaded in the cloud, it is the cloud service provider's responsibility to keep it safe and provide access to authorised parties. This means that the service provider is in full control of reading and editing all data within its domain. For individuals, this may be acceptable, but for corporations with highly sensitive data, this is out of the question.

The consequences of data leakage can be enormous for corporations. In 2013, Yahoo was breached and information about 3 billion users was exposed. The breach knocked 350 million dollars off the selling price when Yahoo was bought by Verizon. This is just one of many examples when poor data management has led to sensitive data being exposed. So, how do companies that outsource their storage ensure that their data at rest is safe and shareable?

Proxy re-encryption (PRE) and *attribute-based encryption* (ABE) are two types of cryptographic techniques that can be used overcome this problem. By deploying any of these two techniques it is possible to encrypt data before it is uploaded to the cloud, this while it is still possible to share *without* having to disclose the private key. However, not without consequences; both schemes suffer from drawback of being slow. Thus, if it should be feasible to deploy on large amounts of data, the process must be improved. Another difficulty that arises when we look at sharing is the possibility of revoking ac-

cess that has previously been granted. In this thesis we have investigated how we can benefit from the characteristics of these schemes, while still overcoming their drawbacks.

Table of Contents

List of Figures

List of Tables

Abbreviations

- **ABE** - Attribute-Based Encryption
- **CCA** - Chosen Ciphertext Attack
- **CCA2** - Adaptive Chosen Ciphertext Attack
- **CP** - Ciphertext-Policy
- **CPA** - Chosen Plaintext Attack
- **DLP** - Discrete Logarithm Problem
- **ECDLP** - Elliptic Curve Discrete Logarithm Problem
- **eDBDH** - Extended Decisional Bilinear Diffie-Hellman
- **IND** - Indistinguishability
- **KP** - Key-Policy
- **OTS** - One-Time Signature
- **PRE** - Proxy Re-Encryption
- **RCCA** - Replayable Chosen Ciphertext Attack

Chapter 1
Introduction

With the many advantages that cloud services provide, it has become common for companies to outsource their storage needs to cloud services. The cloud providers are responsible for safeguarding and keeping data accessible for its clients. However, while these services ease the access and distribution of data they also come with various security concerns. As soon as data is uploaded to a cloud storage, its owner loses control. It is now up to the service provider to ensure that no unauthorised access is made. This is a very undesirable situation as customers with high security requirements might not trust the provider or is not legally allowed to. Consequently, the provider may miss out on potential customers and clients may have to choose a more complex solution. A simple, but not very practical approach, would be to encrypt all data before uploading it. However, this makes data impossible to share without disclosing the private key.

1.1 Background

As demand for cloud services has grown sharply in recent years, the concern regarding security have gained increased focus. Clients have grown more aware of the fact that they put their data at risk when outsourcing their storage needs. Hence, it would be beneficial if data could be encrypted before it leaves the owner's domain. However, the properties of cryptographically secure and distributable oppose each other if conventional symmetric or asymmetric encryption is applied. This as the private key must be disclosed for another party to decrypt data.

A common approach adapted in many of today's widely known cloud storages is local encryption. The customer to such a service uploads its data in plaintext, thereafter the service provider encrypts each data block with a randomly generated symmetric data encryption key (DEK). A cluster of DEKs are then encrypted under a key encryption key (KEK). The DEKs are often stored physically close to the data while the KEKs are stored in a key management service. These processes and systems are under the control of the service provider, meaning that the provider has full control to decrypt and read the data stored within its system.

To ensure that only the owner can grant decryption privileges to its data, a more sophisticated solution is needed. A solution that overcomes the issue with distribution of encrypted data and safe key management. It is the ambition of this thesis to adapt and apply available protocols to requirements set in a cloud

service. We will also assess if there is a possible solution that fulfils the desired properties.

1.2 Project aims

How can access control be deployed on data located outside the owner's domain? What are the limitations and consequences of introducing possible solutions and how can they be solved or mitigated? The main goal of this master thesis is to investigate cryptographic schemes and their practical use to secure the transport, storage and distribution of end-to-end encrypted data. It also aims on providing solutions for overcoming/mitigating issues of deployed schemes and finally, evaluate to what degree the schemes are fit for use.

The questions this report aims to answer are the following:

Question one: In a practical way, can data be protected so that it cannot be read by the storage provider, this while it is possible for the owner to grant access to other parties?

Question two: Is it possible to not only grant access to data to other parties, but also efficiently revoke it in practice?

Question three: Can operations as encryption, (re-encryption,) and decryption be performed efficiently enough on large data volumes without introducing significant time delays?

1.3 Method

The *first step* consisted of researching fields of interest and related work. Initially, the use-case of which this thesis was focusing on had to be defined. The use-case require a one-to-many relationship between public and private key. Thus, the project was initiated with a research study to identify techniques that fulfilled this requirement. Once the research was completed a few primitives within each area of interest was selected. The primitives in focus was chosen firstly on their characteristics, and secondly on their level of recognition as sound primitives in academia.

The *second step* was to analyse and mitigate/solve issues related to the cryptographic schemes of interest. This was done by first gaining in depth knowledge of each primitive and its limitations. The insights from the analysis of the schemes were applied in a high-level design in which solutions to overcome found issues were introduced.

In the *third step* the schemes were implemented and modified to introduce solutions earlier presented in the high-level design. This was first done in prototypes in which each primitive was implemented (if not already existing) with the presented solutions. The prototypes were then run to collect initial information of how they performed. These prototypes would then work as the base for the end product in which a data stream was encrypted from end-to-end. The end product would ultimately work as a confirmation of a successful deployment of the chosen primitives.

1.4 Related work

In 1998, Blaze, Bleumer and Strauss introduced what they called atomic proxy cryptography [5], a solution where an untrusted intermediary with a proxy key can transform a ciphertext encrypted under a party's public key to be decrypted by another party's private key. In this solution the intermediary learns nothing about the private keys of the two parties, nor sees the underlying plaintext of the ciphertext. At a first glance, this seems to be an excellent approach to solve this thesis's posed problem. On the other hand, the protocol contained some serious security flaws and undesired features. For instance, the protocol was not collusion resistant, meaning that if the intermediary colluded with one of the parties, they could extract the other party's private key. Also, the protocol allowed ciphertexts to be transformed in both directions with the same proxy key, a feature known as bidirectionality. However, the basic idea of transforming ciphertext so it can be decrypted under another key pair than the pair it originally was encrypted under has huge potential.

Since the introduction of atomic proxy cryptography, a new category of cryptography has emerged, proxy re-encryption (PRE). PRE are trying to make use of the basic idea of Blaze et al. while improving features, and security of the keys involved in it.

The features of PRE schemes were loosely defined until 2003 when Anca and Yevgeniy wrote a paper [2] in which they more formally defined the properties of it. Among other things, they determined that it was possible to divide PRE into two categories; unidirectional and bidirectional schemes.

In 2006, an important paper [3] was presented by Ateniese, Fu, Green and Hohenberger. It was a PRE scheme which its authors had achieved to be unidirectional and collusion safe. Furthermore, the scheme had several other desirable properties in addition to the two mentioned. Now a more useful scheme existed that secured the master key under the discrete logarithm assumption. Unfortunately, the scheme designed by Ateniese et al. was inadequate in one aspect; it was not CCA-secure.

The first unidirectional and replayable chosen ciphertext attack secure PRE primitive, secure in the standard model, was presented in 2011, when Libert and Vergnaud included the use of one-time signatures into their scheme. Now they had overcome the weakness in the work of Ateniese et al. although the operations needed are mathematically more complex.

A second type of cryptographic system that is useful in similar contexts as PRE is so called attribute-based encryption (ABE). It was first introduced by Sahai and Waters in [19] as an extension of identity based encryption. Sahai et al. made an abstraction of identities and instead let these be defined by descriptive attributes. The attributes would then be used to describe decryption rights in keys and ciphertexts. Only if the attributes in key and ciphertext matched to a specified point, then a ciphertext could be successfully decrypted. Bethencourt, Sahai and Waters introduced the first ciphertext-policy ABE primitive in [4]. They made use of "AND" and "OR" gates as they constructed access policies of attributes, an access policy would then be associated with a ciphertext. A key on the other hand, was associated with a set of attributes. Decryption would then only be

successful if the set of attributes in the key passed through the access policy in the ciphertext. This resulted in a stronger sense of access control than the original ABE scheme.

Access policies can either be related with ciphertexts or keys, depending on which a primitive of ABE can be said to belong to ciphertext-policy respectively key-policy.

1.5 Thesis outline

Chapter 2 (Theory) explains the basic theory needed to understand the cryptographic primitives deployed.

Chapter 3 (System requirements) describes the system environment and use cases that this thesis has in focus.

Chapter 4 (System analysis of applicable techniques) evaluates whether interesting cryptosystems are applicable in our system and use cases. The discussions in this chapter results in the decisions of which primitives that should be further investigated. Also, issues related to deploying certain cryptosystems are analysed.

Chapter 5 (Design) describes solutions to the identified issues of each type of cryptosystem. These solutions work as a base for the implementation step.

Chapter 6 (Implementation) presents the underlying construction of the chosen cryptographic primitives, key expiration property, and optimisation. In this stage a proof of concept is constructed, showing that the designed solutions are theoretically and practically applicable.

Chapter 7 (Performance results) presents the results obtained during benchmarking of the implemented and discussed cryptographic primitives.

Chapter 8 (Discussion) contains a discussion about the benchmarking results. The applied cryptosystems are also discussed and analysed regarding their performance.

Chapter 9 (Conclusions) Presents the conclusions of this master thesis project.

Chapter 2
Theory

In this chapter we describe the theory needed to understand the work done in this thesis. We assume that the reader has some basic knowledge about cryptography, like symmetric and asymmetric encryption. We will nevertheless begin by explaining asymmetric encryption because it is such a central concept in this thesis.

2.1 Asymmetric cryptography

Asymmetric cryptography is a cryptosystem where, in contrast to symmetric cryptography, a user possesses a pair of keys, instead of just one. A key-pair consists of one public key and one private key. The private key is, as the name suggests, kept private, just as a symmetric key. The public key, on the other hand, is actively published so that anyone can use it. The keys are mathematically linked so that performing an operation using one of the keys, the other key will invert that operation. Figure 2.1 shows a typical case where Alice sends an encrypted message to Bob. Alice and Bob do not have a pre-shared key as in the case when using a symmetric cryptosystem. Instead, Bob publishes his public key pk_{Bob}, which Alice uses to encrypt the message m and produce the ciphertext c. She then sends the ciphertext to Bob who uses his private key sk_{Bob} together with the decryption algorithm to decrypt c and recover the plaintext message m.

The advantage here is that Alice (or anyone else) can send encrypted messages to Bob, without having a shared key beforehand.

2.2 Hybrid cryptography

Hybrid cryptography is the technique of combining both symmetric and asymmetric cryptography to be able to make use of the advantages they offer while minimising their shortcomings.

A good of example of where hybrid cryptography is used in practice is in the transport layer security protocol (TLS) [11]. In TLS, an asymmetric algorithm is used to exchange a symmetric key securely between two communicating parties. After they both have agreed on a key to use, they can continue to communicate securely using symmetric encryption and decryption of the data they transmit and receive.

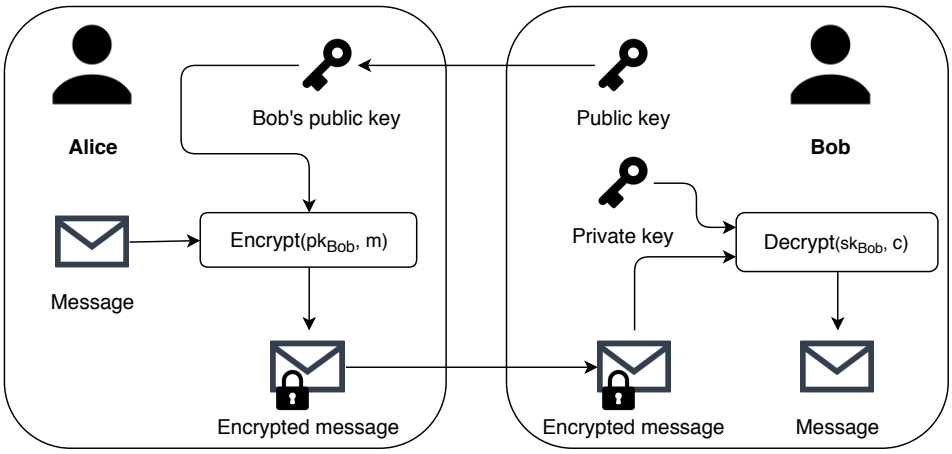

Figure 2.1: Alice using asymmetric encryption to send a message securely to Bob.

2.3 Proxy re-encryption

Proxy re-encryption (PRE) is a technique where it is possible to take a message encrypted under one asymmetric key-pair and encrypt that message again, using a so-called re-encryption key, in such a way that the re-encrypted message becomes encrypted under another key-pair [3]. An example can be seen in Figure 2.2. Here, Alice encrypts a message to Bob using asymmetric encryption as described in Section 2.1. Alice sends the encrypted message to the proxy, instead of Bob, and Bob instead delegates the decryption to Charles. Using his own private key sk_{Bob} and Charles's public key $pk_{Charles}$, Bob produces a re-encryption key $rk_{Bob \to Charles}$ that can re-encrypt ciphertexts that was encrypted with Bobs public key to another ciphertext that is encrypted under Charles's public key. Bob provides the re-encryption key to the proxy. When Charles asks the proxy server for the message, the proxy uses the ciphertext c and the re-encryption key $rk_{Bob \to Charles}$ to produce the ciphertext c', which is sent to Charles. Charles uses his own private key $sk_{Charles}$ to decrypt the ciphertext and recover the plaintext message m.

2.3.1 Directionality

Anca and Yevgeniy [2] define the two types of categories that PRE schemes can be divided into. They can either be bidirectional or unidirectional.

In a bidirectional scheme, re-encryption keys can be used to re-encrypt encrypted messages both ways. So, if there is a re-encryption key that can re-encrypt messages from Alice to Bob, that means that the same key can also be used to re-encrypt messages from Bob to Alice.

In a unidirectional scheme, re-encryption keys can only be used to re-encrypt encrypted messages one way, say from Alice to Bob, but not from Bob to Alice.

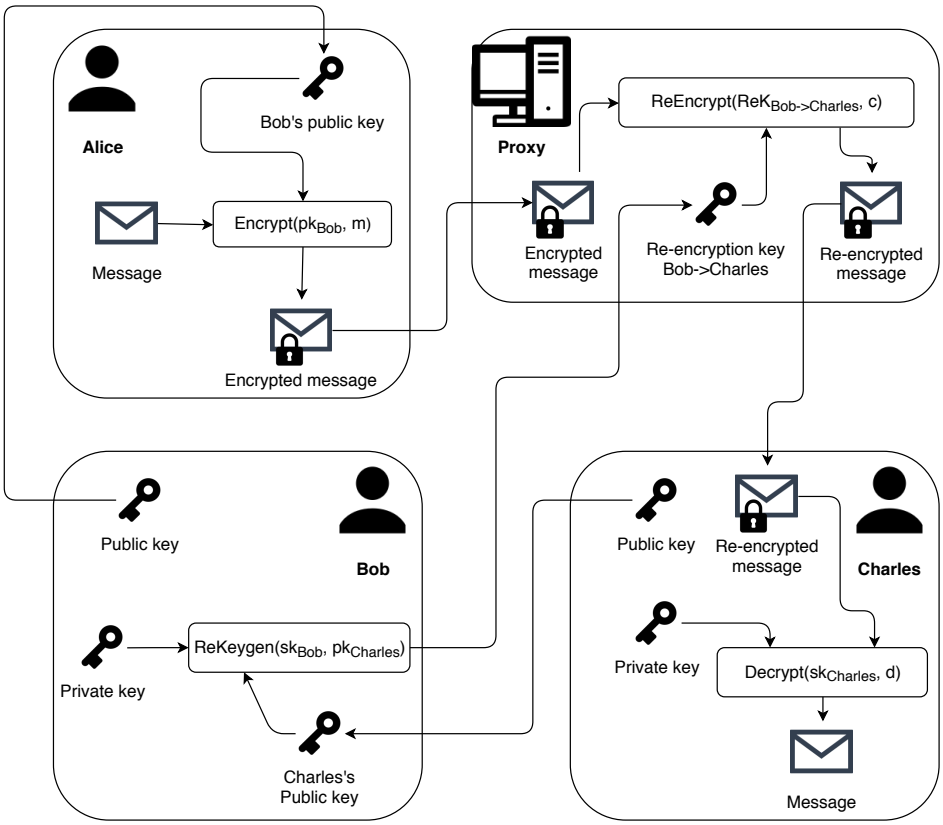

Figure 2.2: Bob delegating decryption of a message, originally intended for himself, to Charles via PRE.

