DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2018

A secure multi-party scheme with certificateless cryptography for secret key extraction

DENNIS FOKIN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

A secure multi-party scheme with certificateless cryptography for secret key extraction

DENNIS FOKIN

Master in Computer Science
Date: May 29, 2018
Supervisor: Sonja Buchegger
Examiner: Mads Dam
Principal: Hannes Salin, Nordea
Swedish title: Ett säkert multipartsberäknande protokoll med certifikatlös kryptografi för kryptonyckeluthämtning
School of Electrical Engineering and Computer Science

Abstract

Many systems contain sensitive data such as user credentials used for authentication purposes. For large systems, a common approach is to store the data in a configuration file at a trusted third party. However, that would imply a single point of failure if an adversary gains access to the trusted party. In theory this could be solved by encrypting the data but in practice this only moves the problem and does not solve it, since some type of credential data is needed to decrypt the configuration file.

A more flexible solution is needed that requires less of human interaction while also providing a higher degree of security. This thesis proposes a complete cryptographical system for solving this problem in a typical enterprise setting with a set of additional implementation requirements by using multi-party computation and Shamir's secret sharing protocol. Additionally, the work combines the mentioned system with a certificateless cryptography based multi-party computation protocol, since certificates usually implies a time-consuming process. The system has been evaluated in terms of security and efficiency with the conclusion that the results look promising. In terms of performance, the bulk of the overhead comes from certificateless cryptography, a constraint for the specific scenario which might not be present in general. The work also provides incentives for developing and further evolving Java libraries for cryptography, especially for multi-party computation and certificateless cryptography.

Sammanfattning

Många system innehåller känslig data, exempelvis användaruppgifter som används för autentiseringsändamål. För stora system är en vanlig lösning att lagra data i en konfigurationsfil hos en betrodd tredje part. Det skulle emellertid innebära att den svagaste länken är om motståndare får tillgång till den betrodda parten. I teorin kan detta lösas genom att kryptera data men i praktiken flyttar det bara på problemet men löser det inte, eftersom någon typ av autentiseringsdata behövs för att dekryptera konfigurationsfilen.

En mer flexibel lösning behövs som kräver mindre mänsklig interaktion samtidigt som det ger en högre grad av säkerhet. Denna avhandling föreslår ett komplett kryptografiskt system för att lösa detta problem i en typisk företagsmiljö med en ytterligare uppsättning implementationskrav genom att använda multipartsberäknande och Shamirs secret sharing protokoll. Dessutom kombinerar arbetet det nämnda systemet med ett certifikatfritt krypteringsbaserat protokoll kombinerat med multipartsberäkningar, eftersom certifikat oftast innebär en tidskrävande process. Systemet har utvärderats med avseende på säkerhet och effektivitet med slutsatsen att det ser lovande ut. När det gäller prestanda kommer huvuddelen av omkostnaden från den certifikatfria kryptografin, en begränsning för det specifika scenariot som kanske inte är närvarande i allmänhet. Arbetet ger också motiv för att vidareutveckla Java-bibliotek för kryptografi, speciellt för multipartsberäknande protokoll och certifikatlös kryptering.

Acknowledgements

I would first like to express my sincere gratitude to my supervisor Sonja Buchegger for the continuous support of my research. Her constant feedback and guidance helped me greatly with the research and the writing of the thesis.

My sincere and very profound thanks also goes to Hannes Salin at Nordea for helping me with his extensive knowledge and patiently answering any questions I had. Without his precious advice and insights this thesis would not have been completed or written.

Contents

Chapter 1

Introduction

Configurations of possibly sensitive data, such as technical user credentials used when an application integrates with external systems, are frequently used both manually and automatically by users and applications respectively. A typical approach is to store the configuration data in a text file on a server, thus implying the server is trusted and secured. However, in reality this is most likely not the case whereas any sensitive data may be compromised. It is further desirable to avoid having a single point of failure, as in the case with the trusted server.

Another approach addresses the use of vaults, i.e. either physical devices or software based solutions that manages keys for authentication. In this solution the sensitive configuration data can be encrypted, where the decryption key is locked in the vault. These types of solutions still need to be accessed by one or more master passwords which in turn have to be stored somewhere or remembered and typed in manually by a human.

A more flexible solution is needed that requires less of human interaction to access data but also provides better security, where the keys are not stored somewhere in one piece. Multi-party computation (MPC) protocols based on secret sharing have previously been used successfully in order to provide privacy to data in a data storage and retrieval system [18]. The efficiency and speed of MPC protocols have also been discussed plenty over the last years with the conclusion that the speed is very promising and that in the near future, MPC will hopefully be applied to more scenarios where distrusting parties wish to perform some computation [19]. It is therefore of interest to investigate the possible applications of MPC in the context of regenerating

keys.

1.1 Research Question

Is it feasible from an efficiency perspective to use secret sharing and multi-party computation in order to protect sensitive data without storing keys?

Is it possible to propose a cryptographic system that guarantees confidentiality of the configuration data given the proposed security model?

1.2 Objective

Compliance of software is the main focus of this thesis. More specifically, there usually exists internal regulations at companies regarding how software (and the security around them) should work, for example rules saying that sensitive data, such as passwords, cannot be stored in plain text. By encrypting the data and storing the decryption key in vaults the problem is still not solved since a vault needs a password, which in turn needs to be stored somewhere or remembered. This is further complicated by the fact that scripts and programs could want access to these passwords but again, such data cannot be stored and used in plain text by automated processes. The aim of this thesis is thus to research what other solutions are available, so that sensitive data is protected without having to store the keys somewhere in one piece.

Multi-party computation is a relatively old field. However, practical solutions have only started to be researched and implemented in the last years. Some argue that it is one of the more important fields in cryptography today [19]. The specific problem at hand is of interest since at most companies it is not uncommon to see a text-file with passwords and other sensitive data in plain text, a problem that could be solved by researching this topic more in depth.

Chapter 2

Background

This chapter will give some necessary background information and theory that the proposed system will incorporate. A section is also dedicated to related work, mentioning what other uses multi-party computation has.

2.1 Principal

Requirements, constraints, and use cases described in this thesis are all modelled based on local policies from Nordea, a Nordic bank who is the principal of this thesis. The solution proposed is general and can be applied to different companies in different industries.

2.2 Configuration data

For this paper, configuration data will be defined as data that could be classified as sensitive, containing information that applications need when communicating with other applications or systems. As such, the data could consist of IP addresses, passwords, or different input that is needed. While the possibilities here are endless, the aim of this thesis is to treat all the configuration data as credential data used for authentication purposes, containing usernames and passwords to servers, thus making sure the data is treated as highly sensitive. Any password stored as configuration data is therefore for user authentication to external (or internal) systems.

2.3 Secret sharing

A tool used frequently in cryptography is secret sharing. The aim is for a party P_i (usually called dealer) to spread a secret across other parties in such a way that no party (except for P_i) can recreate the secret. For instance, if P_i's secret is an integer x in $\mathbb{Z}_p$, where p is a prime then P_i can find three other integers such that

$$x = r_1 + r_2 + r_3$$

P_1 can share the secret with two other parties P_2 and P_3 by sending r_1 and r_3 to P_2, r_1 and r_2 to P_3, and r_2 and r_3 to P_1 itself. P_1 has successfully managed to secretly share the secret. Neither party alone can compute x, but it can be reconstructed by using shares from at least two parties [4]. Since the protocol involves multiple parties and each share should be combined and computed on, secret sharing is generally also considered to be a MPC scheme, depending on how the scheme is constructed since most MPC protocols are based on secret sharing [18].

2.3.1 Shamir's secret sharing

As stated by Shamir [21], a (k,n)-threshold scheme is a scheme where a secret D is divided into n shares (or pieces): $D_1, ..., D_n$ such that:

1) D is easily computed by knowing k or more pieces, and
2) D cannot be computed by knowing $k-1$ or less pieces.

Such a scheme is called a (k,n)-threshold scheme. The idea behind a threshold scheme is that they are helpful for managing cryptographic keys. Shamir states that in order to protect data we encrypt it. However, is not possible to encrypt keys, since keys are needed to decrypt the encrypted data. Thus, Shamir proposes a robust key management scheme by using a (k,n)-threshold scheme where $n = 2k-1$. This way the key can be recovered even if $n/2 = k-1$ shares are destroyed [21].

The scheme Shamir proposed is based on polynomial interpolation. Given k points in the 2-dimensional x-y plane $(x_1, y_1), ..., (x_k, y_k)$ there is only one polynomial $q(x)$ of degree $k-1$ such that $q(x_i) = y_i$ for all i, given that the x_i's are distinct [21].

In order to divide the secret D into n shares, a random $k-1$ degree polynomial $q(x)$ is chosen, where $q(x) = a_0 + a_1x + a_2x^2 + ... + a_{k-1}x^{k-1}$ and $a_0 = D$. The share D_i is then calculated by evaluating:

$$D_1 = q(1), ..., D_n = q(n)$$

Given any subset k of these shares D_i, the coefficients of $q(x)$ can be found by interpolation. The secret is found by evaluating $q(0) = D$ [21].

Verifiable secret sharing

If the dealer in a secret sharing scheme is malicious the parties participating in a secret sharing protocol can get inconsistent or incorrect shares. For this verifiable secret sharing schemes can be used instead, where each party can verify that their share is consistent. Shamir's secret sharing scheme is not verifiable. However, Paul Feldman [8] proposed a modification of Shamir's protocol in order to make it verifiable.

2.4 Multi-party computation

According to [19] multi-party computation is "arguably the most general and important problem in cryptography". As a result of the growth of the Internet, many opportunities for cooperative computation have been created where different parties are conducting computational tasks together [7]. These tasks can occur between trusted or untrusted entities that each have a secret. Generally, in MPC there are n parties, each with their own secret who wish to compute a function based on these secrets. However, this should be done without revealing any new information to the other parties, except for the result of the function. The most famous problem dealing with this is Yao's Millionaires' Problem where Yao asks "Two millionaires wish to know who is richer; however, they do not want to find out inadvertently any additional information about each other's wealth. How can they carry out such a conversation?" [26]. Andrew Yao asked and answered this question in 1982, thus creating a subfield within cryptography known as multi-party computation. Even though the question was answered by Yao over 30 years ago, there is still research to be done in this area, since computers are only now reaching a level that is capable of calculating

necessary functions for MPC. This is also why practical and efficient solutions have only started to be developed and researched in the last years.

