A Blockchain-based Signature Scheme for Dynamic Coalitions

Ricky A. Sewsingh

July 2018

A Blockchain-based Signature Scheme for Dynamic Coalitions

by

Ricky A. Sewsingh

to obtain the degree of Master of Science in Computer Science
Software Technology Track
with a specialisation in Cyber Security
at the Delft University of Technology,
to be defended publicly on Monday July 30, 2018

Student number:	4230981	
Thesis committee:	Assoc. Prof. Dr. Ir. J.C.A. van der Lubbe,	TU Delft
	Assoc. Prof. Dr. Ir. J. Weber,	TU Delft
	M.J. de Boer, MSc,	Water Authority Hollands Noorderkwartier
	Dr. Z. Ren,	TU Delft
Supervisor:	Assoc. Prof. Dr. Ir. J.C.A. van der Lubbe,	TU Delft

TU Delft

Abstract

During military or peacekeeping operations it is often the case that coalitions are formed between nations or instances to achieve common objectives by joint decision making and resource sharing. Often a coalition partner participates for a certain period of time and other nations or instances might want to join the coalition in a later stage, resulting in a dynamic environment. In practice the members in these coalitions are dissimilar with regards to their disposition, requiring the coalition to be set up in a decentralised manner. In order to facilitate such an environment, a secure decentralised network should be in place that enables decision making, access control to shared information and the ability to join and leave in an appropriate way. To enable the coalition to put its signature on important decisions and to be able to provide certified public keys to enable access control, the network should include a signature scheme. A challenge for such a signature scheme is that it should not be able for previous compositions of the coalition to create valid signatures. In other words, Backlogging should be prevented in the system. *Van der Lubbe et al.*[25] introduced a distributed (n,n) signature scheme for dynamic coalitions based on the distributed El Gamal signature scheme of *Park et al.*[21], which can be expanded to a $(n+1,n+1)$ signature scheme. The scheme prevents backlogging by the use of two Oblivious Third Parties. *Van Elsas et al.*[26] then improved this signature scheme by enabling the signature scheme to be reduced to a $(n-1,n-1)$ signature scheme and by replacing the Oblivious Third Parties by One-Way Accumulators, dismissing the need for Third Parties in general. Additionally, the currently popular *Blockchain* model[17] has the capability to perform as a *ledger*. The ledger characteristics, specifically the chain of blocks, the Merkle Tree model and a consensus mechanism, can replace the functionality that prevents backlogging.

In this thesis we present a *Blockchain-based Signature Scheme for Dynamic Coalitions*. The signature scheme is a distributed (n,n)-signature scheme, that is able to expand to a distributed $(n+1,n+1)$-signature scheme and is able to reduce to a distributed $(n-1,n-1)$-signature scheme. The signature scheme uses the Blockchain model to tackle Backlogging Problem. In addition, we provide an analysis of aspects of the signature scheme that could introduce problems when implementing it.

Yesterday is history, tomorrow is a mystery, but today is a gift. That is why it is called the Present.

- Master Oogway
Kung-Fu Panda

All we have to decide is what to do with the time that is given us.

- Gandalf
Lord of the Rings

Contents

1 Introduction

In a military environment, operations are executed to achieve objectives that are beneficial to the parties involved. During these operations it is often the case that several independent parties require to work together. Hence coalitions are formed between these parties, enabling joint decision making and resource sharing. Additionally, It is common that a coalition partner participates in such a joint operation for a certain period of time and that other nations or instances might want to join the coalition in a later stage to assist in the operations. This makes the coalition *dynamic*. Because of the military nature of the environment, in practice the members in these coalitions are also dissimilar with regards to their disposition. This results in a trust barrier between the members. Hence, it is required for a coalition to be set up in a decentralised manner, where all coalition partners are equal and have the same rights. In order to facilitate such an environment, a secure decentralised network should be in place that enables decision making, access control to shared information and the ability to join and leave in an appropriate way. It is important that the coalition is able to achieve consensus regarding missions and its details and is able to place a joint signature on important documents in an appropriate manner. This requires the network to include a signature scheme that meets the requirements of the dynamic environment. A big challenge for such a signature scheme is that it should not be able for previous compositions of the coalition to create valid signatures. In other words, *backlogging* should be prevented in the system.