2.3.2 Single-hop and multi-hop

The notions of single-hop and multi-hop property of PRE schemes refer to how many times an encrypted message can be re-encrypted. In a multi-hop scheme, given a set of re-encryption keys $rk_{i\to i+1}, rk_{i+1\to i+2}$ ... and a set of parties $i, i+1, i+2, \ldots$, a ciphertext can be re-encrypted several times. First from i to $i+1$, then from $i+1$ to $i+2$, and so on. In a single-hop scheme, only one re-encryption is possible. This means that when a ciphertext is re-encrypted from i to $i+1$, the resulting ciphertext cannot be further re-encrypted from $i+1$ to $i+2$, even if the re-encryption key $rk_{i+1\to i+2}$ exists.

2.3.3 Collusion resistance

Collusion resistance refers to inability to extract the private key of the delegator in the case where the proxy and a delegatee collude with each other. In particular, it should not be feasible to combine a re-encryption key from A to B and the private

key of B to recover the private key of A:

$$rk_{A\to B} + sk_B \not\to sk_A$$

2.3.4 Transitivity

A scheme is transitive if it is possible for the proxy to re-delegate decryption capability without involvement of the delegator. Practically, this means that if the proxy can combine two or more re-encryption keys, and by this achieve a new delegation, then the scheme is transitive. Transitivity can be described as:

$$rk_{A\to B} + rk_{B\to C} \to rk_{A\to C}$$

Transitivity is often an unwanted property since the delegator cannot be ensured that parties, other than the originally intended ones, are not given decryption privileges.

2.3.5 Interactivity

If the private key sk_B of the delegatee is needed in the generation of the re-encryption key $rk_{A\to B}$, then the scheme is interactive. This implies that the delegatee must trust the party performing the key generation with its private key. It also, however, gives the delegatee the possibility to either accept or reject delegations, since it is active participation is required to create the re-encryption key.

2.3.6 Algorithms in unidirectional single-hop proxy PRE

A unidirectional single-hop PRE scheme consists of a tuple of algorithms (Setup, Keygen, ReKeygen, Encrypt_1, Encrypt_2, ReEncrypt, Decrypt_1, Decrypt_2) [14]. They have the following properties:

- $Setup(\lambda) \to params$: The setup algorithm takes a security parameter λ and produces a tuple of global parameters $params$. These parameters are publicly available to everybody in the system.
- $Keygen(params) \to (pk_A,\ sk_A)$: The key generation algorithm produces a key-pair for a party A.
- $ReKeygen(params,\ sk_A,\ pk_B) \to rk_{A\to B}$: The re-encryption key generation takes a private key belonging to a party A, sk_A, and a public key belonging to a party B, pk_B, and produces a re-encryption key $rk_{A\to B}$
- $Encrypt_1(params,\ pk_A,\ m) \to c'$: The level 1 encryption algorithm encrypts the message m under the public key pk_A and produces a ciphertext, that can only be decrypted by the party A.
- $Encrypt_2(params,\ pk_A,\ m) \to c$: The level 2 encryption algorithm encrypts the message m under the public key pk_A and produces a ciphertext, that can be re-encrypted to the party B, or decrypted by the party A.

- $ReEncrypt(params,\ rk_{A\to B},\ c) \to c'$: The re-encryption operation re-encrypts a level 2 ciphertext c encrypted under the public key pk_A, into a level 1 ciphertext c' encrypted under the public key pk_B.
- $Decrypt_1(params,\ sk_B,\ c') \to m$: The level 1 decryption operation decrypts a ciphertext c with the private key sk_B into plaintext m.
- $Decrypt_2(params,\ sk_A,\ c') \to m$: The level 2 decryption function decrypts a level 2 ciphertext with the private key sk_A into the plaintext m.

2.4 Attribute-based encryption

Attribute-based encryption (ABE) is a more advanced cryptosystem than the conventional version of asymmetric encryption. Here, private keys and ciphertexts are associated with access policies and attributes. Access policies can either be associated with ciphertexts, called ciphertext-policy ABE (CP-ABE) or with private keys, called key-policy ABE (KP-ABE).

In ABE, access control is performed by matching attributes between a private key and the ciphertext. If the attributes in the private key overlap with the attributes in the ciphertext, decryption is successful. In this type of cryptosystem an authority generates a public key and a master key. The authority can now generate private keys and associate them with either an access policy or an attribute set, depending on the type of ABE used. These keys are then distributed to their intended parties. Once a message is encrypted, either an access policy or an attribute set, depending on the type ABE, is associated with the ciphertext. Once a party decrypts a ciphertext, it will only be successful if his private key fulfils the requirements of the ciphertext.

2.4.1 Access policy location

The access policy can be in either the private key or in the ciphertext.

Key policy

In key-policy attribute-based encryption (KP-ABE), the access policy is placed in the private key generated by the authority. An attribute set is placed in the ciphertext.

Ciphertext policy

In ciphertext-policy attribute-based encryption (CP-ABE), the access policy is placed in the ciphertext (the opposite of KP-ABE) and an attribute set is placed in the private key.

2.4.2 Access structures

Let P be a set of attributes and Γ_P a set of subsets of P ($\Gamma_P \subseteq 2^P$), in other words, Γ_P is an access structure for P. Let C be a qualified subset of Γ_P ($C \subseteq \Gamma_P$). Add

yet another attribute to C to create the resulting set B. If it is always true that B is also a qualified subset then the access structure is a monotone access structure, described as

$$\text{if } C \in \Gamma_P \text{ and } C \subseteq B \subseteq 2^P \text{ then } B \in \Gamma_P.$$

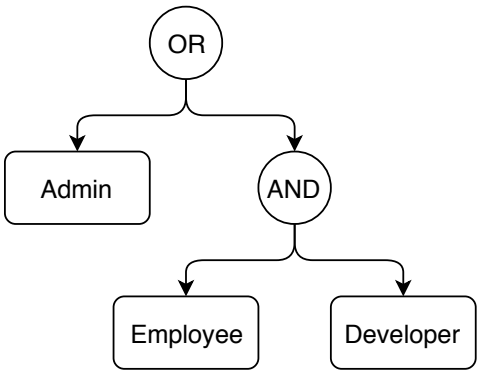

Figure 2.3: A simple monotone access structure.

In Figure 2.3, a simple monotone access policy is represented. In this policy the two subsets `(Admin)` and `(Employee, Developer)` are authorised to decrypt the associated ciphertext. So, any party, whose key contains all the attributes of at least one of the subsets in a ciphertext, can decrypt it.

2.4.3 Algorithms in ABE

- $Setup() \rightarrow pk,\ mk$: The setup process generates the public key pk, the master key mk.
- $Keygen(mk,\ a) \rightarrow sk_a$: The key generation process takes as input the master key mk and a set of attributes or an access policy a. It outputs a private key sk_a.
- $Encrypt(pk,\ a,\ m) \rightarrow c_a$: The encryption function takes as input the public key pk, a set of attributes or an access policy a and a plaintext message m. It outputs a ciphertext c_a.
- $Decrypt(sk_a,\ c_a) \rightarrow m$: The decryption function takes as input a private key sk_a and a ciphertext c_a. If the attributes and access policy in the key and ciphertext overlap, the decryption is successful, and the function outputs the plaintext m.

2.5 Discrete logarithm problem

Given a large prime number p and the multiplicative group $\mathbb{Z}_p^*$, it is very hard to compute discrete logarithms of the elements of this group. This is known as the discrete logarithm problem (DLP). Let g be a primitive root of $\mathbb{Z}_p^*$, p a prime and a

an arbitrary element in $\mathbb{Z}_p^*$. Then the element $g^a \mod p$ can easily be calculated, however given $g \mod p$ and $g^a \mod p$ it is hard to calculate a.

2.6 Elliptic-curve cryptography

Elliptic curve groups are very important to cryptography since they can be constructed in such a way that bilinear pairings can be realised.

An elliptic curve group is constructed by defining an elliptic curve, $E : y^2 = x^3 + ax + b$, over a finite field. This group has one binary operation, called the group law, denoted as $\oplus$. A line between two points at the curve will always intersect the curve at a third point, with the exception when l is a tangent on E, since any line $l : y = cx + d$ substituted into the curve E will give a cubic polynomial with three roots. When computing the result of group law applied to two points, P and Q, we draw a line through P and Q which will intersect E at the third point $\ominus R$. $\ominus$ denotes the point mirrored in the x-axis. The result R is the third intersection point, mirrored in the x-axis, namely $P \oplus R = R$. The computation is the same for the group law applied to a point P and itself, except the line l is then the tangent through P [10].

Computation of the group law is described graphically in Figure 2.4 where the group law is applied to the point P_1 four times, denoted as P_1^4. The curve is in this example visualised in $\mathbb{R}$, for simplicity. In reality, it would of course be some finite field $\mathbb{F}_q^*$.

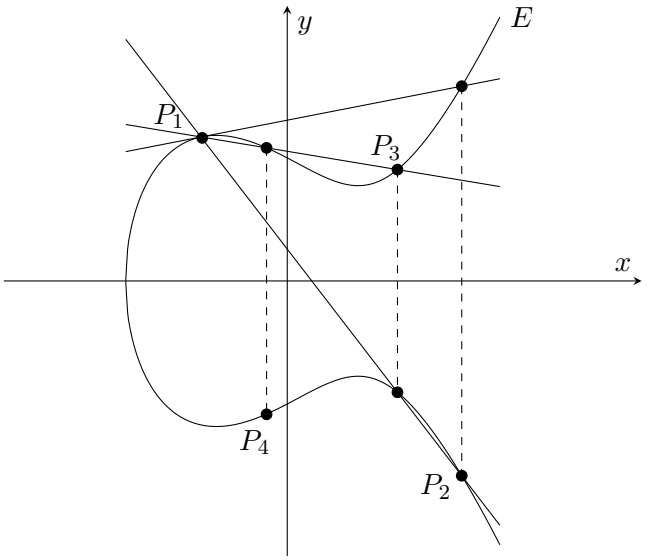

Figure 2.4: Dot operations on an elliptic curve.

If we repeat the group law operation with P_1, n times, we get $P_n = P_1^n$. The problem of finding n given P_1 and P_1^n, is called the elliptic curve discrete logarithm problem (ECDLP) and is the corresponding problem to the DLP in the elliptic

curve groups setting.

2.7 Bilinear pairings

Let G_0 and G_1 be two cyclic groups of the same order q, then bilinear paring is a map

$$e : G_0 \times G_0 \to G_1$$

which satisfies

Bilinearity: For all $P, Q \in G_0$ and all $a, b \in \mathbb{Z}_q$: $e(P^a, Q^b) = e(P, Q)^{ab}$.

Non-degeneracy: For all $P \in G_0$ and $P \neq 0$: $e(P, P) \neq 1$.

Computability: e is efficiently computable.

2.8 Cryptographic attack models

There are several attack models that can be considered when evaluating the security of a cryptographic scheme. They all assume some restrictions/capabilities of an adversary. One important aspect of the security in an attack model is the ciphertext indistinguishability. It refers to the situation where an adversary is given a ciphertext that is an encryption of one out of two known plaintexts. If the adversary cannot distinguish which of the two plaintexts that the ciphertext is an encryption of, then the scheme achieves ciphertext indistinguishability, also referred to as IND. The IND property can be defined in different attack models.

The indistinguishability under different attack models can be formally defined by a game where the adversary plays against a challenger. If the adversary, after playing the game, cannot distinguish which plaintext that has been decrypted with a greater probability than $\frac{1}{2} + \epsilon$, where ϵ is some negligible function, then the scheme is indistinguishable under the attack model. In [20], Sako defines the three commonly used security notions indistinguishability under chosen plaintext attack (IND-CPA), indistinguishability under chosen ciphertext attack (IND-CCA) and indistinguishability under adaptive chosen ciphertext attack (IND-CCA2). There also exists a slightly relaxed definition of IND-CCA2, called indistinguishability under replayable chosen ciphertext attack (IND-RCCA), introduced by Canetti, Krawczyk and Nielsen in [8].

In the IND-CPA game setting, the adversary is provided with an encryption oracle which takes a plaintext as input and gives encryption of that plaintext as output. The game is played in 4 phases.

1. The adversary makes any polynomially bounded number of operations including queries to the encryption oracle.
2. The adversary outputs two messages m_0 and m_1 to the challenger where $m_0 \neq m_1$. The challenger selects a random value $b \in \{0, 1\}$ and encrypts the message m_b and outputs the resulting ciphertext c_b to the adversary.
3. The adversary now repeats Phase 1.

4. The adversary outputs a guess $b' \in \{0, 1\}$. If $b' = b$, the adversary wins the game

In the IND-CCA, IND-CCA2 and IND-RCCA game settings, the adversary is, in addition to the encryption oracle, also provided with a decryption oracle which takes a ciphertext as input and gives the decryption of that ciphertext as output. The game is played in 4 phases.

1. The adversary makes any polynomially bounded number of operations including queries to the encryption oracle and decryption oracle.
2. The adversary outputs two messages m_0 and m_1 to the challenger where $m_0 \neq m_1$. The challenger selects a random value $b \in \{0, 1\}$ and encrypts the message m_b and outputs the resulting ciphertext c_b to the adversary.
3. (a) (IND-CCA) The adversary may not perform any additional operations in this step
   (b) (IND-CCA2) The adversary now repeats Phase 1, with the exception that it cannot query the decryption oracle for the decryption of c_b.
   (c) (IND-RCCA) The adversary now repeats Phase 1, with the exception that it cannot query the decryption oracle for the decryptions of any ciphertexts that decrypt to $m \in \{m_0, m_1\}$.
4. Eventually, the adversary outputs a guess $b' \in \{0, 1\}$. If $b' = b$, the adversary wins the game.

2.9 Security models

When proving the security of a cryptographic primitive, it is usually not possible to prove it secure without putting any restrictions on adversaries or making other assumptions [21]. Therefore, there exists different models in which one assumes some things to be true, to be able to complete the proof. Two of these models are of extra interest to us; standard model and random oracle model. This since they will be used later on to compare schemes of interest.

2.9.1 Standard model

In the standard model an adversary is only limited by time and computational power. The security of cryptographic primitives in the standard model are often based on complexity assumptions, which states that some problem cannot be solved in polynomial time. Schemes which security only rely on these complexity assumptions are said to be secure in the standard model. While security in this model is highly desirable, it is also very difficult to construct security proofs which achieves it [17].

2.9.2 Random oracle model

It is common that cryptographic primitives use pseudo-random functions to approximate random values. However, pseudo-random functions do *not*, as the name

suggest, produce truly random values, only values that are more or less difficult to predict. When trying to prove the security of the primitive, this becomes a problem if one cannot also prove that the pseudo-randomness does not affect the security of the rest of the primitive. The random oracle model is a way to circumvent this problem by replacing all pseudo-random functions with random oracles. These are hypothetical functions that do in fact return truly random values [6].

Chapter 3
System requirements

In this chapter we present the system, the different parties within it and their relationship to each other. Furthermore, we define the requirements that the system should fulfil.

3.1 Prerequisites

The system in question consists of three categories of different parties:

Customer: The customer is the owner of the data it produces.

Proxy: The proxy provides the storage service to the customer.

Partner: The partner is a party that the customer has delegated data access to.

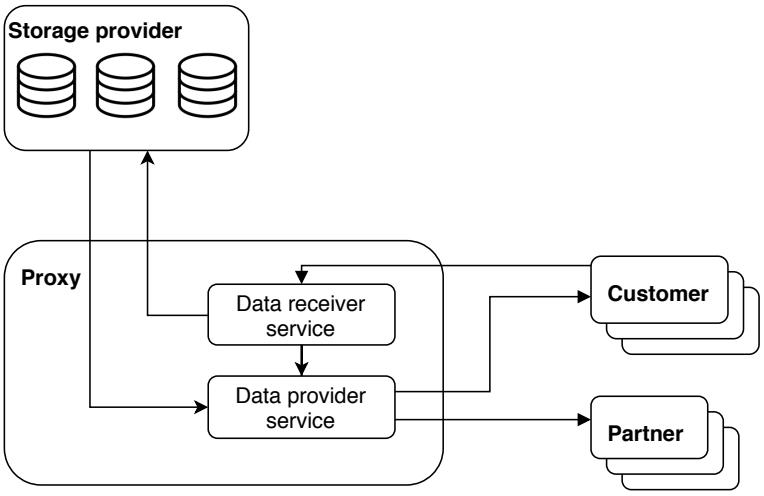

Figure 3.1: Schematic view of the system.