In a multi-party computation problem there are n players, or parties, that participate: $P_1, P_2, ..., P_n$. Each player P_i holds a secret input x_i which they do not want to reveal to the other players. The players agree on a function f that takes n parameters as input and outputs an answer y, such that $y = f(x_1, x_2, ..., x_n)$. At the same time, two conditions must be satisfied:

1) Correctness: the correct value of y is computed
2) Privacy: y is the only new information released.

If both condition 1) and 2) are satisfied when computing function f, it is said that f has been computed securely [4].

A trivial solution to this problem is that all parties agree on a trusted third party T which would gather all the inputs and compute the function f [4]. T announces the result to the parties and then discards all computation results. As pointed out by Cramer et al [4] there are at least two problems with this approach. Firstly, it creates a single point of failure. If an adversary gets access to the third party, all private data could be stolen. Secondly, all parties must now trust T. If they do not even trust each other, how are they supposed to trust T?

The fundamental problem secure MPC is trying to answer is "Do we have to trust someone?". The answer is surprisingly no [4]. By designing a protocol where all parties are exchanging messages between each other, the function f can still be calculated without having a trusted third party and without ever revealing their secrets.

MPC is therefore capable of performing various calculations without revealing any new information to the other parties, except for the result of the function. As such, MPC can preserve each party's privacy.

2.4.1 Two-party computation protocols

Imagine that two parties, $Alice$ and Bob, both have a secret: a and b. Assume the simplest case where the secrets are boolean values. The two parties now want to compute the conjunction of their secrets but

they do not want to reveal anything to each other except for the result of the function $AND(a, b)$. This can be accomplished by using Garbled Circuits, as introduced by Yao in several talks [1].

In a garbled circuit, a regular boolean truth table is computed and obfuscated so that no additional information is leaked. In the case described above, one party, for example $Alice$, will generate a garbled truth table and send it to Bob. Table 2.1 shows a truth table for an AND gate, consisting of two input wires, a and b, and an output wire z.

a	b	z
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.1: Truth table for an AND gate

After generating the truth table, $Alice$ chooses four random "garbled" labels, known as garbled input, one for each value:
$W_A^1, W_A^0, W_B^1, W_B^0$, where W_A^0 means that wire a has value 0, and so on. $Alice$ does the same with the output, thus generating two garbled outputs: W_Z^1, W_Z^0. She further encrypts the output wire z with the input wires. As an example, the input wires W_A^0, W_B^1, which represent the second row in table 2.1 above, will encrypt the output as $E_{W_A^0}(E_{W_B^1}(W_Z^0))$. Finally, the rows in the table are permutated. The final garbled table will consist of all the rows from the output column, in this case column z, and could look as in table 2.2. Notice that the rows have been permuted:

z
$E_{W_A^0}(E_{W_B^1}(W_Z^0))$
$E_{W_A^0}(E_{W_B^0}(W_Z^0))$
$E_{W_A^1}(E_{W_B^1}(W_Z^1))$
$E_{W_A^1}(E_{W_B^0}(W_Z^0))$

Table 2.2: Garbled table for an AND gate

This table is then sent to Bob. An interesting feature of this table is that if Bob receives two garbled inputs (for example W_A^0 and W_B^0), he is only able to decrypt one entry of the table. It is impossible for him

to decrypt any other value. Further, the garbled inputs do not reveal any information about the actual input bits [14]. Thus, Bob tries to decrypt all encryptions, and only one will decrypt correctly. It is also worth to mention that Bob receives the correct garbled output value without knowing anything about the computation he has just done. He has no information regarding what values are associated with the garbled ones, and he has no information on which gate he has computed. However, he will still end up with a correct computation of the gate [14].

Upon receiving the table, Bob now wishes to compute the correct value. For this he needs two garbled inputs, W_A^a and W_B^b. $Alice$ can send W_A^a without any problems since she knows a, and this will reveal nothing about a to Bob, since the garbled label is random. However, getting the garbled input W_B^b is more difficult. $Alice$ cannot, for example, send both W_B^0 and W_B^1, because Bob can then decrypt two outputs. Another problem is that $Alice$ does not know b, and cannot know b, meaning Bob cannot simply ask $Alice$ for the correct garbled value. The answer here is that the two parties should participate in an 1-out-of-2 oblivious transfer protocol for each input wire of Bob. $Alice$ sends $W_B^{x_0}$ and $W_B^{x_1}$, where $x_0 = 0$ and $x_1 = 1$, and Bob chooses the one that corresponds to his secret b. This way, Bob learns the correct W_B^b garbled input and $Alice$ learns nothing. Further, Bob will learn nothing about the other value W_B^{1-b} that was also sent to him.

2.4.2 Multi-party computation protocols

According to [14], there exist several multi-party constructions that follow Yao's garbled circuit construction. However, the authors mention that there are several drawbacks to this. For instance, public-key cryptography has to be used instead of symmetric key operations for every gate in the circuit, and all protocols require communication between every pair of the participating parties resulting in inefficient solutions. Usually, these protocols are mathematically more expensive. In most cases they also rely heavily on secret sharing.

BGW

Ben-Or, Goldwasser, and Wigderson designed a protocol that holds if $t < n/2$ semi-honest corrupted parties participate, or $t < n/3$ malicious

parties [2]. The parties in BGW share their inputs by using Shamir's secret sharing with a degree of $(n-1)/2$ polynomials in the semi-honest attacker, in order to attain privacy for when $t < n/2$ parties are corrupted. In the case of malicious attackers a verifiable secret sharing scheme is used.

Addition gates can be computed locally at every party. For example, assuming that a gate is computing $a + b$, where a and b are input wires, each party P_i locally computes $c_i = a_i + b_i$. Addition of two polynomials of degree x is another polynomial of degree x. However, when multiplying this is not as straightforward. When calculating $a*b$, P_i can locally compute $d_i = a_i b_i$. The polynomial d will thus have a degree of $2x$. Reconstruction works as long as the polynomial used for sharing is of degree x. The polynomial therefore has to be reduced back to x, which can be accomplished with Lagrange interpolation, by having each party share its d_i value. All parties then use Lagrange interpolation to get shares $d_1, d_2, ..., d_n$ of d.

2.5 Certificateless cryptography

In some cases where traditional PKI encryption schemes are not possible, certificateless cryptography could be a substitution. As mentioned by Alexander Dent [6], "In a public-key encryption scheme, a sender encrypts a message based on a public key which has been certified by a PKI". The certificate binds the public key to a digital identifier. In other words, it is possible for parties to verify who the sender of a message could be. However, this is usually a computational burden. Additionally, in some settings it could also be a time consuming process, for example in large companies using many integration nodes using authentication to issue certificates.

In certificateless cryptography the public key does not have to be verified via a certificate. Instead two public/private key pairs are used. The traditional public and private keys are generated by the user. There is also an identity-based key pair, consisting of the user's digital identity and a partial private key, supplied by a key generation centre (KGC).

For this thesis certificateless cryptography is used as a certificateless signature scheme, where a sender can sign a message and a receiver can verify that the message was signed by the sender. A sender

signs a message by using elliptic curves and bilinear maps, more on this in Section 4.3. For this the sender uses a random private value a and a private key, both of which are only known to the sender, and a public generator G known to everyone. The receiver can easily verify that the message was signed by the sender by using the sender's public key and the sender's ID. The sender's ID is also used to compute the sender's private key, meaning that there is a mathematical correlation between several variables used in the algorithm, which is why it is possible to use as a substitution for PKI [20].

2.6 Abstract algebra

2.6.1 Binary operation

A binary operation $*$ on a set S is a mapping from $S \times S$ to S. In other words, $*$ is a rule which assigns two elements from S to an element of S [11].

2.6.2 Groups

A group G consist of a set G and a binary operation $*$ on G, which should satisfy the following three axioms [11]:

1. The group operation $*$ is associative. Meaning, $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$.

2. There is an identity element $1 \in G$, such that $a * 1 = 1 * a = a$ for all $a \in G$.

3. For each $a \in G$ there is an inverse element $a^{-1} \in G$ such that $a * a^{-1} = a^{-1} * a = 1$.

Cyclic groups and generators

A group is called cyclic if there is an element $\alpha \in G$ such that for each $b \in G$ there exists an integer i so that $b = \alpha^i$. The element α is called a generator of G [11].

Groups for this thesis

The notation G^* is used to denote the set of non-identity elements of the group [20]. The main groups used in this thesis are Z_n, G_1, G_2. The group Z_n is the set of integers under addition modulo n, in this case $\{0, 1, 2, ..., n-1\}$. Z_n^* is still under addition modulo n and contains the same set of integers except for the identity element 0. G_1 is an additive group and G_2 is a related multiplicative group. Both G_1 and G_2 are sets of points to some specific elliptic curves. They are also cyclic groups of large prime order related to elliptic curves.

2.7 Elliptic curve cryptography

Elliptic curve cryptography (ECC) is an approach to public key cryptography, where each user partaking in the communication is equipped with a public and a private key. The public key is distributed to all other users, but the private key remains a secret for that particular user. A set of operations are also associated with the keys in order to do cryptographic operations. These mathematical operations are defined on the elliptic curve given by the equation $y^2 = x^3 + ax + b$, where $4a^3 + 27b^2 \neq 0$ has to be satisfied. By changing a and b, different elliptic curves can be created. Each point $P_i(x_i, y_i)$ that satisfy this equation is a point on the curve. Additionally, there is a "point at infinity". In elliptic curve cryptography, the public key P is a point on the curve and the private key is a number n. The public key is obtained by multiplying the generator α, i.e. a point on the curve, with n [12, 17, 10, 20].

2.7.1 Discrete logarithm problem

The security of ECC relies on the difficulty in solving the equation $P = n\alpha$, where P and α are two known points on the curve. This problem is know as the discrete logarithm problem and is computationally infeasible to solve, given that n is sufficiently large [12, 17, 10].

2.7.2 Point addition

Let $P = (x_p, y_p)$, and $Q = (x_q, y_q)$ be two points on the elliptic curve. Point addition is the addition of P to Q, to obtain another point R also

on the elliptic curve. Visually, this is represented by drawing a line through P and Q, joining them and finding the third point of intersection of the line and the curve. This point is $-R$. R is then the reflection of $-R$ in the x-axis [20, 10].