Different signature schemes have been introduced that facilitate the needs of the described environment. *Van der Lubbe et al.*[25] introduced a distributed (n, n) signature scheme for dynamic coalitions based on the distributed *El Gamal* signature scheme of *Park et al.*[21], which can be expanded to a $(n+1, n+1)$ signature scheme. The scheme prevents backlogging by using two *Oblivious Third Parties*. *Van Elsas et al.*[26] then improved this signature scheme by enabling the signature scheme to be reduced to a $(n-1, n-1)$ signature scheme and by replacing the Oblivious Third Parties by *One-Way Accumulators*, dismissing the need for Third Parties in general. Additionally, *Van Elsas et al.* introduced ledger functionalities to tackle the backlogging problem. However this signature scheme does not fully utilise the potential of the One-Way Accumulators and the ledger capabilities. The currently popular *Blockchain* model[17] has the capability to perform as a *ledger*. The ledger characteristics, specifically the chain of blocks, the Merkle Tree model and a consensus mechanism, can be used to efficiently tackle the Backlogging problem.

In this thesis we aim to design a (n, n) signature scheme that is based on the Blockchain model for a secure dynamic coalition environment. Additionally, aspects regarding the implementation of this signature scheme are evaluated and discussed.

1.1. Dynamic Coalition Environment

The term *Dynamic Coalition* is a concept that goes with an environment. In this *coalition environment*, entities require to work together in order to achieve a common goal[11]. A Dynamic Coalition is the coalition formed in this environment with the characteristic that during the lifespan of the coalition, parties may leave the coalition and other parties may join the coalition.

This is best explained using an example. Let's take the scenario where a town exists in a hostile environment. Due to the situation, there is a shortage of food and medicine in this town. The *United States* has a military unit present in the area and decides to execute an operation to provide the required food and medicine. However the *United Kingdom* also has a military unit present in the area and wishes to participate in this operation. Hence the United States and the United Kingdom form a coalition and jointly execute this operation. A week later, *France* deploys a military unit in this area and heard about the peacekeeping operation in the aforementioned town. France would like to assist in this operation and provide extra help. The coalition is therefore *expanded* to include France. A couple days later, the United States decide to remove their military unit from the area and can therefore not participate in the operation anymore. The coalition is therefore *reduced* to exclude the United States.

In the scenario, a *Dynamic Coalition* is required to be present, to facilitate the need of cooperation and the dynamism of the environment.

1.2. Problem Scenario

The scenario we base our research around has several characteristics. The scenario has a *military setting*. This means that the system should be secure, since there is a high probability that the system will be targeted by people with malicious intent. Also in a military setting, trust is a big issue. It is very likely that the members of the coalition are independent organisations or countries who do not fully trust each other, but do want to collaborate. This means the system should also be *decentralised*, where each member has the same influence and equal involvement.

The scenario involves *mission agreement*. Collaborating members, for example countries or military instances, are working together in a (mostly) hostile area. In this area, missions have to be executed. It is important for the collaborating members to ensure that everyone agrees on a mission and its details. To ensure this is the case, each mission is signed by all collaborating members using a signature scheme that requires all collaborating members to cooperate in order to create a valid signature. Since all members are needed to create valid signature, if a mission is signed, it is ensured that there is consensus between the members and therefore the mission can be executed safely. Also, if after signing a mission, a member claims it did not approve of this mission, the signature can be used to verify that the member in question did approve of the mission. Lastly, because in order to create a valid signature all members are needed, changes can not be made to a mission without the knowledge and agreement of all members.

1.3. Signature Schemes & Backlogging

A *Signature Scheme* allows a user to sign a message, resulting in a signature[9]. The signature is constructed in such a way, that all users can verify the validity of this signature. A signature scheme uses a keypair consisting of a *secret key* and a *public key*, where the secret key is used to construct signatures and the public key is used to verify the validity of a signature.

A signature scheme that facilitate the needs of a dynamic coalition, needs to be able to handle multiple users. Such signature schemes are referred to as *Multisignature Schemes*[5]. Multisignature schemes enable multiple users to jointly create a signature, where each user holds a part of the secret key. But in order to work in a dynamic environment, the signature scheme needs to be *expandable* and *reducible*. This makes it possible for users to join and leave the coalition.

However, this requirement introduces a new problem. Let's take the scenario where a user joins the coalition, so the coalition is expanded. This requires the signature scheme to be expanded, to include the new member. However, it should not be possible for previous compositions of the coalition to create valid signatures. This is referred to as the *Backlogging Problem*. Because the previous composition was able to create signatures, it is easy to understand that this will still be possible after a group change has occurred, unless specific measures have been taken. A full explanation of the Backlogging problem is provided in Section 2.3.

1.4. Blockchain Technology

The *Blockchain Technology* was first introduced by *Nakamoto*[17]. It is explained to be a distributed database for transaction processing. Information stored on a blockchain is stored in a distributed ledger, owned by all parties in the network. This information is made immutable using some consensus mechanism, ensuring consensus between the parties in the network about the information stored on the blockchain. It can be said that a blockchain functions as a *distributed ledger*. Additionally each block in a blockchain is linked to the previously created block, creating a *chain of blocks*.