In the system there is a variable number of customers and partners, but there is only one proxy. Any single customer may delegate data access to several partners

and any single partner may have been delegated data access from several customers. Customers produce data, encrypts it, and uploads it to the proxy. The customers may download historic or live data from the proxy. Partners that have been delegated access to a customer's data may also download either historic or live data belonging to that customer from the proxy.

3.2 System requirements

The following requirements roughly define the properties that the system described above should have.

3.2.1 Minimal trust in Proxy and Partners

The customer does not trust the proxy, and so the proxy must not be able to decrypt any data that it stores for a customer. Furthermore, the proxy should not have the ability to delegate access to a customer's data to a partner without the customer's participation. The customer does not fully trust the partner and assumes that both the partner and the proxy might collude and combine their information. In that event, it must be unfeasible for them to reconstruct any secret information belonging to the customer.

3.2.2 End-to-end encryption

To ensure that data is not accessible to anyone other than the parties that should have access (i.e. the customer and partners with access) the data should be encrypted end-to-end. This means that the data should never exist in plaintext from the moment it is produced and encrypted, to arrival and decryption at an authorised partner.

3.2.3 Performance / Video data encryption capability

The data that will be transmitted is relatively large, and so the system must be able handle encryption, (re-encryption,) and decryption of that data at their respective location. Since the system should be able to provide live access to data produced by the customer, it is important not to add too much overhead in these processes that would consequently introduce delays.

3.2.4 Temporary decryption privileges capability

The customer should be able to revoke previously delegated access of a partner, or alternatively restrain the delegation to certain time periods so that the partner cannot access new data after that time. Ideally, the customer should not need to rely on the proxy to enforce this access revocation.

3.2.5 Single point of communication

The system should not require that a customer and a partner need to communicate directly with each other in order to not disclose any secret information to the proxy. All communication should be allowed to go through the proxy.

Chapter 4

System analysis of applicable techniques

In the following chapter, techniques for encrypting data are discussed based on their properties and how they can be applicable in the context of this thesis. In summary, a system which allows for dynamic delegation and revocation of decryption privileges is desired. The owner of the data should be the only authority that delegates these privileges, and the need for other trusted parties, e.g. for key generation, should be avoided.

4.1 Symmetric encryption

As a first step, the simplest type of encryption is considered, symmetric encryption. There are several problems with this type of cryptosystem. Firstly, a private key is used to encrypt data. If the encryption takes place in an unsafe place, in a camera for example, it is not ideal to store the key there. Secondly, to achieve some sort of revocation of decryption privileges, the symmetric key would need to be periodically changed. Thereafter, it would need to be distributed to all devices performing encryption, and all parties with continued decryption privileges. Thirdly, the private key is stored at a potentially large number places, which increases the risk for it leak. If it does, all data encrypted with that key is compromised. Conclusively symmetric encryption *alone* is not enough to achieve the properties we want.

4.2 Conventional asymmetric encryption

Using asymmetric encryption would solve the problem with having to store a private key at devices performing encryption. Asymmetric encryption is instead done using a public key. This key can safely be stored anywhere. However, in order to achieve the possibility to revoke decryption privileges, the key-pair would still need to be periodically changed and distributed to all parties. The public key to the devices that perform encryption, and the private key to parties with continued decryption privileges. This is still a very impractical way of achieving our requirements. Therefore, conventional asymmetric encryption is not enough.

4.3 Attribute-based encryption

A more advanced form of asymmetric encryption is ABE. During the setup process of the cryptosystem, a public and master key is generated. When an authority generates a new private key, this is done by combining the master key with a set of attributes. This would consequently mean an owner of data can produce several private keys that can decrypt data encrypted under its public key, a one-to-many relationship between public and private keys. Additionally, the use of attributes can be used to achieve fine-grained access control and in particular revocation of decryption privileges. This makes ABE to a strong candidate as cryptosystem for enabling distribution of encrypted data, considering that each data owner will be its own authority, and control the key generation and encryption process of generated material under their public key. By controlling encryption and key generation, the customer is in full authority of delegation of decryption privileges of their data.

4.3.1 Key-policy

In key-policy ABE (KP-ABE) private keys are associated with access policies and ciphertexts are associated with descriptive attributes. This means that the key issuer is the one who controls which ciphertexts a key can decrypt and not the encryptor. The encryptor only includes a set of descriptive attributes into the ciphertext [4]. This is not an issue in our system since the encryptor and key generator is the same party. It is however not desirable that any other process than the encryption process should govern the access policy. Changes in the system setup could then potentially result in issues directly related to the use of KP-ABE if for example some process or party wants to encrypt a message but do not, for some reason, fully trust the key generator.

4.3.2 Ciphertext-policy

In ciphertext-policy ABE (CP-ABE) private keys are associated with attributes and ciphertexts are associated with access policies (the reversed of KP-ABE). Since the access policy now instead is included into the ciphertext, the encryptor is the party who controls which keys that can decrypt the ciphertext. This is a more suitable property for our requirements since the encrypting unit should define which type of keys that should be able to decrypt the resulting ciphertext.

An important scheme in the ABE category is the scheme presented by Bethencourt, Sahai and Waters in [4] 2007. They introduced the first scheme of the type CP-ABE. However, the scheme was only proved to be secure in the generic group model [22]. Four years later Waters presented an improved scheme in [22] (from this point referred to as W11) that had similar characteristics as [4] but secure in the standard model.

In both [4, 22], the authors describe that by adapting the technique proposed by Canetti, Halevi and Katz [7] both schemes can be made CCA2-secure without complex modifications of the original designs. This is achieved by signing ciphertexts and verifying that signature before decrypting. If the verification fails, the decryption returns an error.

4.4 Proxy re-encryption

Proxy re-encryption does also have properties which makes it suitable for distributing data in a cloud storage. With PRE, it is possible to delegate decryption privileges to other parties, without having to disclose the private key. Instead, ciphertexts are re-encrypted so that they can be decrypted with other parties' private keys, thus achieving a one-to-many relationship between public and private keys.

A property worth investigating is how bilinear pairings affect the performance of a scheme in practice. While they provide features important for cryptography, they are also computationally expensive to realise compared to hash functions which usually replaces pairings in pairing free PRE schemes. The scheme proposed by Chow, Weng, Yang and Deng in [9] for example, does not make use of pairings. This scheme is in turn only proved secure in the random oracle model. By using pairings, the scheme can be kept secure in the standard model and it allows relatively simple and powerful ways of constructing PRE.

Another aspect of PRE schemes that is highly relevant in the current research is CCA2-security. Protection against RCCA can be realised using some sort of verification of ciphertexts. This needs to be done before any ciphertexts are re-encrypted by the proxy or decrypted by a partner or customer. For example, Libert and Vergnaud realises this in [14] (from this point referred to as LV11), using one-time signatures (OTS) similar to [7]. This is however not as simple as in the case of [4, 22] due to the extra re-encryption step. Therefore, it introduces much more complexity in the scheme. For example, comparing the number of pairing operations in [14] and the scheme proposed by Ateniese, Fu, Green and Hohenberger in [3] (from this point referred to as AFGH06), one can see that in AFGH06 there is only one pairing. This is done in the re-encryption operation. In LV11 on the other hand, there is one pairing in the encryption operations, two in the re-encryption operation, five in the decryption of level one ciphertexts and three in the decryption of level 2 ciphertexts and additionally signing and verification operations. Zhao et al. have a similar approach in [23] as LV11, but without OTS. However, they also require a significant increase in number of pairings compared to AFGH06.

4.4.1 Proxy re-encryption properties

The existence of a proxy in PRE causes some additional properties to arise, which may or may not be of interest depending on the use case. The following reasoning establishes what kind of PRE scheme we need to look for to use in our system.

Directionality

Bidirectionality allows re-encryption of messages in both ways, which in itself is not something bad, and our system requirements does not prohibit it. One could argue that bidirectionality would leave the possibility for partners to send data back to the customer, the same way the customer sent the partner data, open. This can be a good thing as it is not implausible that this feature would be desirable in the future. Even though it is not part of the requirements today. However, we have

yet to discover any scheme achieving bidirectionality, without the need of both the customers and the partners private key in the re-encryption key generation process. This violates the system requirements as no party should trust one another and so, no party want to disclose their private key to another party.

Unidirectionality is in a sense a stronger property than bidirectionality since a unidirectional scheme can provide communication both ways, just as a bidirectional scheme does, by simply creating re-encryption keys for both ways. For example, in a setting where Alice and Bob want to communicate with each other, two unidirectional keys $rk_{Alice \to Bob}$ and $rk_{Bob \to Alice}$, will achieve the same communication capabilities as one bidirectional key $ReK_{Alice \leftrightarrow Bob}$. So, in conclusion we want to find solution utilising a unidirectional scheme.

Transitivity

Transitivity is not a property that would be in any way useful in this system. Customers delegates decryption privileges to specific partners only, and it should only be the customer that produces these delegations. However, if the scheme used in the system would be transitive, it would probably not be a problem anyway since to utilise that property there should exist re-encryption keys such that $rk_{A \to B}$ and $rk_{B \to C}$. If A is a customer, then B is a partner. But partners never delegate decryption privileges to anyone else and so there will never be any re-encryption key $rk_{B \to C}$.

Collusion resistance

Collusion resistance is a vital property that any scheme used in this system needs. The private key of a customer is of course secret because with it one can decrypt any data produced by the customer. Since we do not trust the proxy which holds the re-encryption keys nor do we trust all the partners, the scheme must be resistant against collusion between partners and the proxy.

Interactivity

At first sight, it may not seem that an interactive scheme would incur any problems. The partner does have to share their private key, which in general is bad, with a customer to accept delegations. But, if the partner's key-pair is only used in this particular system, what harm is there to share their key with that customer? It becomes a problem when one considers whether different customers trust each other. Since a partner can accept delegations from several customers, then, if customers A and B both have delegated decryption rights to partner C, both A and B possess C's private key sk_C. This means that for example B can decrypt data produced by A, that has been re-encrypted to C. To avoid this problem, we consider schemes that are not interactive.

Single-hop or multi-hop

This property does not influence the choice of scheme in this case, since data will only be re-encrypted once, from a customer to a partner. Therefore, a scheme

with single-hop is sufficient. However, a scheme with multi-hop capability is not necessarily bad, but it will never be used.

4.5 Overview

The reasoning in this chapter boils down to a decision tree of usable cryptosystems, which can be seen in Figure 4.1. The techniques that are of interest and that we will continue to investigate are PRE and ABE. PRE schemes should be unidirectional, non-transitive, non-interactive and collusion resistant. ABE schemes should use ciphertext-policy and be either monotonic or non-monotonic access structures.

The matter of capability to grant temporary decryption privileges will be further discussed in the next chapter. Revocation of issued keys is a fundamental problem in cryptography. We will however see that it is possible to attribute keys and ciphertexts with time stamps in such a way that keys will expire and not be able to decrypt ciphertexts produced after the keys expiration date.

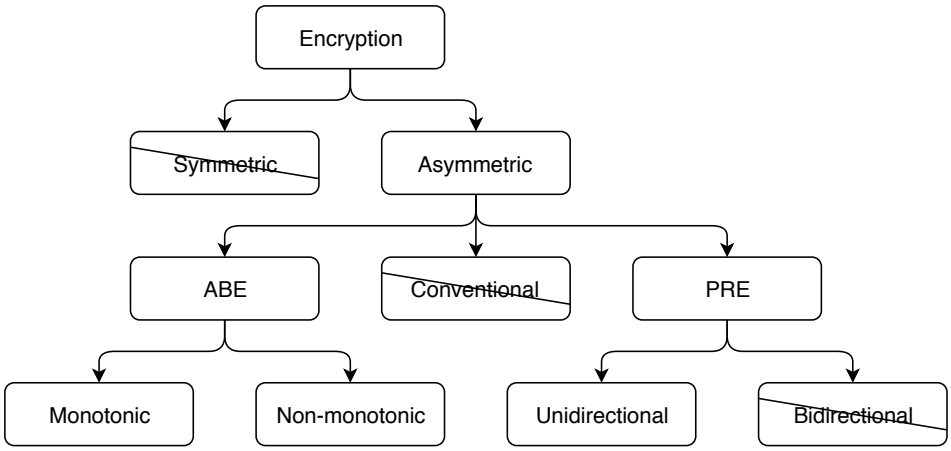

Figure 4.1: Decision tree for use of cryptosystem.

Chapter 5
Design

In this chapter the asymmetric schemes, chosen in Chapter 4, are deployed on the specified system and solutions for overcoming issues are presented. The schemes are deployed along with symmetric encryption and time stamps for key revocation.

5.1 Hybrid encryption

We have established that PRE and ABE are the most promising techniques to use in order achieve the system requirements. It is however, not enough to only apply PRE or ABE on the data that needs protection. Since PRE and ABE are asymmetric encryption techniques, they suffer from the drawbacks of being slow compared to symmetric encryption. Thus, they are not suitable to encrypt of arbitrary data. A common approach to overcome this is to use hybrid encryption. In such approach the advantages of both asymmetric and symmetric encryption are incorporated into one solution. This is done by firstly, encrypt the data with a symmetric encryption scheme using a randomly generated symmetric key, which is relatively fast and effective. These steps can be seen in Figure 5.1 where data is encrypted using symmetric encryption and the key is encrypted using asymmetric encryption. Once the data has been encrypted, the asymmetric scheme encrypts the key used by the symmetric encryption with the public key of the asymmetric scheme. This approach of using hybrid encryption is more practically realistic than without. Since asymmetric schemes can only encrypt elements from the group they are instantiated with, any arbitrary data would first need to be chunked up into pieces that can be encoded as a group element. All these pieces would then be encrypted by the relatively costly asymmetric encryption algorithm, which makes the whole process impractical.

By combining symmetric and asymmetric encryption it is possible to ensure that the system is feasible from a performance perspective. This while benefiting from the properties of the asymmetric scheme.

5.2 First design

The following sequence diagrams show a first design of how both PRE and ABE schemes (see Figure 5.2 respectively Figure 5.3) would be used in combination with

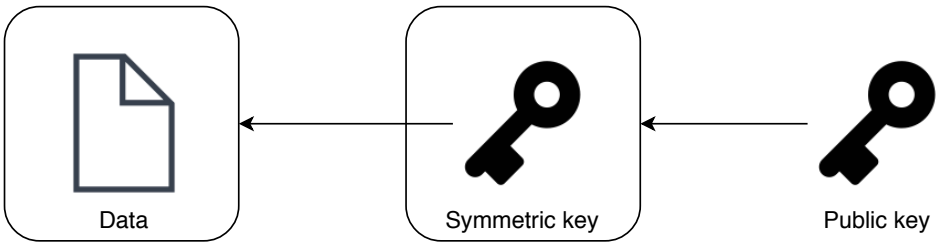

Figure 5.1: The basics of hybrid encryption.

a symmetric scheme. The diagrams show the steps required to give decryption privileges and transfer an arbitrary message to a partner.

5.2.1 Proxy re-encryption

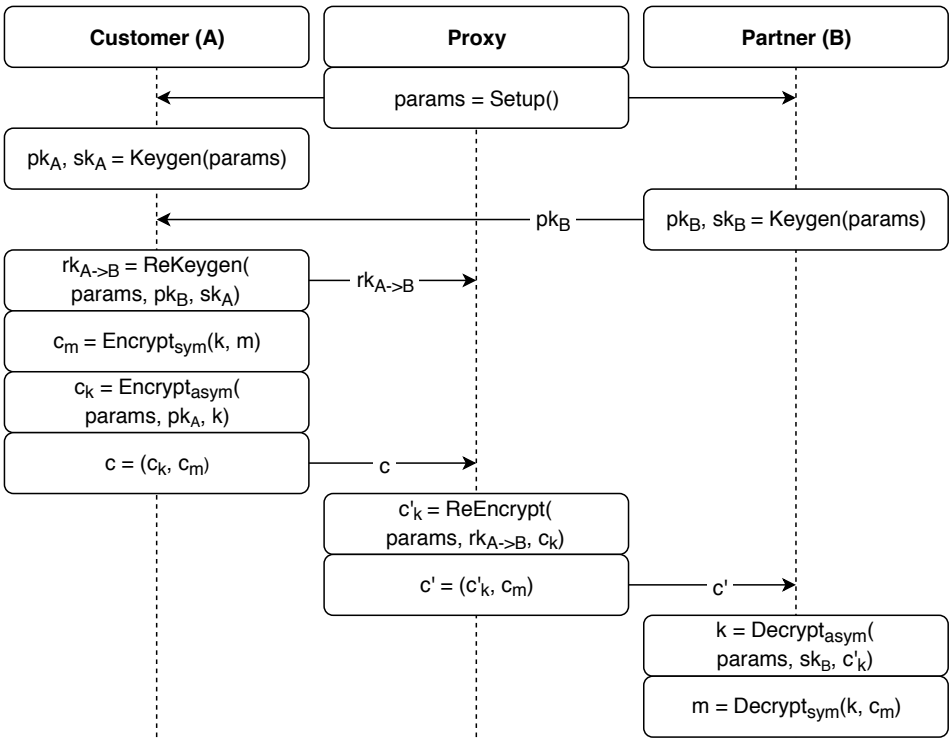

Figure 5.2: Sequence diagram for PRE with symmetric encryption.