Analytically, the aim is to find x and y coordinates for $R = (x_r, y_r)$. This is given by the equations $x_r = s^2 - x_p - x_q$, and $y_r = -y_p + s(x_p - x_r)$, where $s = (y_p - x_q)/(x_p - x_q)$ is the slope of the curve between P and Q [10].

2.7.3 Point doubling

Let $P = (x_p, y_p)$ be a point on the elliptic curve. Point doubling is the addition of P to itself, to obtain another point R also on the elliptic curve. Visually, this is represented by drawing a tangent at P and finding the point of intersection of the tangent with the curve. This point is $-R$. R is then the reflection of $-R$ in the x-axis [20, 10].

Analytically, the aim is to find x and y coordinates for $R = (x_r, y_r)$. This is given by the equations $x_r = s^2 - x_p$, and $y_r = -y_p + s(x_p - x_r)$, where $s = (3x_p^2 + a)/(2x_y)$ is the tangent at point P and a is a parameter chosen for the elliptic curve [10].

2.7.4 Bilinear Maps

Let α be a generator of the additive group $\mathbb{G}_1$ of prime order q. Let $\mathbb{G}_2$ be the multiplicative group where $|\mathbb{G}_1| = |\mathbb{G}_2|$. A pairing is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, with the following characteristics [20]:

Given $G, W, Z \in \mathbb{G}_1$, the following holds:

$$\hat{e}(Q, W + Z) = \hat{e}(Q, W) \cdot \hat{e}(Q, Z)$$

and

$$\hat{e}(Q + W, Z) = \hat{e}(Q, Z) \cdot \hat{e}(W, Z)$$

One consequence of this is that for any $a, b \in \mathbb{Z}_q$ we have:

$$\hat{e}(aQ, bW) = \hat{e}(abQ, W) = \hat{e}(Q, abW)$$

2.8 Related work

MPC has previously been used in several different areas. Nair et. al [18] demonstrated how multi-party computation could efficiently be used to design and implement a private data storage and retrieval system that allows any client to retrieve data from a database while preserving the data's privacy. They proposed a system where confidentiality is achieved by using Shamir's secret sharing, and all data is divided into shares. The system uses several database servers, each containing a share of the complete table. The shares are then combined to form the original data. Since the computation is done on shares, the work uses MPC. However, the work involved several trusted parties, one which actually takes the shares and reconstructs them. Furthermore, nothing in the paper has been said regarding an adversary that corrupts these parties, or even the databases keeping the shares.

Lindell and Pinkas [14] discussed the possibility of using MPC for privacy-preserving data mining where certain algorithms are used on confidential data. The authors give an example of a setting where two or more parties want to run a data mining algorithm on the union of data, while not revealing another party's data. In the proposed system, cryptographic primitives such as oblivious transfer are being used in a setting involving two parties, where the security rests on the decisional Diffie-Hellman assumption. [3] proposed and showed the first large-scale and practical application of MPC, by implementing a system used by Danish farmers for trading trade contracts for sugar beet production. MPC played a vital part for ensuring that each bid submitted to the action was at all times encrypted. No party had access to the bid at any time.

This thesis wants to research the possibility of using MPC in order to transition from a system where keys are located in one place to a system where they are not. In [16] a system was proposed for authenticating and verifying passwords that differs from the traditional solution of having just one server, thus bypassing the single point of failure. The system accomplished its task by involving a set of servers, where a certain threshold of servers must participate, but where compromise of a few servers does not attack the system. As noted by the authors, this work did not follow general multi-party computation techniques and focused solely on distributed authenticated key exchange between a client and k servers. The goal of the protocol was that the client would

end up with k session keys, one with each server, meaning that two parties will know each key; the user and the corresponding server.

This work is meant to contribute to the current state of secret sharing and multi-party computation research, and will therefore follow traditional MPC techniques. Furthermore, the main goal is to apply these topics to traditional settings within the financial industry in order to remove dependency of keeping keys in one single place. Instead MPC will regenerate the keys by using k parties. In traditional MPC-based protocols the k parties are jointly computing a function benefiting all parties, however, for this work the k parties will compute a function for helping one party access their own information, another area lacking research within MPC. In [18] shares from k databases are collected in order to give a user some result, thus proving that research in MPC where k parties are jointly computing a function for one specific user has already been slightly researched. However, the proposed system only focused on databases and nothing was said regarding an adversary that corrupts any of the parties. Furthermore, unlike [14], the setting that should be researched will contain more than two parties, meaning that expanding on the authors' decision of using oblivious transfer in a multi-party scenario might be computationally expensive. Additionally, in contrast to [16], only one party should have knowledge about the key. Similarly to [16], a threshold will be implemented for withstanding attacks where adversaries compromise a subset of the parties, making them untrusted.

From the related works mentioned, it is clear that the field of secure MPC is only now beginning to grow, even though the subject is relatively old. Orlandi [19] showed that MPC protocols have previously been rather slow but are now being applied to vastly different areas, from databases to data-mining thus creating new incentive for applying MPC to specific problems.

Chapter 3

Solution design

This chapter will explain the problem with more details and also propose a solution design.

3.1 Problem

In the system there are internal client applications $I = \{I_1, ..., I_n\}$, external servers $E = \{E_1, ..., E_n\}$, and a gateway server G.

A typical use case is that an internal client I_i wishes to send information to an external server E_j. In order to do this, I_i must be authenticated. For this, a gateway server G is used where login credentials for G on behalf of I_i are stored in configuration files. I_i communicates with G who then looks up the appropriate credentials in order to communicate with E_j, i.e. loading necessary credentials for authentication towards E_j. See Figure 3.1. G sends this to E_j. G then relays back E_j's response to I_i, which normally is an "OK" or "Error" response type.

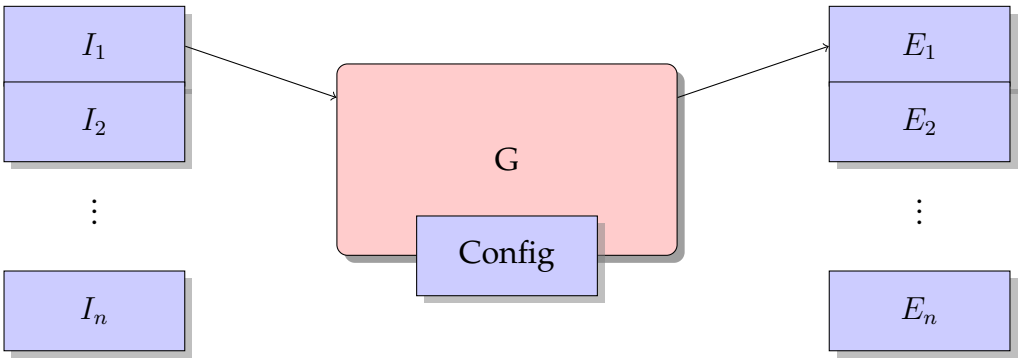

Figure 3.1

However, since I could consist of hundreds of internal clients, all with different credentials, it is vital that the sensitive configuration data does not leak. Further, an adversary could gain access to the file with the data that G is trying to read, with catastrophic results.

As an improvement, instead of having data stored in plain text it is stored encrypted. In that scenario, any attacker gaining access to the server will not be able to authenticate as I_i since the credentials are encrypted. A key is needed to decrypt the data which is stored in a vault V. Upon request from I_i (1 in Figure 3.2) G decrypts the vault (2 in Figure 3.2), takes the key (3 in Figure 3.2), and decrypts the data and sends it to E_j (4 in Figure 3.2). However, G must have a key to V which is not protected, resulting in the same catastrophic result.

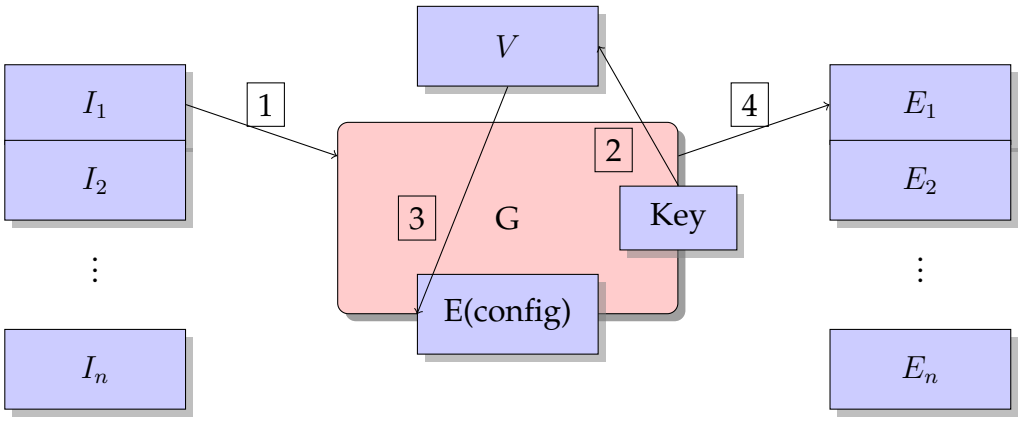

Figure 3.2

3.1.1 Constraints

There are several organizational and infrastructure related constraints acting on the use case, coming from the principal:

- The internal clients I could consist of hundreds of clients, with varying request frequency.
- I can only communicate with G, and E can only communicate with G, for safety reasons acting as a security layer.
- No PKI certificates can be used for authentication between I and G.
- Any solution needs to be robust against the assumption that any node can be compromised and the internal network can be eavesdropped upon.

3.1.2 Assumptions and limitations

For this paper it is assumed that all keys have previously already been generated through a safe key ceremony. Therefore key generation, revocation, or similar aspects are out of scope for this thesis.

3.2 Solution

One possible naive solution is to introduce $P = P_1, ..., P_m$ as a set of parties used in an MPC scheme, since it is then possible to create a distributed data protection protocol where no keys are stored in one place. The configuration data is restructured so that every pair of sensitive data between two parties I_i and E_j are encrypted with a key key_{I_i,E_j}. Vault V is setup to contain these keys.

The proposed system consists of several parts, see Figure 3.3:

1. I_i sends a request to G, stating that it wishes to communicate with E_j.
2. G sends a request to a cluster of nodes P, initiating an MPC protocol
3. P together with V computes a private key key_{I_i,E_j}
4. V sends key_{I_i,E_j} to G
5. G decrypts the configuration data containing credential information between I_i and E_j, and sends the credentials to E_j.