In Figure 5.2 the result of combining PRE with symmetric encryption can be viewed. In this cryptographic system the proxy is an active participant with the purpose of re-encrypting relayed messages.

1. $Setup() \rightarrow params$: The proxy generates the global parameters $params$ to

be used in the PRE system.

2. $Keygen(params) \rightarrow sk_A,\ pk_A$: The customer generates its public key pk_A, and private key sk_A from the parameters $params$.
3. $Keygen(params) \rightarrow sk_B,\ pk_B$: The partner generates its public key pk_B, and private key sk_B from the $params$. The public key is sent to the customer.
4. $ReKeygen(params,\ pk_B,\ sk_A) \rightarrow rk_{A \rightarrow B}$: The customer combines its private key sk_A with the public key pk_B it received from the partner, along with the global parameters $params$ and generates a re-encryption key $rk_{A \rightarrow B}$. The re-encryption key is sent to the proxy.
5. $Encrypt_{sym}(k,\ m) \rightarrow c_m$: The customer generates a random symmetric key k and encrypts message m with a symmetric encryption scheme.
6. $Encrypt_{asym}(params,\ pk_A,\ k) \rightarrow c_k$: The random symmetric key k generated in the previous step is encrypted with the public key pk_A of the customer in an asymmetric PRE scheme.
7. The ciphertext c is a tuple consisting of the encrypted message c_m, and the encrypted key c_k used in the symmetric scheme. This tuple is sent to the proxy.
8. $ReEncrypt(params,\ rk_{A \rightarrow B},\ c_k) \rightarrow c'_k$: The proxy re-encrypts the encrypted symmetric key c_k into a ciphertext c'_k with the re-encryption key $rk_{A \rightarrow B}$ it earlier received from the customer.
9. The new tuple c' containing c'_k and c_m is sent to the partner.
10. $Decrypt_{asym}(params,\ sk_B,\ c'_k) \rightarrow k$: The partner decrypts the re-encrypted ciphertext c'_k with its private key sk_B and obtains the symmetric key k.
11. $Decrypt_{sym}(k,\ c_m) \rightarrow m$: With the symmetric key k it extracted in the previous step the partner decrypts ciphertext c_m and obtains the message m.

5.2.2 Attribute-based encryption

In the Figure 5.3 ABE is combined with symmetric encryption to transfer a message from the customer to the partner, without disclosing the master key or revealing the ciphertext content to the proxy. The proxy is not participating in the cryptographic system, it is only relaying information between the two parties.

1. $Setup() \rightarrow pk,\ mk$: The customer generates a public key pk and a master key mk.
2. $Keygen(mk,\ a) \rightarrow sk_a$: By combining the master key mk and a set of attributes a the customer generates a private key sk_{A^*}.
3. $Encrypt_{sym}(k,\ m) \rightarrow c_m$: The customer generates a random symmetric key k and encrypts the message m using a symmetric encryption scheme to produce the ciphertext c_m.

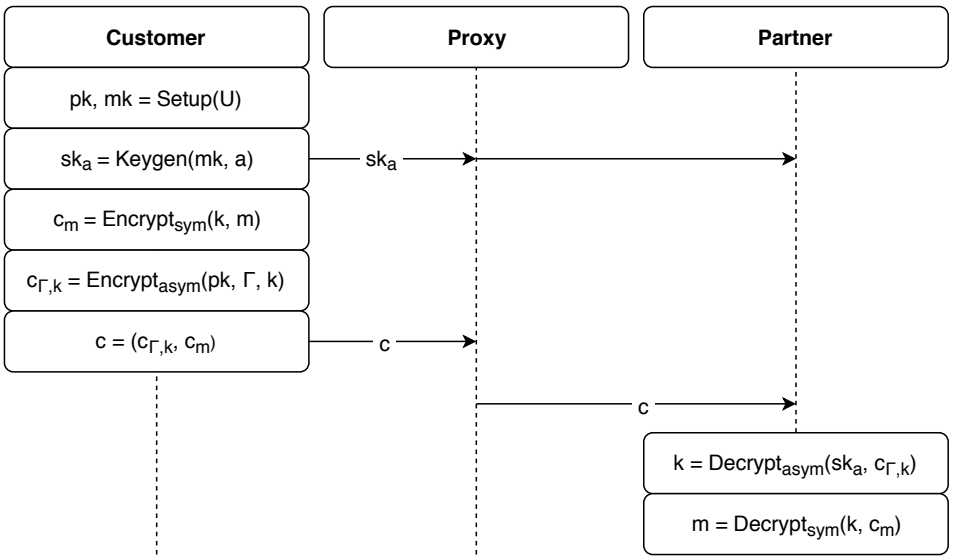

Figure 5.3: Sequence diagram for ABE with symmetric encryption.

4. $Encrypt_{asym}(pk,\ \Gamma,\ k) \rightarrow c_{\Gamma,k}$: The symmetric key k used in the symmetric encryption is now encrypted using an CP-ABE scheme with the public key pk of the customer, and an access policy Γ.

5. The resulting ciphertext c which is sent is a tuple of the encrypted symmetric key $c_{\Gamma,k}$ and the encrypted message c_m.

6. $Decrypt_{asym}(sk_a,\ c_{\Gamma,k}) \rightarrow k$: The partner decrypts the ciphertext $c_{\Gamma,k}$ containing the symmetric key k with its private key sk_a. This step is only successful if the attribute set a fulfils the requirements of the access policy Γ in the private key and ciphertext.

7. $Decrypt_{sym}(k,\ c_m) \rightarrow m$: In the last step the partner decrypts the ciphertext c_m with the symmetric key k extracted from the previous step to obtain the message m.

5.3 Generation rate of symmetric keys

An important thing to note in the designs in Section 5.2, is that it is not required to randomly generate a new symmetric key for each message that should be encrypted. Depending on the practical setting, such as in live video stream, the additional computations required by the asymmetric encryption may impose performance problems. Because of that, we may allow us to use each symmetric key to encrypt several messages, say x of them. When doing so we can remove Step 6, Step 8 and Step 10 in Section 5.2.1 as well as Step 4 and Step 6 in Section 5.2.2, in the process $x-1$ out of x times. This means that we can arbitrarily reduce the level of impact the PRE or ABE scheme has on the performance at the expense

of security. In the case where PRE is used, security is lowered in the sense that a partner is potentially able to decrypt data that has not been re-encrypted and sent to them by the proxy. This is because previously received re-encrypted data may have exposed the symmetric key for that data. The symmetric key can therefore be viewed as a session key for the time that it is used. It is important to ensure that the initialisation vector is properly randomised and not re-used [11].

5.3.1 Revocation of decryption privileges

As mentioned in Section 3.2.4 the system requires some form functionality to revoke previously delegated data access. One way to achieve this could be to embed time information in the encrypted data and re-encryption keys, in such a way that a re-encryption of a ciphertext will only yield correct data, if the time information used during encryption matches the one in the re-encryption key. With such an approach the customer can have full control over whether a partner has access to their data, by periodically generating new re-encryption keys for new time periods. When the customer wishes to revoke access from a partner, they simply stop generating new re-encryption keys for that party. Another way to achieve revocation of decryption capabilities can be done by throwing away the re-encryption keys stored at the proxy. This would result in the immediate stop of re-encryption of material to the dedicated recipient of the deleted key. However, this assumes that the data owner trusts the proxy to perform this task. If the trust for the proxy is low, the data owner can choose to decrease the size of the time periods and obtain a more fine-grained revocation to the cost of having to perform the re-encryption key generation process more frequently.

In practice, it would probably be useful to use a combination of them both. This way, the customer can request that the proxy should discard a certain re-encryption key to immediately revoke the corresponding partner's decryption privileges. However, since the proxy is only partially trusted by the customer, the customer cannot fully trust that a key really is discarded (revoked) until the end of that time period. But, the customer does at least have a bounded time span that decryption privileges may persist after they have requested it to be removed.

5.4 Final design

The design outlined in Section 5.2 does not completely cover the system requirements. First of all, there is no way of revoking previously issued decryption privileges. In ABE there is, but we will specify that explicitly in the design. Secondly, we need to be able to lower the time spent on the computationally expensive asymmetric operations.

5.4.1 Proxy re-encryption

Combining the concept of temporary decryption delegation, by embedding time information in ciphertexts and re-encryption keys, and introducing sessions for symmetric keys we arrive at the following design, described in Figure 5.4.

1. $Setup() \rightarrow params$: The proxy generates the global parameters $params$ to be used in the PRE system.
2. $Keygen(params) \rightarrow sk_A,\ pk_A$: The customer generates its public key pk_A, and private key sk_A from the parameters $params$.
3. $Keygen(params) \rightarrow sk_B,\ pk_B$: The partner generates its public key pk_B, and private key sk_B from the $params$. The public key is sent to the customer.
4. $ReKeygen(params,\ pk_B,\ sk_A,\ t) \rightarrow rk_{A \rightarrow B,t}$: The customer combines its private key sk_A with the public key pk_B it received from the partner along with a time period and the global parameters $params$. From this it generates a re-encryption key $rk_{A \rightarrow B,t}$, which is valid during the time period t. The re-encryption key is sent to the proxy.
5. $Encrypt_{sym}(k,\ m) \rightarrow c_m$: The customer either generates a new random symmetric key k or use a previously generated symmetric key and encrypts message m with a symmetric encryption scheme.
6. $Encrypt_{asym}(params,\ pk_A,\ k,\ t) \rightarrow c_{k,t}$: If a new symmetric key k was generated in the previous step, it is encrypted with the public key pk_A of the customer in an asymmetric PRE scheme. If a previously generated key was used in the previous step, the encryption of that key is re-used.
7. The ciphertext c is a tuple consisting of the encrypted message c_m and the encrypted key $c_{k,t}$ used in the symmetric scheme. This tuple is sent to the proxy.
8. $ReEncrypt(params,\ rk_{A \rightarrow B,t},\ c_{k,t}) \rightarrow c'_k$: The proxy re-encrypts the encrypted symmetric key $c_{k,t}$ into a ciphertext c'_k with the re-encryption key $rk_{A \rightarrow B,t}$ it earlier received from the customer. If $c_{k,t}$ is equal to the encrypted symmetric key from the previous iteration, the re-encryption of that key is re-used.
9. The new tuple c', containing c'_k and c_m, is sent to the partner.
10. $Decrypt_{asym}(params,\ sk_B,\ c'_k) \rightarrow f$: If c'_k is equal to the re-encrypted symmetric key from the previous iteration, the partner re-uses the decrypted key from that iteration. Otherwise, the partner decrypts the re-encrypted ciphertext c'_k with its private key sk_B and obtains the symmetric key k.
11. $Decrypt_{sym}(k,\ c_m) \rightarrow m$: With the symmetric key k it extracted in the previous step the partner decrypts ciphertext c_m and obtains the message m.

5.4.2 Attribute-based encryption

Temporary decryption delegation can be achieved with the previous design by including a time period in the access policy of the ciphertext and as an attribute in generated keys. However, we will specify it explicitly that the key generation algorithm and the encrypt algorithm always takes a time period as input. Similarly,

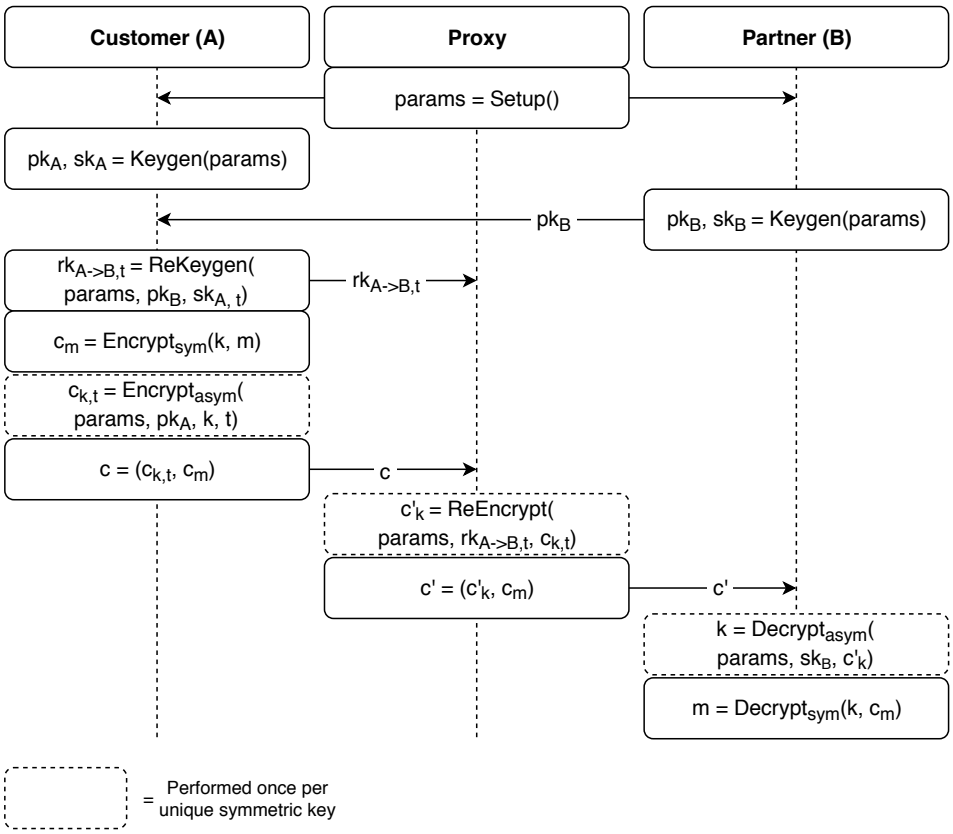

Figure 5.4: Sequence diagram for temporary PRE with hybrid encryption and re-use of symmetric keys.

as above, symmetric session keys are also included. The sequence diagram can be viewed in Figure 5.5.

1. $Setup() \rightarrow pk,\ mk,$: The customer generates a public key pk and a master key mk.

2. $Keygen(mk,\ a,\ t) \rightarrow sk_{a,t}$: By combining the master key mk, a set of attributes a, and a time period t the customer generates a private key $sk_{a,t}$.

3. $Encrypt_{sym}(k,\ m) \rightarrow c_m$: The customer either generates a new random symmetric key k or use a previously generated symmetric key and encrypts the message m using a symmetric encryption scheme to produce the ciphertext c_m.

4. $Encrypt_{asym}(pk,\ \Gamma,\ k, t) \rightarrow c_{\Gamma,t}$: If a new symmetric key was generated in the previous step it is now encrypted using an CP-ABE scheme with the public key pk of the customer and an access policy Γ. Otherwise, the encryption of the key generated in the previous iteration is re-used.

5. The resulting ciphertext c which is sent is a tuple of the encrypted symmetric key $c_{\Gamma,t}$, and the encrypted message c_m.
6. $Decrypt_{asym}(sk_{a,t}, c_{\Gamma,t}) \rightarrow k$: If $c_{\Gamma,t}$ equals ciphertext from the previous iteration the decrypted key from that iteration is re-used. Otherwise, the partner decrypts the ciphertext $c_{\Gamma,t}$ containing the symmetric key k with its private key $sk_{a,t}$. This step is only successful if the attribute set a and the time period t in the private key fulfils the requirements of the access policy Γ and the time period t in the ciphertext.
7. $Decrypt_{sym}(k,\ c_m) \rightarrow m$: In the last step the partner decrypts the ciphertext (c_m) with the symmetric key k, extracted from the previous step, to obtain the message (m).