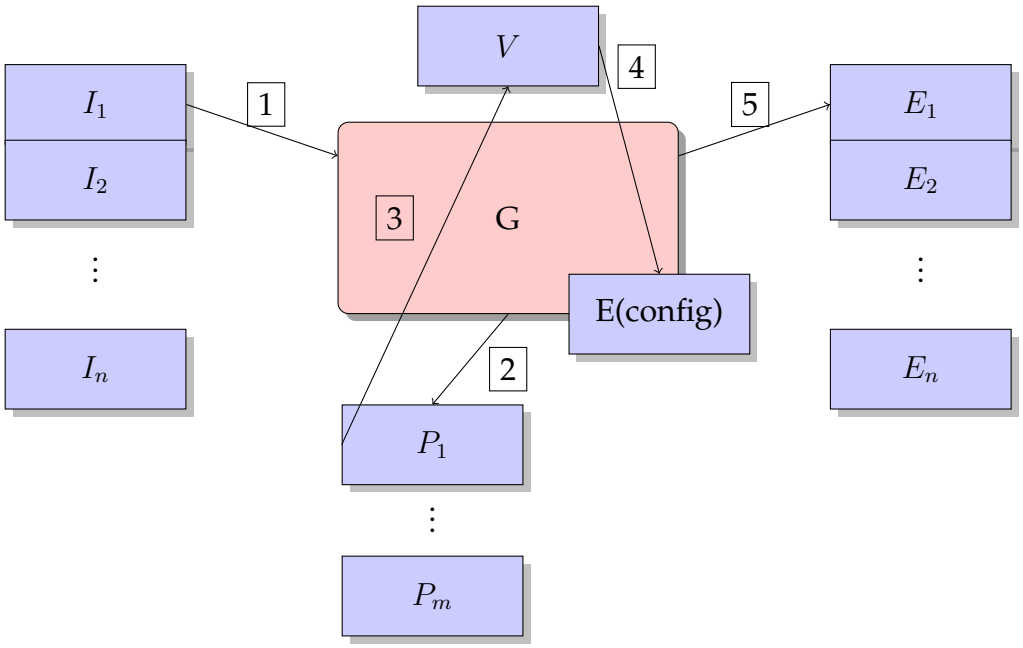

Figure 3.3

The proposed system would thus solve the issue of storing keys in one piece somewhere. Instead, the MPC cluster of nodes together with V reconstruct a key in order to decrypt the configuration data. Since every pair of data in the configuration data is encrypted with different keys, minimal information is exposed each time, meaning that only the necessary data is revealed.

3.2.1 Establish client identity using MPC

A problem with the above mentioned proposed system is that an adversary could corrupt G and therefore trigger the MPC protocol whenever they want. One way of mitigating this attack type is to make sure that G can only initiate the MPC protocol if it has received a valid request from I_i. In order to guarantee that a request was sent by I to G, without using PKI, certificateless cryptography is used in an MPC scheme. Generally, a certificateless cryptography scheme requires the need for a Key Generation Centre (KGC) that manages keys. As described in [6], two key pairs are needed:

1. Private $k^i_{private}$ and a public key k^i_{public} generated by the receiver, in this case i. They act as traditional private and public keys.

2. The receiver's digital identifier $id^i_{idPublic}$ and the partial private key, $id^i_{idPrivate}$. The partial private key is generated by the KGC.

The proposed system will therefore incorporate a KGC with MPC as underlying computational basis in order to verify that a request came from I. The KGC will initiate the protocool with the set of parties P. See Figure 3.4.

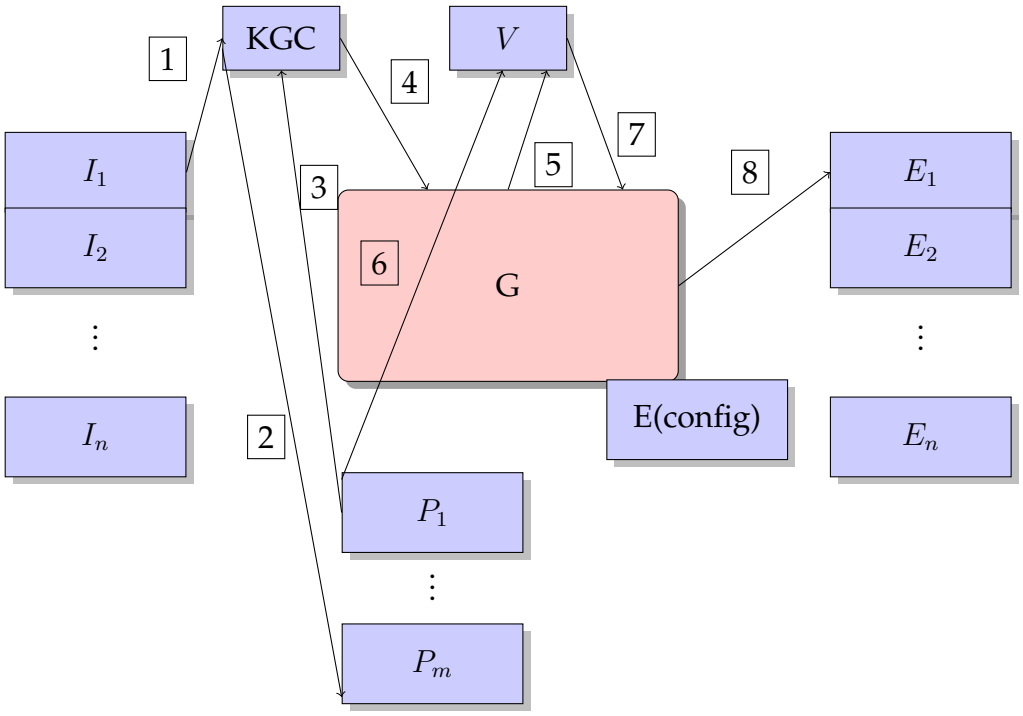

Figure 5.1

The KGC parties involved in the MPC protocol for certificateless cryptography can be implemented by a secret sharing algorithm such as Shamir's secret sharing, as in this paper, or perhaps even the GMW algorithm. The MPC parties could also use Feldman verifiable secret sharing.

3.3 Security model

For this thesis it is assumed that the attacker is modelled as a probabilistic polynomial time adversary, meaning more powerful than any deterministic type of model. The attacker can compromise nodes used in the system. There are three attack scenarios that are of relevance:

1. the adversary corrupts the parties involved in the certificateless cryptography,

2. the adversary corrupts the parties involved in the key regeneration protocol, and

3. any of the internal nodes (I or G) become corrupted.

In the first two scenarios, the protocol should be capable of regenerating the keys even if $t \leq n - 1$ parties have been corrupted. It is assumed that if an adversary can corrupt one or several parties, they are free to send arbitrary messages to other parties. The corrupted parties have three choices, they may either refuse to participate in the protocol at all, they may substitute their secret with something else, or they may abort the protocol prematurely.

In the second attack, the attacker is assumed to have full privileged administrator access to any internal node.

It is further assumed that G is contained within a network, separately from any other party, protected by typical industry solutions such as firewalls and proxies. The initial attacker model will not consider communication security, i.e. data in transit between G and P or between G and any external systems E.

Chapter 4

Method

This chapter will describe more in-depth the design of the proposed system. It consists of a combination of known protocols. Contribution to the field of MPC has been made by altering parts of these protocols. This chapter will also contain information regarding how the results will be evaluated.

4.1 Encrypted configuration data

By encrypting the data, an adversary cannot easily gain access to any sensitive data at G. Instead of encrypting all data with a single key, each pair of sensitive data between two parties I_i and E_j are encrypted with different keys key_{I_i,E_j}. Vault V will regenerate the necessary keys by executing the key regeneration protocol.

4.2 Key regeneration multi-party computation protocol

The solution will use an MPC protocol based on Shamir's secret sharing for regenerating the decryption key. This protocol will be initiated by V.

Each key key_{I_i,E_j}, henceforth referred to as s, is split into random shares $s_1, s_2, ..., s_m$ and each share s_i is sent to each party $P_i \in P$. By using Lagrange interpolation the key can be reconstructed and the data can be decrypted. For this, Shamir's secret sharing scheme will be used, where $n = 2k - 1$, as recommended [21].

4.3 Certificateless multi-party computation protocol

The solution will use a certificateless signature scheme [20] for establishing client identity, where a small part of the scheme will contain MPC. This protocol will be used between the set of internal nodes I and G. This protocol bases its mathematics on elliptic curve cryptography and bilinear maps. The scheme is specified by seven algorithms: Setup, Partial-Private-Key-Extract, Set-Secret-Value, Set-Private-Key, Set-Public-Key, Sign and Verify. The algorithms are described in the following way [6]:

1. In the Setup part, the KGC generates a master public key $s \in \mathbb{Z}_q^*$ and a private master key k, used for the whole system. The master public key is public for everyone in the system while the private master key is kept private for the KGC only. This part will also generate two groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order q and the bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. A generator $G \in \mathbb{G}_1$ shall also be chosen here. Lastly, $G_0 = sG$ has to be set.

2. In Partial-Private-Key-Extract, the KGC creates a partial-private key for user A by using A's identity ID_A and the master private key. The partial-private key is then delivered to A.

3. In Set-Secret-Value, the user A sets a random secret value $x_A \in \mathbb{Z}_q^*$, which is uniformly distributed.

4. In Set-Private-Key, the user A computes their private key. This is generated by A's partial-private key and secret x_A. This way, no one except for A knows A's private key, not even the KGC.

5. In Set-Public-Key, the user A computes their public key P_A. This is generated by the secret x_A. $P_A = (X_A, Y_A)$, where $X_A = x_a G$ and $Y_A = x_a G_0$.

6. In Sign, a message M is signed with a user's private key.

7. In Verify, a signature on a message M is verified by using the identity of the sender ID_A and their public key.

For this paper step 1-5 are not of any interest since they cover key generation aspects. Instead, step 6 and 7 are of more importance since the assumption is that all keys are already set in an initial setup step once.

4.3.1 Algorithm design: Sign

To sign a message M using user A's private key S_A several steps have to be executed:

1. First, a random value $a \in \mathbb{Z}_q^*$ of some prime order q has to be chosen.
2. In the second step $r = \hat{e}(aG, G) \in \mathbb{G}_2$ is being calculated.
3. Then $v = H(M, r) \in \mathbb{Z}_q^*$ should be set, where H is a cryptographically secure hash function mapping a bit-string to an element in $\mathbb{G}_1$, i.e. the digest is a point on the curve.
4. In the last calculation, $U = vS_A + aG \in \mathbb{G}_1$.

The signature is then outputed as (U, v) [20].

4.3.2 Algorithm design: Verify

In order to verify the signature (U, v) on a message M for identity ID_A with public key (X_A, Y_A) the following steps are executed [20]:

1. Check $\hat{e}(X_A, G_0) = \hat{e}(Y_A, G)$. Because of the characteristics of bilinear maps, this should hold.
2. Compute $r = \hat{e}(U, G) \cdot \hat{e}(Q_A, -Y_A)^v$, where $Q_A = H(ID_A) \in \mathbb{G}_1^*$.
3. Check if $v = H(M, r)$ holds.

4.3.3 Modification

The certificateless cryptography protocol used in this system will be an MPC protocol. This is made possible by modifying the algorithm. For this paper, each party's private and public keys are assumed to have already been generated, see Section 3.1.2. In order for the users

to not have to store the keys at any time, they will instead be regenerated once needed by using the same principle of key regeneration with MPC. This will slow down the protocol somewhat but it is deemed to add a higher degree of security to the whole system.

4.4 Evaluation

The proposed protocol will be evaluated in three aspects: solvability, security measure, and efficiency. If the protocol manages to solve the objective described in Section 1.2, the solvability case is considered accomplished.

The point regarding security will be evaluated by applying the attacker model on the system model. There are three attacks the protocol must be able to defend against, all written in 3.3. It is also a requirement to guarantee the confidentiality of the configuration data.

The last point will be evaluated by measuring computational overhead, such as time. If the protocol is slower than a specified overhead threshold, it is deemed to have sacrificed too much efficiency. More on this in Section 4.4.1.

4.4.1 Efficiency requirements

The efficiency requirement of the system when the set P contains three parties is set to 500ms. This is based on the fact that the internal systems that use the set of clients I usually have timeout values. If there is no response during this time a circuit breaker is activated, the communication gets interrupted, and an error will be sent to the logs. Different systems have different timeout values and around 500ms is the lowest common threshold between them all.

There is no requirement on the network or on storage, since the most important aspect for this proof-of-concept is exactly running time. Storage and network can differ plenty between companies. However, the running time will not differ much between different hardware, even though it is possible to reduce.

Chapter 5

Implementation

This chapter will explain the different libraries used for the implementation of the system.

5.1 Libraries

5.1.1 Java Pairing-Based Cryptography library

The Java Pairing-Based Cryptography library (jPBC) is a Java port of the PBC library written in C which "provides the implementation of different types of elliptic curves that are efficiently computable and are still cryptographically secure" [5]. jPBC simplifies the use of bilinear maps for the Java programming language which is also why it has been chosen for this thesis. Different types of elliptic curves are also supported by this library.

For this thesis the certificateless cryptography portion is based on the Type A pairings from jPBC, for its simplicity. They are constructed on the curve $y^2 = x^3 + x$ over the field $\mathbb{F}_q$ for some prime $q = 3 \bmod 4$. The two groups $\mathbb{G}_1$ and $\mathbb{G}_2$. used in this thesis are of prime order r. The order of r is some prime factor of $q+1$. $q+1 = r \times h$. The variables for this type is [24]:
$q = 87807107996633125224377819847540498158068831994142082110286533992664756308802229570786251794226622214231558587695823174592777133673174813249251299982247 91$

$h = 12016012264891146079388821366740534204802954401251311822919615131047207289359704531102844802183906537786776$

$r = 730750818665451621361119245571504901405976559617$

5.1.2 Shamir's secret sharing library

A Java implementation of Shamir's secret sharing from open source project Codahale was used [22]. It is a (k,n)-threshold that splits a secret s into n shares, where k shares are needed in order to recover s. Lagrange interpolation is used to recover a secret.

5.1.3 Spring Framework

The Spring Framework is a Java platform that provides infrastructure support for developing Java applications [23]. For this paper, Spring is used to enable http requests between nodes.

5.2 Design

The system is developed in the Java programming language for several reasons. Firstly, it is the most popular programming language as of the writing of this thesis according to Tiobe [25], which means that most people who have learned to program have also learned to program in Java. The principal's current system is Java based and therefore any solution should be running with Java.

The code for the proof-of-concept is available at `https://github.com/hannessalin/masterthesis2018` and consists of several different Java classes:

1. **SMPCApplication.java** - the main class for each node.
2. **SecretFunctionService.java** - a class that represents the f function in an MPC protocol. For this proof-of-concept that method is called compute(). It will collect each party's share of the secret in order to regenerate the secret.
3. **SMPCController.java** - a controller class for Spring that handles every http request.

4. **CLC.java** - a class containing the certificateless cryptography protocol for signing and verifying messages.

The implementation is based on the solution design from chapter 3, where each node will be a Java object of the SMPCApplication.java object on different ports. Communication is done via http requests, which are sent between entities using different ports.

5.2.1 Typical scenario

Figure 5.1 below demonstrates a typical scenario between one internal node I_1 and one external node E_1, with three parties partaking in the MPC protocols. The figure shows a typical flow with typical port numbers that can be used. All communication between nodes is sent by http GET requests to `http://localhost:port` localhost since all parties run on the same server, this could be changed to different servers but it is assumed such difference would make the performance overhead negligible since any such servers run within the principal's internal network structure.

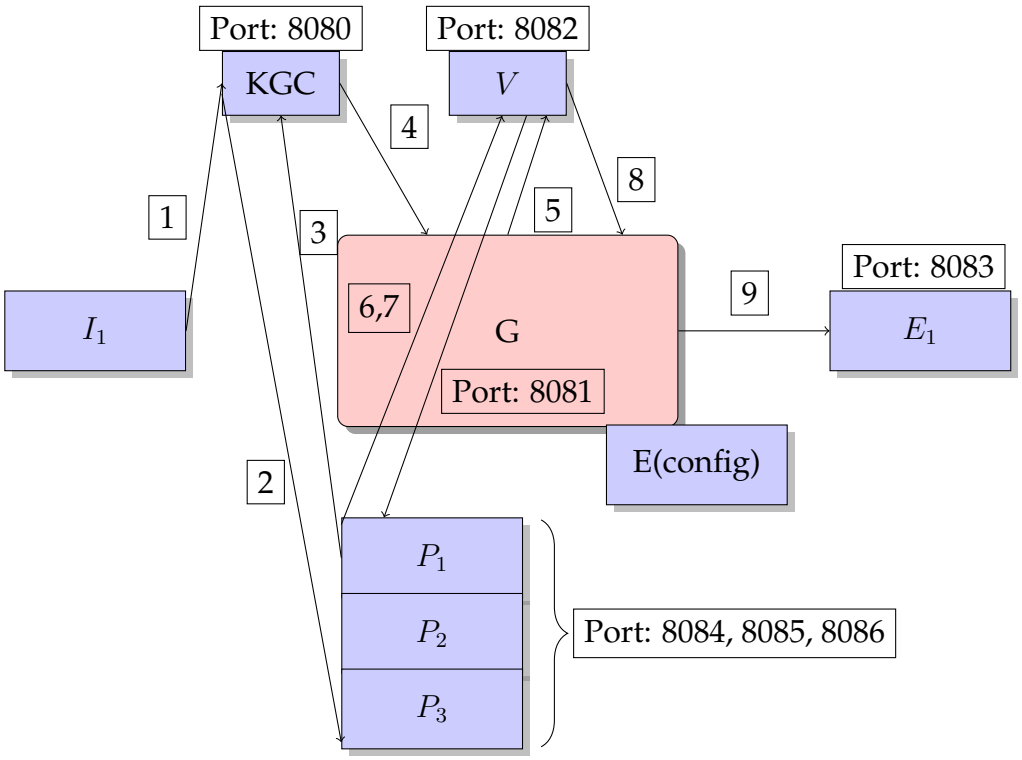

Figure 5.1

1. The internal node I_1 initiates the protocol by sending an http GET request to `http://localhost:8080/ItoKGC`, which is a request to the KGC. The KGC in turn will initiate the certificateless cryptography protocol by creating a CLC.java instance.

2. When KGC is running the CLC algorithm, the `compute()` function in SecretFunctionService.java is executed in order to recover public and private keys used for the CLC algorithm. This is done by sending an http GET request to each party in the set of parties P: `http://localhost:8084/KGCtoP1`, ..., `http://localhost:8086/KGCtoP3`.

3. Upon receiving the GET request from KGC, all parties in P send their share of the secret to the KGC with an http GET request to `http://localhost:8080/P1toKGC`, ..., `http://localhost:8080/P3toKGC`. The KGC uses the received shares to recover the public and private keys and then computes all necessary curves and groups needed for the CLC algorithm, such as elliptic curves,

generators, and bilinear maps. At this point I_1 can sign a message.

4. If I_1 has signed a message it should be verified by G. If it is verified, a request from KGC to G is sent, stating that I_1 is a verified node and the protocol can continue. For this, an http GET request `http://localhost:8081/KGCtoG` is sent.

5. G sends an http GET request `http://localhost:8082/GtoV` to V in order to initiate the key regeneration protocol.

6. V receives the request from G, and initiates the MPC protocol for recovering the decryption key. This is done by sending an http GET request to the parties in P: `http://localhost:8084/VtoP1`, ..., `http://localhost:8086/VtoP3`.

7. When the set P receives the requests from V, they send their share in an http GET request to V: `http://localhost:8080/P1toV`, ..., `http://localhost:8080/P3toV`.

8. When V manages to regenerate the decryption key, it gets sent to G with an http GET request `http://localhost:8081/VtoG`.

9. G decrypts the credentials in the configuration file and sends this to E_1 in order to authenticate I_1. This is done via an http GET request to `http://localhost:8083/GtoE`.

10. (not visible in figure) E_1 sends the response for I_1 to G. G relays the response to I_1.

Chapter 6

Results

This chapter will evaluate how well the protocol performs based on the evaluation criteria given in Section 4.4.

6.1 Solvability analysis

In Section 4.4 it is stated that the solvability aspect of the thesis is considered accomplished if the proposed solution design will solve the objective described in Section 1.2. This objective includes compliance of software with regulations that sensitive data cannot be stored in plain text. This is the general problem but the thesis is also incorporating several constraints listed in Section 3.1.1.