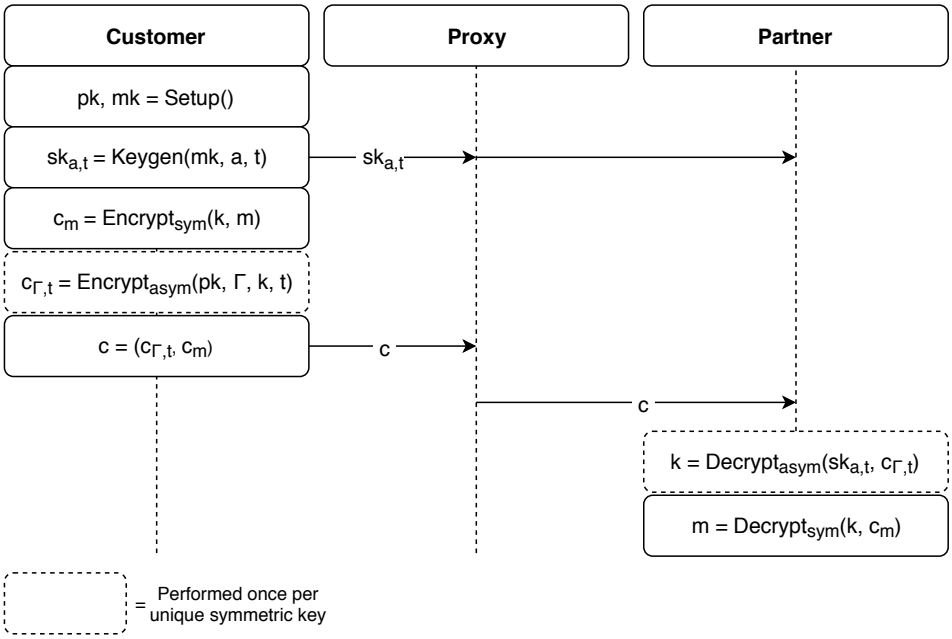

Figure 5.5: Sequence diagram for temporary ABE with hybrid encryption and re-use of symmetric keys.

Chapter 6
Implementation

In the previous chapter the potential and issues of different types of cryptosystems were discussed and concluded. The decision fell on ABE and PRE as they accommodate properties of special interest for this thesis objectives. In the start of this chapter the choice on which specific schemes to focus on is made. This is then followed up with the necessary modifications for overcoming shortcomings in these. And finally, proof is presented that the security of the modified schemes still is preserved.

6.1 Choice of schemes

As discussed in Section 4.4, there are several PRE and ABE schemes with different properties. Regarding PRE, with help of the analysis made by Nuñez, Agudo and Lopez in [18], the two schemes AFGH06 and LV11 are found to be suitable for our needs. They are both well renowned schemes and they fulfil the required properties. The big difference between them is that AFGH06 is IND-CPA and relatively fast, while LV11 is IND-RCCA and relatively slow.

When it comes to ABE, the choice fell on the scheme W11. Just like AFGH06 and LV11, it is too a well renowned scheme fulfilling the required properties.

The implementation of the schemes needed to be based on a cryptographic library containing operations for elliptic curve cryptography and bilinear pairing. We found the library Charm [1] to be very well suited for this. It is a framework intended for prototyping of cryptosystems using the high-level language Python. However, computationally intensive operations are performed by native C modules, in order to keep the performance at a high level. Implementations of AFGH06 and W11 were already present in Charm, but it lacked one for LV11, which we implemented ourselves. In order to implement the LV11 scheme, an OTS scheme was required. We decided to implement and use the simple hash-based signing algorithm described by Lamport in [13]. Furthermore, a benchmarking program was written and used to measure the performance of all the schemes and their following modified versions.

Lastly, the possibility to combine PRE and ABE with symmetric encryption, to achieve the desired hybrid encryption, described in 5, was implemented.

6.2 CCA2-security of attribute-based encryption

The original ABE scheme W11, presented by Waters in [22], is not CCA2 secure, but, as discussed in Section 4.3, it can be made IND-CCA2 if the modification presented by Canetti, Halevi and Katz in [7] is applied on the scheme. As W11 already existed as an implementation in the Charm library we extended this and added the feature needed for CCA2 security. We make use of the Lamport OTS [13] to accomplish ciphertext verification, to ensure that ciphertexts are unaltered at the point of decryption.

Let *Encrypt* and *Decrypt* denote the original encryption and decryption algorithms in W11. Furthermore, $SigGen$, $Sign$, $Verify$ are the key generation, signing and verification algorithms respectively, in the Lamport OTS scheme. Γ is some access policy, sk_a a with the attribute set a, and pk the public key. We can now define modified versions of the original algorithms, to construct a new CCA2-secure version of the W11 scheme, which we will call W11-CCA2. The operations in W11-CCA2 are defined as following:

Encrypt'$(pk,\ \Gamma,\ m)$: (1)
$c' \leftarrow Encrypt(pk,\ \Gamma,\ m)$
$svk,\ ssk \leftarrow SigGen()$
$c = (svk,\ c,\ Sign(ssk,\ c))$
Return: c

Decrypt'$(sk_a,\ c)$: (2)
$svk,\ c',\ \sigma \leftarrow c$
if $Verify(svk,\ c') \neq 1$: Return invalid
$m \leftarrow Decrypt'(sk_a,\ c')$
Return: m

6.3 Expiration property of re-encryption keys

As discussed in Section 5.3.1, the system requires some way to revoke decryption rights that has previously been delegated to a partner. ABE inherently has this capability. For example, using ciphertext-policy ABE, messages would be encrypted using an access policy that requires a specific time stamp. For PRE however, it is not that simple. The trivial way to do it, as mentioned before, would be for the proxy to stop re-encrypt material for a partner whose decryption rights has been revoked. However, in this case there are no cryptographic insurances that material would not get re-encrypted anyway and distributed to the partner. There is only a matter of trust in that the proxy does what it is supposed to do.

The better option would be to embed the time data into the re-encryption keys, so that they are only valid for data encrypted with matching time data.

6.3.1 A first attempt

In our first attempt to embed time data, the customer would sign time stamps of a chosen granularity with their private key. This can be done in advance and these

signed time stamps are then used during encryption. Note that it only makes sense to embed time data when performing level 2 encryption. This since the purpose is to assure that re-encryption keys have valid time stamps, and only level 2 encrypted messages can be re-encrypted.

AFGH06

To realise this construction in the PRE scheme AFGH06, let us recall the level 2 encryption, re-encryption key generation and level 2 decryption algorithms. G and G_T are two groups of prime order with the bilinear mapping $e : G \times G \to G_T$. The global parameters are $params = (g,\ Z)$ where g is a random generator in G and $Z = e(g, g)$ is a random generator in G_T. The two users, A and B, have one key-pair each; $(pk_A,\ sk_A) = ((Z^{a_1},\ g^{a_2}),\ (a_1,\ a_2))$ and $(pk_B,\ sk_B) = ((Z^{b_1},\ g^{b_2}),\ (b_1,\ b_2))$ respectively. $\xleftarrow{R}$ denotes a random number generator.

Encrypt$_\mathbf{2}$ $(params,\ pk_A,\ m)$: (1)
$g \leftarrow params$
$Z^{a_1} \leftarrow pk_A$
$r \xleftarrow{R} \mathbb{Z}_q^*$
$c = (g^r,\ m \cdot Z^{a_1 \cdot r})$
Return: c

ReKeygen$(params,\ sk_A,\ pk_B)$: (2)
$a_1 \leftarrow sk_A$
$g^{b_2} \leftarrow pk_B$
$rk_{A \to B} = g^{b_2 \cdot a_1}$
Return: $rk_{A \to B}$

Decrypt$_\mathbf{2}$ $(params,\ sk_A,\ c)$: (3)
$g^r,\ m \cdot Z^{a_1 \cdot r} \leftarrow c$
$a_1 \leftarrow sk_A$
$m = m \cdot Z^{a_1 \cdot r} / e(g^r,\ g^{a_1})$
Return: m

Let us now introduce a new algorithm called $SignTimestamp$. $sk_A = a_1, a_2$ is the private key of user A and t is an element representing a time stamp.

SignTimestamp$(sk_A,\ t)$: (4)
$a_1 \leftarrow sk_A$
$st = t^{a_1}$
Return: st

The actual time stamps can be anything representing some time interval. For example, the string `'2018-36'` could represent the 36th week of 2018. The object representing the time interval is encoded into $\mathbb{Z}_q$ before it is given to the $SignTimestamp$ algorithm. We assume that the time stamp t is public knowledge, but since DLP is hard, st is not feasible to compute for someone that does not possess sk_A, i.e. anyone else but the customer.

We can now modify the level 2 encryption, re-encryption key generation and level 2 decryption algorithms, to construct a new version of the AFHG06 scheme

with ability to make temporary delegations, as

(5)

Encrypt$_2$'$(params,\ pk_A,\ m,\ st)$:
$g \leftarrow params$
$Z^{a_1} \leftarrow pk_A$
$r \xleftarrow{R} \mathbb{Z}_q^*$
$c_{st} = (g^r,\ m \cdot Z^{a_1 \cdot r \cdot st})$
Return: c_{st}

(6)

ReKeygen'$(params,\ sk_A,\ pk_B,\ st)$:
$a_1 \leftarrow sk_A$
$g^{b_2} \leftarrow pk_B$
$rk_{A \xrightarrow{st} B} = g^{b_2 \cdot a_1 \cdot st}$
Return: $rk_{A \xrightarrow{st} B}$

(7)

Decrypt'$_2$ $(params,\ sk_A,\ c_{st},\ st)$:
$g^r,\ m \cdot Z^{a_1 \cdot r \cdot st} \leftarrow c$
$a_1 \leftarrow sk_A$
$m = m \cdot Z^{a_1 \cdot r \cdot st} / e(g^r,\ g^{a_1 \cdot st})$
Return: m

We will refer to this scheme as AFGH06 Temp. v1. To verify that the modifications hold, let us check the decryption of a level 2 encrypted message, and the decryption of a re-encrypted message. The parameter *params* is omitted below. For decryption of a level 2 message

$$
\begin{aligned}
m &= Decrypt_2'(sk_A,\ c_{st},\ st) \\
&= Decrypt_2'(sk_A,\ Encrypt_2'(pk_A,\ m,\ st),\ st) \\
&= Decrypt_2'(sk_A,\ (g^r,\ m \cdot Z^{a_1 \cdot r \cdot st}),\ st) \\
&= \frac{m \cdot Z^{a_1 \cdot r \cdot st}}{e(g^r, g^{a_1 \cdot st})} \\
&= \frac{m \cdot Z^{a_1 \cdot r \cdot st}}{Z^{a_1 \cdot r \cdot st}} \\
&= m
\end{aligned}
$$

and decryption of a re-encrypted message

$$
\begin{aligned}
m &= Decrypt_1(sk_B,\ c'_{st}) \\
&= Decrypt_1(sk_B,\ ReEncrypt(rk_{A \xrightarrow{st} B},\ c_{st})) \\
&= Decrypt_1(sk_B,\ ReEncrypt(ReKeygen'(sk_A,\ pk_B,\ st),\ Encrypt'_2(pk_A,\ m,\ st))) \\
&= Decrypt_1(sk_B,\ ReEncrypt(g^{a_1 \cdot b_2 \cdot st},\ (g^r,\ m \cdot Z^{a_1 \cdot r \cdot st})) \\
&= Decrypt_1(sk_B,\ (e(g^r, g^{a_1 \cdot b_2 \cdot st}),\ m \cdot Z^{a_1 \cdot r \cdot st})) \\
&= Decrypt_1(sk_B,\ Z^{r \cdot a_1 \cdot b_2 \cdot st},\ m \cdot Z^{a_1 \cdot r \cdot st})) \\
&= \frac{m Z^{a_1 \cdot r \cdot st}}{(Z^{r \cdot a_1 \cdot b_2 \cdot st})^{\frac{1}{b_2}}} \\
&= \frac{m Z^{a_1 \cdot r \cdot st}}{Z^{r \cdot a_1 \cdot st}} \\
&= m
\end{aligned}
$$

It is easy to verify that the signed time stamp st need to match, in the encrypted message and in the re-encryption key, when decrypting a re-encrypted message or else the decryption will fail.

LV11

The same idea can be applied for the LV11 scheme, but due to the more complex nature of this scheme, we need to modify all algorithms. G and G_T are two groups of prime order, with the bilinear mapping $e : G \times G \to G_T$. The global parameters are $params = (g,\ u,\ v,\ Sig)$ where g, u and v are random generators in G. $Sig = (SigGen,\ Sign,\ Verify)$ is a strongly unforgeable OTS. The two users, A and B, have one key-pair each; $(pk_A,\ sk_A) = (g^a,\ a)$ and $(pk_B,\ sk_B) = (g^b,\ b)$ respectively. The original algorithms is defined as

Encrypt$_1$ $(params,\ pk_A,\ m)$: (8)

$g,\ u,\ v \leftarrow params$
$ssk,\ svk \leftarrow SigGen$
$r,\ t \xleftarrow{R} \mathbb{Z}_q^*$
$c_1 = svk,\ c_3 = e(g,\ g)^r \cdot m,\ c_4 = (u^{svk} \cdot v)^r$
$c'_2 = pk_A^t,\ c''_2 = g^{1/t},\ c'''_2 = pk_A^{rt}$
$\sigma = Sign(ssk,\ (c_3,\ c_4))$
$c = (c_1,\ c'_2,\ c''_2,\ c'''_2,\ c_3,\ c_4,\ \sigma)$
Return: c

Encrypt$_2$ $(params,\ pk_A,\ m)$: (9)

$g,\ u,\ v \leftarrow params$
$ssk,\ svk \leftarrow SigGen$
$r \xleftarrow{R} \mathbb{Z}_q^*$
$c_1 = svk,\ c_2 = g^{pk_A \cdot r},\ c_3 = e(g,\ g)^r \cdot m,\ c_4 = (u^{svk} \cdot v)^r$
$\sigma = Sign(ssk,\ (c_3,\ c_4))$
$c = (c_1,\ c_2,\ c_3,\ c_4,\ \sigma)$
Return: c

ReKeygen$(params,\ sk_A,\ pk_B)$: (10)
$rk_{A\to B} = pk_B^{1/sk_A}$
Return: $rk_{A\to B}$

Re-encrypt$(params,\ rk_{A\to B},\ c,\ pk_A)$: (11)
$u,\ v \leftarrow params$
$c_1,\ c_2,\ c_3,\ c_4,\ \sigma \leftarrow c$
if $e(c_2,\ u^{c_1}\cdot v) \neq e(pk_A,\ c_4)$: Return invalid
if $Verify(c_1,\ \sigma,\ (c_3,\ c_4)) \neq 1$: Return invalid
$t \xleftarrow{R} \mathbb{Z}_q^*$
$c_2' = pk_A^t,\ c_2'' = rk_{A\to B}^{1/t},\ c_2''' = c_2^t$
$c' = (c_1,\ c_2',\ c_2'',\ c_2''',\ c_3,\ c_4,\ \sigma)$
Return: c'

Decrypt$_1$ $(params,\ sk_B,\ pk_B,\ c')$: (12)
$g,\ u,\ v \leftarrow params$
$c_1,\ c_2',\ c_2'',\ c_2''',\ c_3,\ c_4,\ \sigma \leftarrow c$
if $e(c_2',\ c_2'') \neq e(pk_B,\ g)$: Return invalid
if $e(c_2''',\ u^{c_1}\cdot v) \neq e(c_2',\ c_4)$: Return invalid
$m = c_3/e(c_2'',\ c_2''')^{1/sk_B}$
Return: m

Decrypt$_2$ $(params,\ sk_A,\ pk_A,\ c)$: (13)
$g,\ u,\ v \leftarrow params$
$c_1,\ c_2,\ c_3,\ c_4 \leftarrow c$
if $e(c_2,\ u^{c_1}\cdot v) \neq e(pk_A,\ c_4)$: Return invalid
if $Verify(c_1,\ \sigma, (c_3,\ c_4)) \neq 1$: Return invalid
$m = c_3/e(c_2,\ g)^{1/sk_A}$
Return: m

These algorithms can be modified in the following way, to embed the time stamp and construct a new scheme with the ability to make temporary delegations

Encrypt'$_1$ $(params,\ pk_A,\ m)$: (14)
$g,\ u,\ v \leftarrow params$
$ssk,\ svk \leftarrow SigGen$
$r,\ t \xleftarrow{R} \mathbb{Z}_q^*$
$c_1 = svk,\ c_3 = e(g,g)^r\cdot m,\ c_4 = (u^{svk}\cdot v)^r,\ c_5 = g$
$c_2' = pk_A^t,\ c_2'' = g^{1/t},\ c_2''' = pk_A^{rt}$
$\sigma = Sign(ssk,\ (c_3,\ c_4))$
$c_{st} = (c_1,\ c_2',\ c_2'',\ c_2''',\ c_3,\ c_4,\ c_5,\ \sigma)$
Return: c_{st}

(15)

Encrypt'$_2$ $(params,\ sk_A,\ st,\ m)$:
$g,\ u,\ v \leftarrow params$
$ssk,\ svk \leftarrow SigGen$
$r \xleftarrow{R} \mathbb{Z}_q^*$
$c_1 = svk,\ c_2 = g^{sk_A \cdot r},\ c_3 = e(g,g)^{r \cdot st} \cdot m,\ c_4 = (u^{svk} \cdot v)^r,\ c_5 = g^{st}$
$\sigma = Sign(ssk,\ (c_3,\ c_4))$
$c_{st} = (c_1,\ c_2,\ c_3,\ c_4,\ c_5,\ \sigma)$
Return: c_{st}

(16)

ReKeygen'$(params,\ sk_A,\ pk_B,\ st)$:
$rk_{A \xrightarrow{st} B} = pk_B^{st/sk_A}$
Return: $rk_{A \xrightarrow{st} B}$

(17)

Re-encrypt'$(params,\ rk_{A \xrightarrow{st} B},\ c_{st},\ pk_A)$:
$u,\ v \leftarrow params$
$c_1,\ c_2,\ c_3,\ c_4,\ c_5,\ \sigma \leftarrow c$
if $e(c_2,\ u^{c_1} \cdot v) \neq e(pk_A,\ c_4)$: Return invalid
if $Verify(c_1,\ \sigma,\ (c_3,\ c_4)) \neq 1$: Return invalid
$t \xleftarrow{R} \mathbb{Z}_q^*$
$c_2' = pk_A^t,\ c_2'' = rk_{A \xrightarrow{st} B}^{1/t},\ c_2''' = c_2^t = pk_A^{r \cdot t}$
$c_{st}' = (c_1,\ c_2',\ c_2'',\ c_2''',\ c_3,\ c_4,\ c_5,\ \sigma)$
Return: c_{st}'

(18)

Decrypt'$_1$ $(params,\ sk_B,\ pk_B,\ c_{st}')$:
$g,\ u,\ v \leftarrow params$
$c_1,\ c_2',\ c_2'',\ c_2''',\ c_3,\ c_4,\ c_5,\ \sigma \leftarrow c_{st}'$
if $e(c_2',\ c_2'') \neq e(pk_B,\ c_5)$: Return invalid
if $e(c_2''',\ u^{c_1} \cdot v) \neq e(c_2',\ c_4)$: Return invalid
if $Verify(c_1,\ \sigma,\ (c_3,\ c_4)) \neq 1$: Return invalid
$m = c_3 / e(c_2'',\ c_2''')^{1/sk_B}$
Return: m

(19)

Decrypt'$_2$ $(params,\ sk_A,\ pk_A,\ c_{st})$:
$c_1,\ c_2,\ c_3,\ c_4,\ c_5 \leftarrow c_{st}$
if $e(c_2,\ u^{c_1} \cdot v) \neq e(pk_A,\ c_4)$: Return invalid
if $Verify(c_1,\ \sigma,\ (c_3,\ c_4)) \neq 1$: Return invalid
$m = c_3 / e(c_2,\ c_5)^{1/sk_A}$
Return: m

We will refer to this scheme as LV11 Temp. v1. One extra component, c_5, had to be added to the ciphertext, so that the verification of the ciphertext's well-formedness would remain intact. We verify the correctness of these modifications by again, decrypting a level 2 encrypted message and a previously re-encrypted message:

$$
\begin{aligned}
m &= Decrypt'_2(sk_A,\ c_{st},\ st) \\
&= Decrypt'_2(a,\ Encrypt'_2(pk_A,\ m,\ st),\ st) \\
&= Decrypt'_2(a,\ (svk,\ g^{a\cdot r},\ e(g,g)^{r\cdot st}\cdot m,\ (u^{svk}\cdot v)^r,\ \sigma),\ st) \\
&= \frac{e(g,g)^{r\cdot st}\cdot m}{e(g^{a\cdot r\cdot st},g)^{\frac{1}{a}}} \\
&= \frac{e(g,g)^{r\cdot st}\cdot m}{e(g,g)^{\frac{a\cdot r\cdot st}{a}}} \\
&= \frac{e(g,g)^{r\cdot st}\cdot m}{e(g,g)^{r\cdot st}} \\
&= m
\end{aligned}
$$

$$
\begin{aligned}
m &= Decrypt'_1(sk_B,\ c'_{st}) \\
&= Decrypt'_1(b,\ ReEncrypt'(rk_{A\xrightarrow{st}B},\ c_{st})) \\
&= Decrypt'_1(b,\ ReEncrypt'(ReKeygen'(sk_A,\ pk_B,\ st),\ Encrypt'_2(pk_A,\ m,\ st))) \\
&= Decrypt'_1(b,\ ReEncrypt'(g^{\frac{b\cdot st}{a}},\ (svk,\ g^{a\cdot r},\ e(g,g)^{r\cdot st}\cdot m,\ (u^{svk}\cdot v)^r,\ \sigma))) \\
&= Decrypt'_1(b,\ (svk,\ g^{a\cdot t},\ g^{\frac{b\cdot st}{a\cdot t}},\ g^{a\cdot r\cdot t},\ e(g,g)^r\cdot m,\ (u^{svk}\cdot v)^r,\ \sigma))) \\
&= \frac{e(g,g)^{r\cdot st}\cdot m}{e(g^{\frac{b\cdot st}{a\cdot t}},g^{a\cdot r\cdot t})^{\frac{1}{b}}} \\
&= \frac{e(g,g)^{r\cdot st}\cdot m}{e(g,g)^{\frac{b\cdot st\cdot a\cdot r\cdot t}{b\cdot a\cdot t}}} \\
&= \frac{e(g,g)^{r\cdot st}\cdot m}{e(g,g)^{st\cdot r}} \\
&= m
\end{aligned}
$$

The above constructions do accomplish the goal of putting an expiration date on re-encryption keys. In fact, the solution is of course not restricted to be used with time stamps to achieve expiration dates. Arbitrary data can be given to the SignTimestamp function, and the encrypted data and re-encryption key would need to have matching embedded data, which would allow for any type of categorisation, not only by time.

However, the solution decreases the scheme's flexibility due to the need of secret information during level 2 encryption. Algorithm 5 and Algorithm 15 take the parameter $st = t^{a_1}$, which clearly needs access to the secret value a_1, when constructed. In the general case, where we need other parties than the customer itself to encrypt messages under the customers public key, this imposes a problem. The customer does now need to distribute signed time stamps to whoever is doing the encryption. Although, as mentioned before, multiple signed time stamps can of course be created and distributed in advance, perhaps at system setup, so that no interaction is required during runtime.

6.3.2 A second attempt

The introduction of time stamps into the LV11 and AFGH06 schemes results in an unwanted issue, each time stamp must be generated and distributed to the point where data will be encrypted. The aim of the second attempt is to remove the need to distribute time stamps, these should be possible to generate at an arbitrary point without the need of secret data.

In [14], Libert and Vergnaud published a new version of their previous article [15]. In this version, they introduced a scheme with temporary delegation, that do not require any secret information during the level 2 encryption operation. Basically, the time information t is protected by randomising the function $F_A(t) = g^t \cdot g^{a_2}$, where g^{a_2} is the second element of user A's public key. This gives us the term $(g^t \cdot g^{a_2})^r = g^{(t+a_2)r}$. We will call this scheme LV11 Temp. v2.

We can use the same approach to make another modification of AFGH06, which we will call AFGH06 Temp. v2. Instead of providing a signed time stamp as input to Algorithm 5, a time stamp (encoded into $\mathbb{Z}_q$) is directly given as input to the algorithm, and the second part of the user's public key is used to define the same function $F_a(t)$ as above. Compared to the original AFGH06, a second level ciphertext looks like $(g^{(a_2+t)^r},\ m \cdot Z^{a_1 \cdot r})$ instead of $(g^r,\ m \cdot Z^{a_1 \cdot r})$, and the level 2 encryption algorithm is modified as

(20)

Encrypt$_2$"$(params,\ pk_A,\ m,\ t)$:
$g \leftarrow params$
$Z^{a_1},\ g^{a_2} \leftarrow pk_A$
$r \xleftarrow{R} \mathbb{Z}_q^*$
$F_a(t) = g^{a_2} \cdot g^t = g^{(a_2+t)}$
Return: $c_t = (F_a(t)^r,\ m \cdot Z^{a_1 \cdot r})$

As we can see, no secret information is required by the algorithm. The re-encryption key generation algorithm is altered similarly, by dividing the exponent in the re-encryption key by $a_2 + t$, so that these cancel each other out at the re-encryption step.

(21)

ReKeygen"$(params,\ sk_A,\ pk_B,\ t)$:
$a_1,\ a_2 \leftarrow sk_A$
$g^{b_2} \leftarrow pk_B$
$rk_{A \xrightarrow{t} B} = g^{b_2 \cdot a_1/(a_2+t)}$
Return: $rk_{A \xrightarrow{t} B}$

Lastly, the level 2 decryption algorithm is modified to

(22)

Decrypt"$_2$ $(params,\ sk_A,\ c_t,\ t)$:
$g^{(a_2+t)r},\ m \cdot Z^{a_1 \cdot r} \leftarrow c$
$a_2 \leftarrow sk_A$
$m = m \cdot Z^{a_1 \cdot r} / e(g^{(a_2+t)r},\ g^{a_1/(a_2+t)})$
Return: m

The correctness of the modifications can be verified by decrypting a level 2 ciphertext and a previously re-encrypted ciphertext

$$
\begin{aligned}
m &= Decrypt_2''(sk_A,\ c_t,\ t) \\
&= Decrypt_2''((a_1,\ a_2),\ Encrypt_2''(pk_A,\ m,\ t),\ t) \\
&= Decrypt_2''((a_1,\ a_2),\ (g^{(a_2+t)\cdot r},\ m \cdot Z^{a_1 \cdot r}),\ t) \\
&= \frac{m \cdot Z^{a_1 \cdot r}}{e(g^{(a_2+t)\cdot r},\ g^{\frac{a_1}{a_2+t}})} \\
&= \frac{m \cdot Z^{a_1 \cdot r}}{Z^{a_1 \cdot r}} \\
&= m
\end{aligned}
$$

$$
\begin{aligned}
m &= Decrypt_1(sk_B,\ c_{b,t}) \\
&= Decrypt_1(b_2,\ ReEncrypt(rk_{A \xrightarrow{t} B},\ c_{a,t})) \\
&= Decrypt_1(b_2,\ ReEncrypt(ReKeygen''(sk_A,\ pk_B,\ t),\ Encrypt_2''(pk_A,\ m,\ t))) \\
&= Decrypt_1(b_2,\ ReEncrypt(g^{\frac{b_2 \cdot a_1}{a_2+t}},\ (g^{(a_2+t)\cdot r},\ m \cdot Z^{a_1 \cdot r}))) \\
&= Decrypt_1(b_2,\ (e(g^{\frac{b_2 \cdot a_1}{(a_2+t)}}, g^{(a_2+t)\cdot r}),\ m \cdot Z^{a_1 \cdot r})) \\
&= Decrypt_1(b_2,\ Z^{b_2 \cdot a_1 \cdot r},\ m \cdot Z^{a_1 \cdot r}) \\
&= \frac{m \cdot Z^{a_1 \cdot r}}{Z^{\frac{b_2 \cdot a_1 \cdot r}{b_2}}} \\
&= \frac{m \cdot Z^{a_1 \cdot r}}{Z^{a_1 \cdot r}} \\
&= m
\end{aligned}
$$

Security proof

This version of AFGH06 looks promising. We extend the security proof of theorem 3.1 in [3], to verify that the security is still intact. The proofs are copied directly, possibly with some minor differences in notations, and with our own extensions added in bold text.

- **Standard security.**

  1. On input $(g,\ g^a,\ g^b,\ g^c,\ Z^{bc^2},\ Z^d)$, the simulator sets $y = g^c$, $W = e(y,\ y)$ and obtains the tuple $(y = g^c,\ y^\alpha = g^a,\ y^\beta = g^b,\ y^\gamma = g,\ W^{\beta/\gamma} = Z^{bc^2},\ W^{\alpha\beta/\gamma} = Z^d)$ for $\alpha = a\frac{a}{c},\ \beta = \frac{b}{c},\ \gamma = \frac{1}{c}$ The global parameters $params = (y,\ W)$ and the target users public key $pk_T = (W^{\beta/\gamma},\ g^t)$ are sent to A.
  2. **For $j = 1$ up to poly(k) time periods, A can perform the following:**
     (a) For i=1 up to poly(k), A can request the following:

i. a delegation from T to an honest party **for the time period $\boldsymbol{l_j}$**. S randomly selects $(r_{(i,1)},\ r_{(i,2)}) \xleftarrow{R} \mathbb{Z}_q$ and sets $rk_{T\to i,\boldsymbol{l_j}} = y^{\gamma r(i,2)/(\boldsymbol{t+l_j})}$ and $pk_i = (W^{r_{(i,1)}},\ y^{\beta r(i,2)})$. $rk_{T\to i,\boldsymbol{l_j}}$ and pk_i are sent to A.

ii. a delegation to T from an honest party, **for the time period $\boldsymbol{l_j}$**. The simulator uses either the recorded value $r_{(i,1)}$ or generates a new random values for the party i and sets $rk_{i\to T,\boldsymbol{l_j}} = (g^t)^{r(i,1)/(\boldsymbol{r_{(i,2)}+l_j})}$. pk_i and $rk_{i\to T,\boldsymbol{l_j}}$ are sent to A.

iii. a delegation to T from a party corrupted by A, **for the time period $\boldsymbol{l_j}$**. A can generate these internally by running $(pk_i,\ sk_i) \leftarrow KeyGen$ and computing $rk_{i\to T,\boldsymbol{l_j}} = (g^t)^{i_1/(\boldsymbol{i_2+l_j})}$.

3. Eventually, **during the last time period $\boldsymbol{j}$**, A must output a challenge $(m_0,\ m_1,\ \tau)$ where $m_0 \neq m_1$ and τ is its internal state information. The simulator randomly selects $b \in \{1, 0\}$ and computes the ciphertext $c_b = (y^{\alpha(\boldsymbol{t+l_j})},\ m_b W^{\alpha\beta/\gamma})$. S sends $(c_b,\ \tau)$ to A and waits for A to output $b' \in \{0, 1\}$.
4. If $b = b'$, then S guesses "$d = abc$", otherwise S guesses "$d \neq abc$"

If $d = abc$ then this is a perfect simulation. If $d \neq abc$ then m_b is information-theoretically hidden from A, since $W^{\alpha\beta/\gamma} = (Z^d)$ was chosen independently from $y^\alpha = (g^a)$. Thus, if A succeeds with probability $\frac{1}{2} + \epsilon$, then S succeeds with probability $\frac{1}{2} + \frac{\epsilon}{2}$. This contradicts the extended decision bilinear Diffie-Hellman (eDBDH) assumption. **The additional term in the delegation keys and the cipher text has no effect on S's chance of success, it still contradicts the eDBDH assumption.**

- **Master secret security.**

1. On input $(g,\ g^a)$ in G_1, output the global parameters $(g,\ Z)$ and the target public key $pk_t = (Z^a,\ g^{t_2})$, where $Z^a = e(g,\ g^a)$ and t_2 is a random element in $\mathbb{Z}_q$.
2. **For $\boldsymbol{j = 1}$ up to poly($\boldsymbol{k}$) time periods, A can perform the following:**
   (a) For $i = 1$ up to poly(k), A can request:
      i. a delegation from T to a party corrupted by A, **for the time period $\boldsymbol{l_j}$**. S randomly selects $(r_{(i,1)},\ r_{(i,2)}) \xleftarrow{R} \mathbb{Z}_q$, sets $rk_{T\to i,\boldsymbol{l_j}} = g^{ar_{(i,2)}/(\boldsymbol{t_2+l_j})}$, $pk_i = (W^{r_{(i,1)}},\ g^{r_{(i,2)}})$, and $sk_i = (r_{(i,1)},\ r_{(i,2)})$, and sends $(pk_i,\ sk_i,\ rk_{T\to i,\boldsymbol{l_j}})$ to A.
      ii. a delegation to T from a party corrupted by A, **for the time period $\boldsymbol{l_j}$**. A can generate these delegations internally by running $(pk_i,\ sk_i) \leftarrow KeyGen()$ and computing $rk_{i\to T,\boldsymbol{l_j}} = (g^{t_2})^{i_1/(\boldsymbol{i_2+l_j})}$
      iii. Eventually, A must output a purported for T of the form (α, β). The simulator returns the value α.