The objective from 4.4 has been solved with the proposed solution design. Originally, the configuration data was stored in a text file at a server, see Section 3.1 Figure 3.1. This is solved by encrypting the sensitive data stored in the configuration file. The decryption key can be stored in a text file at a server but that does not solve the problem, see Section 3.1 Figure 3.2. The key is therefore regenerated with a secret sharing based protocol involving multiple parties.

The constraints from 3.1.1 have also been fulfilled. The proposed setup can handle many internal clients I, where each I_i can only communicate with G. The set of external clients E can only communicate with G. In order to verify that a request came from an internal client I_i, message have to be signed by I_i and verified by G. As a requirement no PKI certificates can be used. This has been solved by using certificateless cryptography combined with a Shamir's secret sharing based multi-party computation protocol. In this thesis, private and public

keys are already assumed to have been generated, see Section 3.1.2. Secret sharing is therefore used in order to regenerate these keys every time they are used.

Any of the nodes can be eavesdropped upon, more on this in Section 6.2.

6.2 Security analysis

In Section 3.3 a security model has been designed containing three attack scenarios that the system should be able to protect against. Each attack scenario considers a specific set of nodes to be corrupted i.e. the sets $\{KGC, P\}, \{V, P\}$, and $\{G, I\}$. The first set contains the nodes used in CLC, the second the nodes used in MPC, and the third the remaining nodes. All communication is done through http GET requests. In order to protect against eavesdropping, all communication should be encrypted. A viable option is therefore to use https instead of http.

6.2.1 Attack scenario: 1. $\{KGC, P\}$ in certificateless cryptography are corrupted

Attack type: KGC gets compromised

If the KGC gets compromised the greatest concern is that an adversary would be able to control and manipulate users' public and private keys. However, the certificateless public-key signature algorithm is based on a provably secure signature scheme [9].

In the first step in Section 4.3 the KGC sets the master public key s and the master private key k. Since the master keys are initially setup once an adversary would not be able to create an attack by choosing specific master keys. The KGC then creates a partial-private key D_A for user A by using k and ID_A. Since the algorithm is based on a provably secure scheme, nothing here can be used for an attack. In the next step, A computes their private key by using D_A and a secret. The reasoning behind this is that the KGC should not create the keys and will therefore not have access to them, instead each user creates their own private and public keys. This prevents yet another attack.

However, one attack vector that does exist is that a sender that wants to send a message to a receiver cannot be absolutely sure that

the receiver's public key is in fact the receiver's authentic public key. Usually, this is solved by using digital certificates and PKI. However, the reason for using certificateless cryptography is that a PKI is not available. A compromised KGC would therefore be able to convince the sender to use a receiver public key that has been generated by the KGC, and not by the receiver [6]. For this paper, this means a compromised KGC can impersonate a client from I, and would therefore be able to initiate the whole system legitimately.

Attack type: P in certificateless cryptography gets compromised

By compromising one (or several) parties used in the Shamir secret sharing scheme, the adversary wants control of the secret. By controlling k number of shares of the secret the adversary can regenerate the secret on their own.

The scheme should therefore be able to fulfil its tasks even if $t \leq n/2 = k - 1$ parties in P have been corrupted, a threshold recommended [21]. These parties are used in order to regenerate already created public and private keys. In order to regenerate the secret, k shares have to be collected. k can naturally differ between implementations, in either case, it allows for a certain subset of the parties to be corrupt. The keys can still be regenerated. For this thesis, $k = n/2 + 1$. If an adversary can control k shares it is possible to regenerate users' public and private keys, which in turn can be used to impersonate a client from I. By accomplishing this attack, an adversary would be able to properly authenticate as a client to any of the external nodes in E.

Another possible attack is that the adversary changes the share of a secret for a party P_i. In this case it is impossible to recover the secret, even if there are enough legitimate shares gathered, since only legitimate shares can be used when recovering the secret and it is impossible to verify the shares in Shamir's secret sharing.

Attack type: Both KGC and P gets compromised

In the case when an adversary can compromise both KGC and P, the impact is not greater than the impact already listed above for the individual attacks. The greatest impact on the two previous attacks is deemed to be through attacking the KGC, since a corrupted KGC could convince a sender to use a public key that has been generated

by the KGC. The greatest impact of this attack is therefore the same as attacking the KGC.

6.2.2 Attack scenario: 2. $\{V, P\}$ in the key regeneration protocol are corrupted

Attack type: V gets compromised

The vault V is considered to be a node in P but unlike a typical P_i, V will collect shares, make a computation on them, and then reconstruct the decryption key. A trivial attack is that V does not send the key to G, thus preventing the data from becoming decrypted. The key can instead be sent to someone else who can decrypt the configuration data.

Another trivial attack is that an attacker can corrupt V and listen for all the keys. Given enough requests from the set of internal clients I, all key pairs can be extracted and all configuration data can be decrypted.

Attack type: P in key regeneration protocol gets compromised

By compromising one (or several) parties used in the Shamir secret sharing scheme, the adversary wants control of the secret. By controlling k number of shares of the secret the adversary can regenerate the secret on their own.

In this attack type the parties in P gets compromised when regenerating the key used for encrypting sensitive data. For this, Shamir's secret sharing scheme will be used once again, where each party P_i gets a share of the secret. The shares are sent to the vault V which will reconstruct the key and then send it to G. If an adversary can control k shares it is impossible to regenerate the decryption key, since this is done at V and by no one else. However, an adversary could instead choose to not send the shares of the secret to V who would therefore not be able to regenerate the key.

It is also possible for an adversary to change each party P_i's share of the secret here and thus making sure it is impossible to recover the secret, since only legitimate shares can be used when recovering the secret and it is impossible to verify the shares in Shamir's secret sharing.

Attack type: Both V and P gets compromised

In the case when an adversary can compromise both V and P, the impact is not greater than the impacts already listed above for the individual attacks. The greatest impact on the two previous attacks is deemed to be through attacking V, since a corrupted V could listen for all key pairs or send nonsense to G. The greatest impact of this attack is therefore the same as attacking the V.

6.2.3 Attack scenario: 3. $\{G, I\}$ are corrupted

Attack type: G gets compromised

If G gets compromised and sensitive data is written in a configuration file stored at G, the adversary would have access to that data. This is solved by making sure nothing is stored at G and that any sensitive data is stored encrypted. This way no sensitive data is leaked if G gets compromised. Another problem with G becoming corrupted is that the adversary could initiate the key regeneration MPC protocol multiple times by sending requests to the set P and this way gather all data. This is solved by making sure that G can only act if it receives a legitimate request from I, which is solved by using certificateless cryptography. Lastly, with a high enough frequency of requests from I, G could collect all sensitive data by waiting and remembering everything that is being decrypted. This is a difficult problem to solve since if G gets compromised the adversary can read G's memory. Analysis of this attack will require future work.

Another problem with G becoming corrupted is that all decryption keys from V are sent to G. It is therefore possible for an adversary to simply remembering all the key pairs. However, that is a compromise for this solution and it is assumed to be magnitudes more difficult to read the exact right memory cells for a few microseconds than it is to retrieve a key stored at G.

Attack type: I gets compromised

If I_i gets compromised, the adversary can send arbitrary number of requests to G which will trigger requests to P, V, and eventually E_j.

6.3 Efficiency analysis

6.3.1 Benchmarks from jPBC

The jPBC library [5] contains benchmark scores to some of the operations for three different testbeds:

1. TestBed 1: Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz, 3 GB Ram, Ubuntu 10.04
2. TestBed 2: iMac Intel(R) Core(TM) 2 Duo @ 2.66GHz, 4 GB Ram, MacOSX 10.6.5
3. TestBed 3: HTC Desire HD A9191, Android 2.2. (Java Port)

For this thesis only the first two testbeds are assumed to be of interest. The proof-of-concept has the following mathematically heavy operations (as defined by the jPBC website):

1. Pairing#pairing(in1, in2), which is the bilinear map
2. Element#mul(BigInteger), which is a point on the curve multiplied by a big integer

These operations are only used for the CLC. The bilinear map was used 5 times in this algorithm, while the multiplication was used 12 times, which makes for a total of 17 mathematically operations per request from I to E.

Testbed 1

In the first testbed the bilinear map takes 14.654ms to execute. The multiplication takes 19.428ms. This makes for a total of $14.654 * 5 + 19.428 * 12 = 306.406$ms for testbed 1.

Testbed 2

In the second testbed the bilinear map takes 15.722ms to execute. The multiplication takes 17.869ms. This makes for a total of $15.722 * 5 + 17.869 * 12 = 293.038$ms for testbed 2.

6.3.2 Benchmarks from Shamir's secret sharing

The Github page for the secret sharing library has benchmark scores just like jPBC.

For $n = 4$ parties, where $k = 3$, it takes approximately 197 microseconds to recover the secret.

6.3.3 Performance data

The entire system has been executed on the following testbed:

- Operating System: Microsoft Windows 10 Home
- Processor: Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 2401 Mhz, 4 cores
- RAM memory: 8.00 GB

Table 6.1 shows the execution time after 500 executions, when $P = \{P_1, P_2, P_3\}$. It measures five different parts of the system, each having their own column:

1. MPC computation, the time it takes to regenerate the secret with the shares.
2. MPC, the time it takes to get the shares (through http requests) and regenerate the secret with them.
3. CLC computation, the time it takes for the whole CLC algorithm to execute excluding time to regenerate keys.
4. CLC, the complete CLC algorithm.
5. System, the time it takes to execute the whole system, from start to finish.

For each part of the system the table shows four different values, each on every row:

1. Avg time, the average running time of the part.
2. Std dev, the standard deviation.
3. Runs, number of runs on that part.

4. 95% level, the 95% confidence level, used to calculated the 95% confidence interval.

Table 6.2 shows when the number of parties has doubled: $P = \{P_1, P_2, P_3, P_4, P_5, P_6\}$. The keys used for the certificateless cryptography are all mod q, mentioned in Section 5.1.1.

	MPC comp(ns)	**MPC(ms)**	**CLC comp(ms)**	**CLC(ms)**	**System(ms)**
Avg time	54496.84	21.97	401.22	422.63	459.35
Std dev	15525.43	1.76	8.14	6.69	10.45
Runs	500	500	500	500	500
95% level	**1360**	**0.154**	**0.713**	**0.586**	**0.916**

Table 6.1: Performance data with 3 parties in the set P

	MPC comp(ns)	**MPC(ms)**	**CLC comp(ms)**	**CLC(ms)**	**System(ms)**
Avg time	91957.59	42.29	401.22	442.28	541.51
Std dev	83282.84	2.60	8.14	8.20	11.99
Runs	500	500	500	500	500
95% level	**7300**	**0.228**	**0.713**	**0.719**	**1.05**

Table 6.2: Performance data with 6 parties in the set P. Note that CLC comp has not changed since it is not affected by the size of P

From table 6.3 it is clear that, on the hardware this was tested on, 16 clients is an upper limit on how many clients can send requests to the system at the same time.