The simulation is perfect, thus A must not be able to recover the master secret key of T, despite accepting and providing numerous delegations to T, because otherwise, S can efficiently solve the DLP in G_1. **The additional term in the delegation keys has no effect on S's chance of success, it still contradicts the discrete logarithm assumption.**"

6.4 Pre-processing of group element exponentiations

In almost all the scheme algorithms there are exponentiations involved and they are a relatively costly operation to perform. As one can see in AFGH06 and LV11, group elements in the global parameters and public keys are repeatedly exponentiated in the scheme algorithms. Therefore, there may be of interest to pre-process exponentiations of these group elements. The Charm framework [1] provides a feature to do this, and the affect this have on the benchmarking results can be seen in Chapter 7. This feature does of course require extra memory and this should definitely be taken under consideration if the hardware running some part of the system have memory constraints or if the number of elements that are pre-processed can become very large. To get an idea about how much memory that are required for storing the pre-processed information we look at the source code for the PBC library [16].

For an element $e \in G$ with order q and an integer k (default $k = 5$), the pre-processing information for element e will consist of a table with the size $(\left\lfloor \frac{\log_2 q}{k} \right\rfloor + 1) \cdot 2^k \cdot element\_representation\_size$. For example, using the curve called "SS512" in the Charm library, the group G_1 will have order 2^{159}. The element representation size is 16 bytes, and so the space required is $(\left\lfloor \frac{159}{5} \right\rfloor + 1) \cdot 2^5 \cdot 16$ bytes $= 1024$ bytes. This is a relatively small amount of data and should not impose any problems when pre-processing a pre-determined amount of group elements. But, if for example the proxy wants to pre-process re-encryption keys, where the number of keys can grow, it could be of interest to take the extra memory requirements needed for the pre-processing into consideration.

6.5 Proof of concept

In order to show how these concepts can be used in practice, a prototype for streaming video through proxy re-encryption was built.

6.5.1 GStreamer

GStreamer is a library and framework for constructing software that processes multimedia [12]. The framework is built with modularity in mind and the basic building blocks when creating software that utilises GStreamer is called elements. Elements take incoming data, processes the data and outputs the result. These elements are chained together to form pipelines that processes the media data that is fed into it. The first and last elements of the pipeline are typically called sources and sinks. For example, the source could come from a video file on a disk, and the sink could be a video player that displays the video on a monitor.

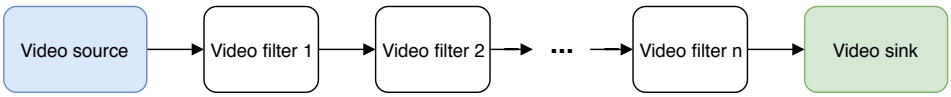

Figure 6.1: Video flows from a video source through any number of filters that do some sort of processing and finally into a video sink.

6.5.2 Design

In the described system in Chapter 3, video is produced inside the customer's domain, it is stored and re-encrypted in the proxy's domain and then finally downloaded and decrypted in the partner's domain. This means that all three PRE operations are executed independently in different domains. To simulate this environment, three different programs were written; `encryptor`, `re-encryptor` and `decryptor`. The `encryptor` encrypts some video source and produces a file with the result, which represents the data stored at the proxy. The proxy can then run `re-encryptor` with the encrypted data as input and produce a new file with the re-encrypted data. This is the data that should be sent to the partner. The partner can run `decryptor`, to decrypt the data and recover the original video stream.

The actual code that does the PRE operations was implemented as an GStreamer element, which means that it is not in any way specific to the above programs, but can be used in any GStreamer pipeline. The element performs the PRE operations by calling the appropriate functions in the instantiated scheme from the Charm library. In this prototype, the LV11 Temp. v2 scheme is used, but it can easily be exchanged to another PRE scheme as they all follow the same interface. The element encrypts any incoming data by calling the level 2 encryption algorithm and writing the resulting encrypted data as output, prepended by the total size of the encrypted data chunk. The need to prepend each encrypted chunk of data with its size, comes from that when the proxy is going to read the encrypted data, in order to re-encrypt it, there is no telling exactly how much data will have been buffered each time the element is asked to perform re-encryption. Therefore, by first reading the size of encrypted data chunk, the element can wait until it has accumulated an amount of data, equal to the size of the data chunk, before it performs the re-encryption. The same process is done when decrypting. Figure 6.2 illustrate how data flows through the element during the different PRE operations.

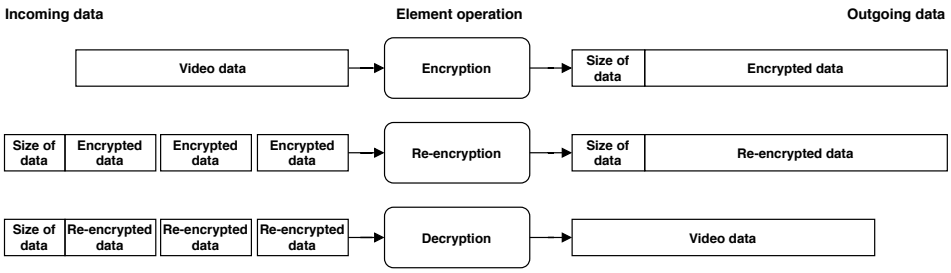

Figure 6.2: How the element performs the PRE operations on the video data.

Chapter 7
Performance results

This chapter contains all data collected from testing the different schemes from Chapter 6. To evaluate their performance, two different type of benchmarks were performed. The first measured the schemes performances in terms of execution time and the other in terms of size. The same elliptic curve was used for each scheme, which was a super-singular curve with a 512-bit base field, referred to as "SS512" in the Charm library.

7.1 Execution time

To evaluate the execution time, a small benchmarking program was written using the Python language. It uses a built-in benchmarking functionality in the Charm library that is able to measure the execution time of specific operations. What is worth noting is that it only measure the mathematical operations that are performed in the native C modules, which means that the results should not be influenced by the overhead of the relatively slow Python language. For each scheme, the benchmarking program ran each algorithm 500 times, measured the execution times, and the reported the mean value.

7.1.1 Hardware used

Even though the main purpose of the results is to compare the different schemes with each other, it may also be of interest to know which hardware benchmarking was performed on. This if some operations are to be run on hardware with limited performance and we want to know roughly what results we may expect from such a device. The hardware specifications used for our benchmarking were the following:

CPU: Intel i7-7700K @ 4.20 GHz

RAM: 16 GB @ 2400 MHz

7.1.2 Proxy re-encryption schemes

This subsection contains all data collected from the test runs performed on the PRE schemes, which in turn can be divided into implementations with and without pre-processed group elements.

Without pre-processed group elements

Table 7.1 and Table 7.2 shows the execution time of each algorithm in the PRE schemes, *without* the use of pre-processed group elements.

Table 7.1: Benchmark of AFGH06 and its temporary delegation versions, without pre-processed group elements.

	CPU Time (μs)		
Operation	AFGH06	AFGH06 Temp. v1	AFGH06 Temp. v2
Encrypt$_1$	877.8	777.4	781.6
Encrypt$_2$	1106.8	1069.0	2390.9
Re-keygen	965.7	970.9	953.9
Re-encrypt	589.3	592.1	589.1
Decrypt$_1$	86.5	84.4	86.4
Decrypt$_2$	1620.7	1583.4	1563.8

Table 7.2: Benchmark of LV11 and its temporary delegation versions, without pre-processed group elements.

	CPU Time (μs)		
Operation	LV11	LV11 Temp. v1	LV11 Temp. v2
Encrypt$_1$	5857.2	5737.4	10037.9
Encrypt$_2$	3797.6	4744.4	5771.7
Re-keygen	977.2	970.1	3903.5
Re-encrypt	4315.9	4337.4	9626.6
Decrypt$_1$	3328.3	3416.8	5735.4
Decrypt$_2$	2064.4	2064.8	919.6

With pre-processed group elements

Table 7.3 and Table 7.4 shows the execution time of each algorithm in the PRE schemes, *with* the use of pre-processed group elements.

Table 7.3: Benchmark of AFGH06 and its temporary delegation versions, with pre-processed group elements.

	CPU Time (μs)		
Operation	AFGH06	AFGH06 Temp. v1	AFGH06 Temp. v2
Encrypt$_1$	723.3	741.7	718.9
Encrypt$_2$	176.9	219.7	323.1
Re-keygen	135.3	135.7	142.9
Re-encrypt	591.8	591.2	592.9
Decrypt$_1$	83.9	86.1	84.9
Decrypt$_2$	745.9	737.1	752.5

Table 7.4: Benchmark of LV11 and its temporary delegation versions, with pre-processed group elements.

	CPU Time (μs)		
Operation	LV11	LV11 Temp. v1	LV11 Temp. v2
Encrypt$_1$	3248.1	3225.8	6289.9
Encrypt$_2$	2921.8	3074.4	4032.3
Re-keygen	4893.8	4987.9	10970.7
Re-encrypt	2669.8	2743.7	6202.7
Decrypt$_1$	3316.9	3341.2	5823.8
Decrypt$_2$	2066.3	2110.0	889.0

7.1.3 Attribute-based encryption schemes

To get a comparable result between PRE and ABE, only one attribute was included in the access structure used to encrypt the ciphertext, as well as only one attribute in the generated encryption key. This attribute represents the time stamp used in PRE so that the two alternatives have comparable properties. This subsection contains data collected from test runs of ABE, which in turn can be divided into implementations with and without pre-processed of group elements.

Without pre-processed group elements

Table 7.5 shows the execution time of each algorithm in the ABE schemes, *without* the use of pre-processed group elements.

Table 7.5: Benchmark of W11 and W11-CCA2 without pre-processed group elements.

	CPU Time (μs)	
Operation	W11	W11-CCA2
Keygen	5125.0	5123.7
Encrypt	6968.8	9487.2
Decrypt	3062.5	3226.5

With pre-processed group elements

Table 7.6 shows the execution time of each algorithm in the ABE schemes, *with* the use of pre-processed group elements.

Table 7.6: Benchmark of W11 and W11-CCA2 with pre-processed group elements.

	CPU Time (μs)	
Operation	W11	W11-CCA2
Keygen	3437.5	3457.1
Encrypt	4437.5	6891.6
Decrypt	3062.5	3243.5

7.2 Object sizes

When using any of the schemes, they are going to require some extra storage for the different objects. Ciphertexts are going to be larger and in PRE, the proxy will potentially be storing a very large number of re-encryption keys. In ABE, it is instead the user that will store a potentially large number of decryption keys. Table 7.7 and Table 7.8 shows the additional space needed by the PRE schemes in ciphertexts and re-encryption keys.

Table 7.7: Size of objects that needs to be stored in a large amount using AFGH06 and its temporary delegation versions.

	Size (bytes)		
Object	AFGH06	AFGH06 Temp. v1	AFGH06 Temp. v2
Re-encryption key	90	90	90
Level 1 ciphertext	348	348	348
Level 2 ciphertext	264	264	264

Table 7.8: Size of objects that needs to be stored in a large amount using LV11 and its temporary delegation versions.

	Size (bytes)		
Object	LV11	LV11 Temp. v1	LV11 Temp. v2
Re-encryption key	90	90	180
Level 1 ciphertext	630	630	990
Level 2 ciphertext	450	450	630

Table 7.9 shows the additional space needed by the ABE schemes in ciphertexts and decryption keys.

Table 7.9: Size of objects that needs to be stored in a large amount using W11 and W11-CCA2.

	Size (bytes)	
Object	W11	W11-CCA2
Decryption key	270	270
Ciphertext	444	540

Chapter 8

Discussion

In this chapter, we discuss the different aspects of PRE and ABE, as well as the results and insights gained during the work on this thesis.

8.1 Applications and properties

Although it is possible to accomplish our goal with both PRE and ABE, the two techniques will result in a system with very different properties. One of the main differences is where the actual delegation transformation of the data takes place, meaning, where are the keys containing the information about whether a partner has decryption right to certain data stored and managed, as well as where the ciphertexts are decrypted/re-encrypted by these keys. Using PRE, the transformation takes place at the proxy, when a ciphertext encrypted under a customer's public key is re-encrypted to a ciphertext encrypted under a partner's public key, given that a valid re-encryption key exists of course. The computational cost of doing this is paid by the proxy. Using ABE on the other hand, the same transformation is in a sense performed directly at the partner when decrypting the ciphertext, given that they have a valid decryption key. In this case, the partner pays the computational cost.

The task of doing the transformation does not only require a higher computational cost, but it also requires a more complex system. It requires the ability to receive new decryption/re-encryption keys and store them safely, mapping the keys to the right data packages and so on. There could be a problem with doing this at the partner side, if it for example should be implemented by some software developed by another party than the storage service provider. In other words, it is more difficult for third-party applications to provide support for the service if it requires complex implementations.

Another important difference between the two is that, while we have PRE schemes that can "attribute" ciphertexts and re-encryption keys with for example time stamps, ABE has the possibility to make complex access policies. This opens up for more possible use cases where decryption privileges can be delegated based on more than just a single attribute as in PRE.

8.2 Security

The nature of PRE schemes make it fundamentally hard to achieve IND-CCA2. Two ciphertexts, where the second one is a re-encryption of the first on, are both well-formed ciphertexts. But they both decrypt to the same plaintext which violates the rules of CCA2-security. However, Canetti, Krawczyk and Nielsen defined a slightly relaxed variant of CCA2 in [8] which they called replayable chosen ciphertext attack (RCCA). This variant essentially the same as CCA2, except that it allows an attacker to generate new ciphertexts that decrypts to the same plaintext as another given ciphertext. Canetti, Krawczyk and Nielsen also finds that for most practical applications it is sufficient for a scheme to achieve IND-RCCA instead of IND-CCA2. Because of this, we consider the LV11 scheme [14], which achieves IND-RCCA, to have a high level of security even though it cannot be given the IND-CCA2 label.

Attribute based encryption does not have same inherent properties that make it hard to achieve IND-CCA2 as PRE. Here we can more easily achieve the full notion of IND-CCA2, which of course is desirable. However, PRE and ABE work very different in practice and have a lot of different properties, which we consider being of more gravity when it comes to deciding which one to use, especially when comparing IND-RCCA and IND-CCA2 since they are very close to each other.

The PRE scheme AFGH06 [3] lacks the protection against CCA that the schemes above have, but in return, its construction is much simpler. This also means that it is faster than the others. Even though it is highly desirable to reach CCA security, this scheme could still be of interest if the performance is of special importance. Maybe CCA secure alternatives are just too slow to be practical in certain cases. However, in the end the scheme *is* vulnerable against CCA. A CCA means that the attacker has some sort of possibility to retrieve information from re-encryptions or decryptions of ciphertexts, of the attackers choosing. One could take measures to minimize the chance of such attacks being possible in the system, but never be sure that there are not any. If the data being protected is sensitive, which it in most cases are, this scheme should not be used. It has however served a good purpose during our work due to its simpler construction which made it easy to work with and understand. Progress from work with this scheme did help us while working with the other schemes.

8.2.1 Key revocation

A crucial part of access control is access revocation. A data owner may at any time want to revoke access privileges from certain parties. As mentioned in Section 4.3 and Section 4.4 the drawbacks of both ABE and PRE is the lack of efficient ways to revoke existing keys. The use of time stamps in keys and ciphertexts achieves some sort of lazy revocation. Once a partner or the proxy no longer have a key with time stamps matching the current time period, the partner has in some sense been revoked. The fact that an old key is still valid for older data, which do have a matching time stamp, should however not be a problem. After all the partner did have access to that data and may as well have copied and stored the data elsewhere during that time period. Therefore, it makes no difference that old keys

are still valid even though the partner is not being delegated any new keys.

The PRE implementations, *a first attempt* and *a second attempt*, presented in Section 6.3, are two approaches of achieving lazy revocation of keys. Both approaches embed time data into ciphertexts and re-encryption keys. The first attempt makes use of signed time stamps. These must be produced at the location where the private key of the owner exists, which consequently results in a distribution problem, as these signed time stamps must be distributed to all points where encryption occurs. If more fine-grained time periods are used, this operation must be performed more frequently. In [14], Libert et al. present a version of their own LV11 scheme, which uses a technique to generate time stamps using only the data owner's public key. The second attempt borrows this technique to modify the AFGH06 scheme, to a version that also has this property. By doing this we overcome the distribution problem in the first attempt and thus allowing for more fine-grained time stamps without having to first distribute these.

If the owner of data can trust the cloud service provider to perform actions correctly on demand, such as destroying re-encryption keys, PRE can achieve instant revocation, since data can no longer be re-encrypted if the re-encryption key no longer exists. This would of course require partial trust for the service provider. Regarding ABE, it is only possible to achieve lazy revocation, at least in our design. It could be possible that one can achieve instant revocation using non-monotone access structures, by expressing that revoked partners do *not* qualify to decrypt. However, the investigation in this property is outside the scope of this thesis.

8.3 Performance

The performance is another important factor because our system should support streaming of data without adding unacceptable delays. Let us initially look at the differences between PRE and ABE by comparing the content of Figure 5.2 and Figure 5.3 as this will give a good picture of the differences between the two cryptosystems. PRE requires more steps to distribute data compared to ABE, but depending on the individual schemes and settings, one cannot say that ABE would perform better. Naturally, some operations are executed more frequently, some have direct impact on the performance, while others should have a very low or no impact. The operations $Setup$ and $Keygen$ in both PRE and ABE as well as $ReKeygen$ in PRE, are only performed during setup and when new delegations are issued and should thus have no direct impact on performance. The time critical operations are primarily $Encrypt_1$, $Encrypt_2$, $ReEncrypt$, $Decrypt_1$, and $Decrypt_2$ for PRE, and $Encrypt$ and $Decrypt$ for ABE, as these will be performed for each data packet.

The benchmarking results found in Table 7.1 and Table 7.2 displays the differences between the two PRE schemes AFGH06 and LV11 and as viewed there is a noticeable variation in time needed for the same operations in the both schemes, AFGH06 is considerably faster in the performance-affecting operations as they in general require less complex computations (at the cost of security) than LV11. So, we can within a type of cryptosystem see vast differences depending on construc-

tion and implementation. The benchmarks performed for ABE contained only one attribute, a time stamp, so that the comparison between the two cryptosystems would be balanced. The down-part of this is that ABE is that it is not utilised as it is meant to be, thus the benchmark should be seen as a best-case scenario.

ABE on the other hand have fewer operations needed to perform the same objective as PRE and if we look at the values in Table 7.2 (LV11, original, without pre-computations) for LV11 and compare these with the values in Table 7.5 for W11 [22] (W11, original, without pre-computations) we can see that encryption-decryption from end-to-end is more or less the same. However, the advantages of ABE are also its disadvantages. As access policies and attribute lists grow larger so does the computational time needed to encrypt and decrypt the material, thus if fine-grained access control is a requirement then the consequences will be large ciphertexts and reduced performance. The penalty for adding more attributes vary between schemes and implementations, the increase is often linear with varying steepness depending on how the scheme is constructed. The consequence of this might be that a less specific access policy is used, which directly affects security in order to maintain high performance.

8.3.1 Symmetric to asymmetric key ratio

Asymmetric encryption may introduce delays as time critical data is streamed, an approach of mitigating this penalty would be by creating sessions where symmetric keys are re-used. This would mean that the operations of the asymmetric scheme would not have to be applied to each individual package but rather instead to a session of packages. The symmetric key now only has to be encrypted, (re-encrypted) and decrypted once for the session of data packages protected with the same symmetric key. If the time delays in a stream is too large, then the symmetric key generation ratio can be decreased. If there is margin for a lower ratio, then the frequency of new symmetric keys can be increased. However, the consequences of symmetric key sessions are decreased security and weaker access control. If a symmetric key is compromised more data is exposed than the standard scenario of one unique symmetric key per data package. Also, decryption delegation would include all data in a session instead of single packages. Ultimately the ratio of symmetric keys is a trade-off between security and performance, the good part in this is that it is a very flexible value that can be changed at a moment's notice.

8.4 Financial considerations

All computational operations performed, and memory used to store data are financial expenses, it is thus in the interest of both cloud storage provider and customer to keep these down to a necessary minimum. The use of ABE schemes may result in larger amounts of storage utilised compared to PRE as ciphertexts grow in size with access policies and so will CPU time. However, if attributes are used sparsely in ABE, the load on the system is equivalent to when PRE is used. The benchmarks performed for ABE contained only one attribute, a time stamp, so that the comparison between the two cryptosystems would be balanced. The

down-part of this use of ABE is that it is not utilised as it is meant to be thus the benchmark should be seen as a best-case scenario.

The size of re-encryptions keys generated in a PRE scheme with a 512-bit curve for the various PRE schemes can be seen in Table 7.7 and Table 7.8. Consider the LV11 scheme with temporary delegation version 2. Then a re-encryption key takes 180 bytes to store. The number of keys that the proxy needs to store if one assumes that every customer continuously delegates decryption privileges to every partner can be calculated by $\#k = n_c \cdot n_p \cdot k_f \cdot t$ where n_c is the number of customers, n_p is the number of partners, k_f is the key frequency and t is the duration a key is stored. Let us exaggerate and say that we have 10,000 customers, 100 partners, the key frequency is 365 keys per year (a key is valid for 1 day) and we store keys for 1 year. Then the amount of storage required would be $\frac{180 \text{ bytes}}{\text{key}} \cdot 36 \cdot 10^9$ keys = 65.7 GB, which with the storage price of today is negligible.

The storage required for the additional cryptographic data that is appended to ciphertexts when using any of the schemes it is also a minor problem. This does of course depend on the type of data that is stored, and more specifically, how much data that is encrypted each time. In our case this is video data and according to Axis Site Designer, a tool that can show an estimated bandwidth for a camera with a certain set of settings, a modest camera recording at the resolution 800x600 with 5 frames per second, the bandwidth is estimated to be 325 kb/s. Let us again assume we are using the LV11 scheme with temporary delegation version 2 and that we are storing level 1 ciphertexts. This requires an additional 990 bytes for each ciphertext. The average part of the ciphertext size that that belongs to the additional cryptographic data is then $\frac{990 \text{ bytes}}{990 + \frac{325 \cdot 10^3}{5} \text{ bytes}} = 1.5\%$ which is quite small in a very pessimistic setting.

So, what will most likely be the major expense is CPU time spent on performing the computationally heavy operations in the ABE or PRE scheme. ABE and PRE differ here on where these operations take place. PRE will place the heavy computational operations at the proxy. In fact, the load on the proxy will be linear to the amount of simultaneous re-encryption operations performed at a given moment. In ABE all cryptographic operations will be distributed between the encryptor and decryptor. ABE does not require the proxy to store anything but ciphertexts and this without any additional required operations, it is not participating in the cryptographic protection which of course becomes cheaper for the proxy.

Chapter 9
Conclusions

This master's thesis has investigated techniques to encrypt data before it is stored outside the owner's domain, inside cloud storage services, this while still assuring it is distributable. The study showed that there are two existing techniques which show promising characteristics; *proxy re-encryption*, and *attribute based encryption*. However, both techniques have drawbacks and limitations, whereof the common one is key revocation. In an attempt to mitigate the issue of revocation related to PRE, we developed what we call signed time stamps, which could be integrated into already existing PRE schemes. These time stamps assure re-encryption keys are only valid for ciphertexts with matching time stamps.

PRE can achieve a stronger level of key revocation than ABE if we can assume that the cloud storage provider can be trusted to destroy keys on demand. If this assumption cannot be made, PRE and ABE can be seen as somewhat equal in their limitation of key revocation which is that it can take up to the duration between two following time stamps before the revocation is effective.

Asymmetric encryption in general is very inefficient in encrypting/decrypting large amounts of data, as a way of mitigating this, hybrid encryption can and should be used. The end product of this thesis deploys hybrid encryption. First, data is encrypted using symmetric encryption with a randomly generated. Secondly, the randomly generated key is then encrypted with the public key of the asymmetric scheme.

The operations commonly used in PRE and ABE primitives; such as bilinear pairings, and exponentiations are rather expensive to perform and may thus result in performance issues if executed frequently. We introduce an approach in which random symmetric keys are reused within a session of data packages, this to reduce the amount of expensive operations performed by the asymmetric scheme. By using this along with hybrid encryption the performance penalties of PRE or ABE can be reduced arbitrarily, meaning that the size of sessions determines the reduction impact.

Question one: In a practical way, can data be protected so that it cannot be read by the storage provider, this while it is possible for the owner to grant access to other parties?

> Yes, granting access to encrypted data to third parties can be done rather easily with the use of either PRE or ABE.

Question two: Is it possible to not only grant access to data to other parties, but also efficiently revoke it in practice?

> Yes, but to a limited extent. In PRE it is possible to instantly revoke access *if* we can trust that the cloud storage provider discards a re-encryption key when requested. Otherwise, for both PRE and ABE, the revocation can take up to the duration between two following time stamps.

Question three: Can operations as encryption, (re-encryption,) and decryption be performed efficiently enough on large data volumes without introducing significant time delays?

> Yes, performance requirements can be met. Depending on requirements the time delay can be dynamically mitigated if a small decrease in security and access control is acceptable, this as a symmetric key can be reused within a session of data packages.

9.1 Future work

As our research mainly focused on achieving low-trust storage with multiple parties involved, interesting areas of study have been excluded by our scope (i.e. lack of time) and thus are there still a lot of aspects left to further investigate. As the posed system which this thesis focus on is relatively complex and contains multiple parties testing of the cryptosystems in a more realistic environment have been left as future work. We are curious to see how an actual implementation in a realistic scenario would behave and what issues that may arise. This may be issues related to performance as our testing were performed on only one machine and did not take issues with e.g. transport into account.

As we only have investigated cryptographic primitives based on bilinear pairing, alternative techniques to realise similar schemes have not been examined. Hence, it would be of interest to widen the group of primitives and evaluate if a better alternative exists to the bilinear schemes.

There exists some research where PRE and ABE are combined in a single scheme. It would be interesting to investigate this idea further to see if it possible to achieve a solution having both the advantages of PRE and ABE at the same time, i.e. the re-encryption of ciphertexts at the proxy and fine-grained access control by attributing ciphertexts and keys.

Key revocation was an important part in our research and we did investigate and test methods to perform it. It was successful in some way but not completely satisfying as the revocation is "lazy" as we have described earlier. A revocation process that revokes keys instantly would be highly desirable and with more time we would have investigated it further. A more specific property regarding key revocation that would be interesting to look further into is the use of non-monotone access structures in ABE. The use of non-monotone access structures could possibly allow for a more efficient key revocation process.

Bibliography

[1] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green and A. D. Rubin. "Charm: a framework for rapidly prototyping cryptosystems". In: *Journal of Cryptographic Engineering* 3.2 (2013), pp. 111–128. ISSN: 2190-8508. DOI: 10.1007/s13389-013-0057-3. URL: http://dx.doi.org/10.1007/s13389-013-0057-3.

[2] I. Anca and D. Yevgeniy. "Proxy Cryptography Revisited". In: *in Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2003.

[3] G. Ateniese, K. Fu, M. Green and S. Hohenberger. "Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage". In: *ACM Trans. Inf. Syst. Secur.* 9.1 (Feb. 2006), pp. 1–30. ISSN: 1094-9224. DOI: 10.1145/1127345.1127346. URL: http://doi.acm.org/10.1145/1127345.1127346.

[4] J. Bethencourt, A. Sahai and B. Waters. "Ciphertext-Policy Attribute-Based Encryption". In: *2007 IEEE Symposium on Security and Privacy (SP '07)*. May 2007, pp. 321–334. DOI: 10.1109/SP.2007.11.

[5] M. Blaze, G. Bleumer and M. Strauss. "Divertible protocols and atomic proxy cryptography". In: *Advances in Cryptology — EUROCRYPT'98*. Ed. by Kaisa Nyberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 127–144.

[6] G. Bleumer. "Random Oracle Model". In: *Encyclopedia of Cryptography and Security*. Ed. by H. C. A. van Tilborg and S. Jajodia. Boston, MA: Springer US, 2011, pp. 1027–1028. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_220. URL: https://doi.org/10.1007/978-1-4419-5906-5_220.

[7] R. Canetti, S. Halevi and J. Katz. "Chosen-Ciphertext Security from Identity-Based Encryption". In: *Advances in Cryptology - EUROCRYPT 2004*. Ed. by C. Cachin and J. L. Camenisch. Berlin, Heidelberg:

Springer Berlin Heidelberg, 2004, pp. 207–222. ISBN: 978-3-540-24676-3.

[8] R. Canetti, H. Krawczyk and J. Nielsen. "Relaxing Chosen-Ciphertext Security". In: *Advances in Cryptology - CRYPTO 2003*. Ed. by D. Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 565–582. ISBN: 978-3-540-45146-4.

[9] S. Chow, J. Weng, Y. Yang and R. Deng. "Efficient Unidirectional Proxy Re-Encryption". In: *Progress in Cryptology – AFRICACRYPT 2010*. Ed. by D. J. Bernstein and T. Lange. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 316–332. ISBN: 978-3-642-12678-9.

[10] C. Costello. `http://www.craigcostello.com.au/pairings/PairingsForBeginners.pdf`. Accessed: 2018-06-03.

[11] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. `https://tools.ietf.org/html/rfc5246`. RFC Editor, Aug. 2008. URL: `https://tools.ietf.org/html/rfc5246`.

[12] *GStreamer*. `https://gstreamer.freedesktop.org/`. Accessed: 2018-05-22.

[13] L. Lamport. *Constructing Digital Signatures from a One Way Function*. Tech. rep. Oct. 1979. URL: `https://www.microsoft.com/en-us/research/publication/constructing-digital-signatures-one-way-function/`.

[14] B. Libert and D. Vergnaud. "Unidirectional Chosen-Ciphertext Secure Proxy Re-Encryption". In: *IEEE Transactions on Information Theory* 57.3 (Mar. 2011), pp. 1786–1802. ISSN: 0018-9448. DOI: `10.1109/TIT.2011.2104470`.

[15] B. Libert and D. Vergnaud. "Unidirectional Chosen-Ciphertext Secure Proxy Re-encryption". In: *Public Key Cryptography – PKC 2008*. Ed. by R. Cramer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 360–379. ISBN: 978-3-540-78440-1.

[16] B. Lynn. *PBC Library*. `https://crypto.stanford.edu/pbc/`.

[17] D. Naccache. "Standard Model". In: *Encyclopedia of Cryptography and Security*. Ed. by H. C. A. van Tilborg and S. Jajodia. Boston, MA: Springer US, 2011, pp. 1253–1253. ISBN: 978-1-4419-5906-5. DOI: `10.1007/978-1-4419-5906-5_518`. URL: `https://doi.org/10.1007/978-1-4419-5906-5_518`.

[18] D. Nuñez, I. Agudo and J. Lopez. "Proxy Re-Encryption: Analysis of constructions and its application to secure access delegation". In: *Journal of Network and Computer Applications* 87 (2017), pp. 193–209. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2017.03.005. URL: http://www.sciencedirect.com/science/article/pii/S1084804517301078.

[19] A. Sahai and B. Waters. "Fuzzy Identity-Based Encryption". In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by R. Cramer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 457–473. ISBN: 978-3-540-32055-5.

[20] K. Sako. "Semantic Security". In: *Encyclopedia of Cryptography and Security*. Ed. by H. C. A. van Tilborg and S. Jajodia. Boston, MA: Springer US, 2011, pp. 1176–1177. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_23. URL: https://doi.org/10.1007/978-1-4419-5906-5_23.

[21] M. Tibouchi. "Security Reduction". In: *Encyclopedia of Cryptography and Security*. Ed. by H. C. A. van Tilborg and S. Jajodia. Boston, MA: Springer US, 2011, pp. 1167–1168. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_515. URL: https://doi.org/10.1007/978-1-4419-5906-5_515.

[22] B. Waters. "Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization". In: *Public Key Cryptography – PKC 2011*. Ed. by D. Catalano, N. Fazio, R. Gennaro and A. Nicolosi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 53–70. ISBN: 978-3-642-19379-8.

[23] G. Zhao, L. Fang, J. Wang, C. Ge and R. Yongjun. "Improved Unidirectional Chosen-Ciphertext Secure Proxy Re-encryption". In: June 2010, pp. 476–480.

Printed by Tryckeriet i E-huset, Lund 2018

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2018-660
http://www.eit.lth.se