Figure 6.1 shows the running time for four out of the five parts listed when $P = 3$. MPC comp is not showed in the chart since it is in nanoseconds. Figure 6.2 shows the running time for the same four parts part when $P = 6$.

Number of clients	**Average time(ms)**
4	459.59
8	672.88
12	893.99
16	1080.75

Table 6.3: Number of clients and how long it takes to run the whole system when $P = 3$

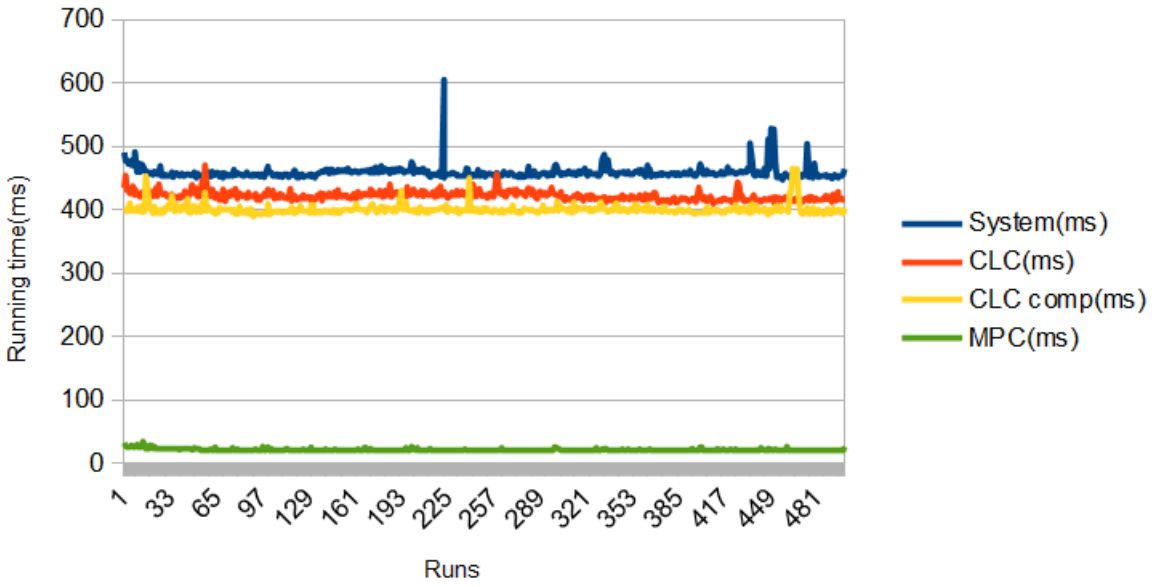

Figure 6.1: Running time for four out of five parts of the system when $P = 3$

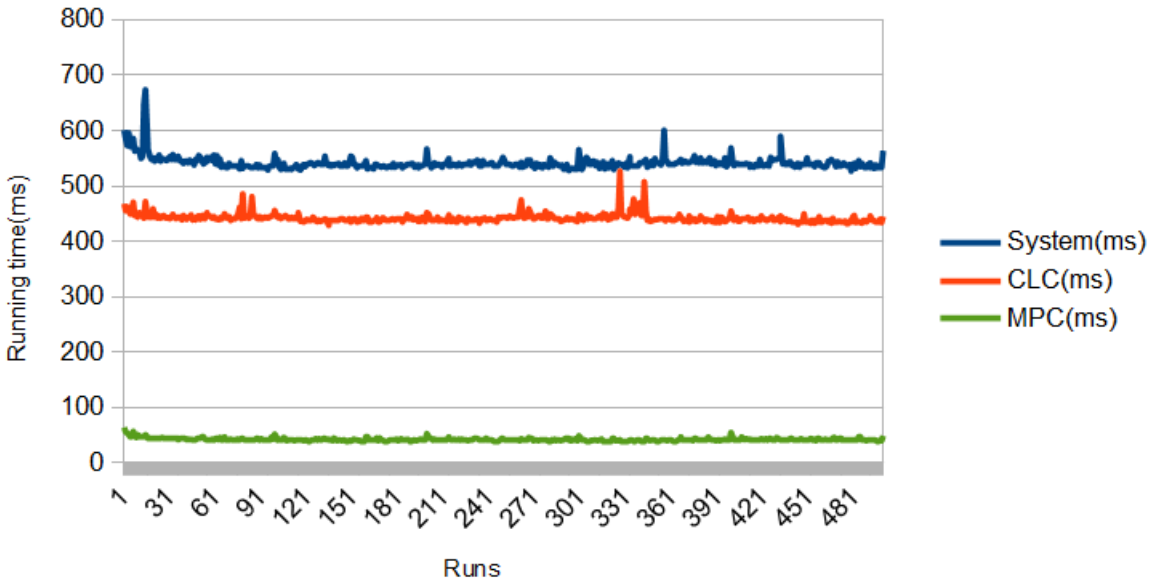

Figure 6.2: Running time for four out of five parts of the system when $P = 6$

Chapter 7

Discussions and conclusions

This chapter will discuss the results.

7.1 Solvability analysis

While the proposed system can handle many internal clients it is worth noting that the certificateless cryptography part can be very burdensome on the system, resulting in a huge computational overhead since several computationally hard mathematical calculations have to be carried out in order to sign and verify a message from one single client.

A secret sharing based MPC protocol is used for the certificateless cryptography. However, other algorithms such as BGW could work just as well or even better. The reasoning behind this is that the MPC protocol used in this thesis is a simplified variant. It is simple in its calculations since a secret sharing scheme does not require heavy mathematical operations. The heaviest mathematical operation is most likely the Lagrange interpolation. It is also simpler in code, meaning that a secret sharing scheme has easier theory to understand and is therefore easier to implement than an algorithm such as BGW, thus probably more open to optimization. A secret sharing scheme is also a fundamental part to most protocols. Since this simpler protocol works for this thesis, it is interesting to see if more complex algorithms would work, such as BGW. They would most likely prove to be more secure, since secret sharing is only a small step in the algorithm and other processes are executed on top of that to obfuscate the algorithm as much as possible. But it would also be more difficult to implement.

7.2 Security analysis

7.2.1 System analysis

An interesting aspect to discuss is step 4 in Section 3.2, mentioning that V should send the decryption key to G. There are two possibilities for this step. The first is the one that has been mentioned in this thesis: V collects all shares, makes a computation on them, and regenerates the decryption key. This key is used to decrypt the configuration data stored at G. However, another possibility is for V to regenerate the key and then decrypt the data, which is then sent to G. It is a small but important difference. In the second case, V decrypts the data and not G.

If V decrypts the data and then sends it to G it must mean that G has to trust that whatever V sends is legitimate. If an attackers corrupts V they could simply send nonsense data to G or send the correct data to someone else.

If G decrypts the data all key pairs will be sent from V to G. It is therefore possible for an adversary to corrupt G and collect all keys being sent to G. Given enough requests from I, all key pairs can be stolen. However, it is difficult for an adversary to get ahold of this key if it is stored in memory. Especially if it is stored in memory for a short period of time, such as a few microseconds. Furthermore, if an adversary wants more than just one key pair, it will also be a time consuming process to get ahold of every key pair. By combining these two elements it is deemed to be a quite difficult attack to carry out. In either case, it is also considered to be a compromise for the solution proposed by this thesis. It is greatly more difficult compromising G and trying to listen to all key pairs than it is to read a file where all data is written in plain text. By having V only sending the decryption key to G it is also easier to guarantee confidentiality of the data. As mentioned above it is still possible for an adversary to gain access to the data, meaning it is not possible to guarantee confidentiality on the data given the attacker model in Section 3.3. However, the adversary in that model has a great deal of freedom and power, since it can compromise any nodes in the system. If a model with less power would have been chosen it would also be possible to guarantee confidentiality.

The threshold k in the secret sharing scheme can naturally be set to

any number. If k is set to a high enough number, the adversary would have to corrupt multiple parties which is more difficult then simply corrupting one. However, if k is chosen to be too large, such as $n-1$, an adversary could simply corrupt a small number of parties, e.g. two, in order to prevent a legitimate use of the secret sharing scheme since there will only be $n-2$ legitimate parties but s is set to $n-1$. Therefore, this thesis went with Shamir's recommendation of setting $n = 2k-1$ when the set P contains three parties.

Summary of attacks

KGC

A sender cannot be absolutely convinced that the receiver's public key is in fact the receiver's public key. As mentioned by Alexander Dent [6] "By definition, the KGC can compute all partial private keys; hence, it can always replace the public key with one for which it knows the underlying secret value and therefore decrypt all ciphertexts computed by the sender". For this paper this means that the KGC can sign any message. It would therefore be possible to impersonate a client I. It is worth noting that Section 3.1.2 states that all keys for the proposed system should be pre-generated. Therefore, the KGC should never have to create keys, not even partial private keys. They should already be generated. However, the keys have to be generated somehow at some point and if they are generated by following steps 1-5 in Section 4.3 this attack vector is possible. If the keys are generated in a safe manner, this attack should not be feasible to carry out.

P in CLC

If an adversary controls k shares of the secret it is possible to regenerate any user's public and private keys. In this system the users are client in I. An attacker carrying out this attack could therefore impersonate clients and legitimately authenticating themselves to any E_i. It is worth noting that k can differ between implementations but is usually $k = n/2 + 1$. Thus, if there are three parties, k is two, meaning, an adversary have to corrupt two nodes and not one. The more the parties, the harder it is to carry out this attack.

An adversary could corrupt P_i and change that party's secret. It would not be possible to regenerate the secret if invalid shares are used.

V

By compromising V the attacker can choose to not send the decryption key to G, but to someone else, and instead send nonsense or nothing to G. G would of course notice this. The real threat here though is that the attacker would have the decryption key and could break the confidentiality of the data by decrypting it. Naturally, the configuration data is located at G. The attacker would therefore have to also corrupt G in order to gain access to the data.

P in key regeneration

By controlling k shares of the secret it is impossible for the adversary to regenerate the decryption key, since this is done by V only. However, an adversary could choose to not send the shares to V. This way V would not be able to regenerate the keys and cannot continue with the protocol.

An adversary could also corrupt P_i and change that party's secret. It would not be possible to regenerate the secret if invalid shares are used.

G

G is the greatest target in the system, since the configuration data is located at G and other attacks rely on also corrupting G.

An adversary who compromises G could remember all the keys being sent from V to G. That involves reading the memory cells for a few microseconds and is considered difficult to carry out. Another possible attack is that an attacker who corrupts G can also store memory content to a file or send a copy to someone else.

In summary, since it is not possible to be completely protected against G being compromised it is vital to strengthen the system security in order to prevent intrusions to G and to restrict what can be executed on G.

I

If I gets compromised it will be possible for an adversary to send requests to G. In the end, this means that the corrupt node can authenticate to nodes in E.

7.2.2 Feldman verifiable secret sharing scheme

By using a verifiable secret sharing scheme such as Feldman verifiable secret sharing instead of Shamir's secret sharing scheme, each party can verify their share. This adds another level of security to the protocol.

7.2.3 Encrypted communication

As mentioned in 6.2, an adversary could eavesdrop on the communication, for example when P sends their shares to V. For this encrypted communication has to be used, for example https.

7.3 Efficiency analysis

The benchmark scores from jPBC give a rough estimation of the practical lower limit for this thesis' CLC implementation. The number of bilinear maps and multiplications have been kept to a minimum in order to decrease the computational overhead as much as possible.

By studying table 6.1 it is clear that the MPC protocol constitutes the lower portion of the running time. It is also clear that what takes the most time in the MPC protocol is the http requests, as demonstrated clearly by the first and the second columns in table 6.1 and 6.2. Without the http requests it takes approximately 0.05ms to regenerate a secret with 3 shares. With 6 shares, this time is almost doubled. However, with http requests this time goes up to 22ms for 3 parties, and to 42ms for 6 parties. Again, the time going from 3 parties to 6 has doubled, suggesting the overhead is linear. This seems to add up nicely with the benchmarks received from the Shamir's secret sharing repository on Github, which stated that for $n = 4, k = 3$ it takes roughly 197 microseconds to recover the secret. For this thesis $n = 3, k = 2$ took 55 microseconds, and $n = 6, k = 2$ took 92 microseconds. If a higher k would have been chosen, the time would match the benchmark more closely.

What is also clear from table 6.1 and 6.2 is that certificateless cryptography takes up the most time, around 400ms (without the MPC part). This is not a surprising result, seeing how it contains difficult mathematical operations. Additionally, the benchmark from jPBC gave a clue regarding how well the algorithm would perform, roughly

300ms. However, that still means 100ms got added somehow. This can be explained by several factors:

1. The jPBC benchmarks only had data for two different operations. However, the CLC algorithm contains more than just two different operations. The two noted by jPBC would most likely take up the most time to execute, since they are the mathematically heaviest operations. However, the smaller operations also adds up in the end.

2. The experiments were done on a hardware executing other tasks than just these experiments.

Overall the system finished in approximately 460ms for 3 parties, and 540 for 6. This passes the efficiency requirement stated in Section 4.4. For companies and enterprises the hardware that would run this system would be better. Therefore the total execution time would also be lower. If PKI is used instead of CLC the time would also be lowered since PKI does not require mathematically heavy operations to perform.

Table 6.3 gives an approximation to how many clients can send requests to the system at the same time. From these results it is clear that the time increases with roughly 200ms for every 4 clients, suggesting a linear overhead. These results are also very dependent on the hardware they run on and can therefore be decreased.

7.4 Ethical aspect and sustainability

The proposed system has the main goal of protecting vital information in a way that only authorized users can gain access to it. As stated in Section 1.1, the confidentiality of the data is of the highest priority for this solution. By having the sensitive data located at G, only G can gain access to it. In this way the confidentiality of the data is not broken and therefore any morally questionable actions connected with the data is kept to a minimum. However, there are still ways for adversaries to gain access to the data even though they are not authenticated as G, as mentioned in Section 7.2. Nevertheless, the proposed solution increases the difficulty and complexity of carrying out such attacks.

There is nothing specific in this thesis that would point to the fact that the system is not sustainable. The proof-of-concept can be setup at

home with minimal hardware and can also be scaled up to the needs of companies. The proposed system does not have an energy consumption problem or similar.

7.5 Conclusion

The proposed system passes all the evaluation criteria from Section 4.4 except for guaranteeing the confidentiality of the data at all times. As mentioned in Section 7.2.1 there are attacks on the vault V that gives an attacker access to the data. These attacks are deemed to be of the more difficult kind to carry out in this thesis. Therefore confidentiality can still be guaranteed given an attacker with less freedom over V. In terms of performance, the bulk of the overhead comes from certificateless cryptography, a constraint for the specific scenario which might not be present in general. As a first step toward a full security evaluation, an informal analysis points to major improvements over solutions that store passwords/keys and identifies areas of future work.

This work has showed that jPBC is a mature enough library that is efficient enough for protocols relying on bilinear maps.

However, the thesis has also showed that there seems to be a need for more Java libraries for cryptographical protocols, especially for MPC and CLC. One of the reasons for not implementing a true MPC protocol, such as BGW, is that there really is no easy way to do it currently in Java, and would therefore require too much work for a master thesis. Since the system proposed in this thesis is successful in practical terms there are incentives for evolving these kinds of Java libraries more. Perhaps even add it to BouncyCastle or as a standard to the Java cryptography libraries.

Chapter 8

Future work

It is of interest to research if MPC protocols such as BGW could be used for the certificateless cryptography. For example, this thesis did not look into the first five steps of the certificateless signature scheme. However, when creating private and public keys BGW could be used since it allows for addition and multiplication.

Two critical vulnerabilities in modern processors were discovered in January 2018, known as Meltdown [15] and Spectre [13]. Since some of the attack types from Section 6.2 are based on reading memory it is important to get a better understanding for how these types of attacks can be avoided in order to better secure the system.

In this thesis, secret sharing was used to regenerate the key which could decrypt the sensitive data. It is of interest to research what other MPC based protocols could be used here as well.

Multiple interesting questions regarding keys exists, such as key revocation or updating sensitive data. In the first case, the keys used for encrypting the sensitive data might be revoked and the system should be able to handle this. In the second case, the sensitive data might be updated. How would this affect the number of keys?

A formal security analysis is also needed.

It is still an open research question if it is possible to solve the problem without exposing a key at any point of time.

Bibliography

[1] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. "Foundations of garbled circuits". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM. 2012, pp. 784–796.

[2] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness theorems for non-cryptographic fault-tolerant distributed computation". In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM. 1988, pp. 1–10.

[3] Peter Bogetoft et al. "Secure multiparty computation goes live". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2009, pp. 325–343.

[4] Ronald Cramer, Bjerne Damgård Ivan, and Buus Nielsen Jesper. *Secure multiparty computation and secret sharing*. Cambridge University Press, 2015. ISBN: 9781107043053.

[5] Angelo De Caro and Vincenzo Iovino. "jPBC: Java pairing based cryptography". In: *Computers and communications (ISCC), 2011 IEEE Symposium on*. IEEE. 2011, pp. 850–855.

[6] Alexander W Dent. "A brief introduction to certificateless encryption schemes and their infrastructures". In: *European Public Key Infrastructure Workshop*. Springer. 2009, pp. 1–16.

[7] Wenliang Du and Mikhail J Atallah. "Secure multi-party computation problems and their applications: a review and open problems". In: *Proceedings of the 2001 workshop on New security paradigms*. ACM. 2001, pp. 13–22.

[8] Paul Feldman. "A practical scheme for non-interactive verifiable secret sharing". In: *Foundations of Computer Science, 1987., 28th Annual Symposium on*. IEEE. 1987, pp. 427–438.

[9] Florian Hess. "Efficient Identity Based Signature Schemes Based on Pairings". In: *Selected Areas in Cryptography*. Ed. by Kaisa Nyberg and Howard Heys. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 310–324. ISBN: 978-3-540-36492-4.

[10] Vivek Kapoor, Vivek Sonny Abraham, and Ramesh Singh. "Elliptic curve cryptography". In: *Ubiquity* 2008.May (2008), p. 7.

[11] Jonathan Katz et al. *Handbook of applied cryptography*. CRC press, 1996.

[12] Neal Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of computation* 48.177 (1987), pp. 203–209.

[13] Paul Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *ArXiv e-prints* (Jan. 2018). arXiv: 1801.01203.

[14] Yehuda Lindell and Benny Pinkas. "Secure multiparty computation for privacy-preserving data mining". In: *Journal of Privacy and Confidentiality* 1.1 (2009), p. 5.

[15] Moritz Lipp et al. "Meltdown". In: *ArXiv e-prints* (Jan. 2018). arXiv: 1801.01207.

[16] Philip MacKenzie, Thomas Shrimpton, and Markus Jakobsson. "Threshold password-authenticated key exchange". In: *Annual International Cryptology Conference*. Springer. 2002, pp. 385–400.

[17] Victor S Miller. "Use of elliptic curves in cryptography". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1985, pp. 417–426.

[18] Divya G Nair, VP Binu, and G Santhosh Kumar. "An effective private data storage and retrieval system using secret sharing scheme based on secure multi-party computation". In: *arXiv preprint arXiv:1502.07994* (2015).

[19] Claudio Orlandi. "Is multiparty computation any good in practice?" In: *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE. 2011, pp. 5848–5851.

[20] Sattam Al-Riyami. "Cryptographic schemes based on elliptic curve pairings". PhD thesis. University of London, 2004.

[21] Adi Shamir. "How to share a secret". In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[22] *Shamir's Secret Sharing*. [Online; accessed 9-May-2018]. 2017. URL: https://github.com/codahale/shamir.

[23] Spring. *Spring Framework Overview*. [Online; accessed 9-May-2018]. 2018. URL: https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html.

[24] Stanford. *Type A internals*. [Online; accessed 17-May-2018]. URL: https://crypto.stanford.edu/pbc/manual/ch08s03.html.

[25] Tiobe. *TIOBE Index for May 2018*. [Online; accessed 9-May-2018]. 2018. URL: https://www.tiobe.com/tiobe-index/.

[26] Andrew C Yao. "Protocols for secure computations". In: *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*. IEEE. 1982, pp. 160–164.