1.5. Research Question

The problem scenario explains the need of a signature scheme that is able to perform in a dynamic coalition environment. The signature scheme needs to handle multiple users and needs to be expandable and reducible. Additionally, the issue of backlogging needs to be tackled. Because of the ledger capabilities, Blockchain technology can be used as a decentralised solution to this problem.

Hence, the main research question for this research can be defined as follows:

How to construct a signature scheme that facilitates the needs of a dynamic coalition environment, and tackles the Backlogging problem by employing Blockchain technology?

To understand the problems that needs to be tackled, the research question can be divided into two parts:

1. *How to construct a signature scheme that facilitate the needs of a dynamic coalition environment?*
2. *How to employ Blockchain technology to tackle the Backlogging problem?*

1.5.1. Requirements

Based on the research question, a set of requirements has been created regarding the flexibility and functionality of the signature scheme.

1. At initialisation the dynamic coalition consists of n members, which jointly generate a distributed secret key and a public key using a suitable protocol.
2. The n members are able to create a signature together using a suitable (n,n)-signature scheme. This scheme makes it possible to create a signature without revealing the secret keyshares. Note that it should only be possible to create a valid signature if all n members participate in the signing procedure. This ensures that the members of the dynamic coalition have a veto.
3. It must be possible to expand the (n,n)-signature scheme to a $(n+1,n+1)$-signature scheme. The new member generates a new secret keyshare. To retain the original secret keyshares, the secret keyshares must be combined with the secret keyshare of the new member to generate a new distributed secret key and public key.
4. It must be possible to reduce the (n,n)-signature scheme to a $(n-1,n-1)$-signature scheme. The original secret keyshares, excluding the secret keyshare of the leaving member, must be used to generate a new distributed secret key and public key.
5. After a change has occurred in the composition of the coalition, either an addition or removal of a member, the n old members may not be able to create a valid signature. In other words, instances of backlogging must be prevented. One can assume that no member will give his secret keyshare to another member, since this can be used against him.
6. It must be impossible for any member to find the entire secret key throughout the entire lifetime of the dynamic coalition.

1.6. Contributions

Our contributions are as follows:

1. We present a *Blockchain-based Signature Scheme for Dynamic Coalitions.* The signature scheme allows the participation of multiple users. The signature scheme is expandable, allowing a user to join in a secure way, and is reducible, allowing a user to leave in a secure way. Additionally, Blockchain Technology is implemented in the signature scheme to prevent previous compositions of the coalition to create valid signatures.
2. We provide an analysis of the implementation aspects for our signature scheme. This regards aspects that could introduce issues when implementing our signature scheme.
3. We provide a validation of the requirements set for our signature scheme and an analysis of the security of our signature scheme.
4. We published a paper at the *2018 Symposium on Information Theory and Signal Processing in the Benelux* named: *"A Blockchain-based Signature Scheme for Dynamic Coalitions".* The paper discusses our proposed signature scheme and can be found in the proceedings of the symposium [23].

1.7. Report Outline

This thesis report is structured in the following way: In the first part of this report the dynamic coalition environment is explained, the problem scenario is provided, the backlogging problem is introduced, and the goal of this research is stated. In Chapter 2, background knowledge is provided to understand our signature scheme. In Chapter 3, prior art related to this research topic is discussed. Our signature scheme is presented in Chapter 4. The basic idea is stated, followed by a detailed description including protocol definitions. Chapter 5 provides an analysis of implementation aspects for our signature scheme. In Chapter 6 we provide a complexity analysis for the Merkle Tree and discuss validation for our signature scheme. Chapter 7 contains a discussion regarding certain aspects and decision-making for our signature scheme. Finally, in Chapter 8 we revisit the research question and provide an outlook for future work.

2 Background

To understand the Blockchain-based Signature Scheme for Dynamic Coalitions, it is necessary to possess some background knowledge. In this chapter, this background knowledge is provided. First, we discuss some cryptographic primitives, namely the types and implementations of the signature schemes that are important to understand our signature scheme and Public-Key Infrastructures. Then, the Blockchain technology is explained. Key characteristics of this technology are highlighted. And lastly, in addition to section 1.3, a more extensive explanation and example of the Backlogging problem is provided.

2.1. Cryptographic Primitives

2.1.1. Signature Schemes

A signature scheme is used to generate verifiable signatures for chosen messages. A signature scheme contains atleast three protocols: the *Key Generation protocol*, the *Signing Protocol* and the *Verification protocol*[12]. The Key Generation protocol is used to generate in polynomial time a keypair (x, y) of matching secret key x and public key y. The Signing protocol uses the secret key to generate a signature for some message m. The Verification protocol is then able to check whether a signature is valid for some message m, using the public key.

Multisignature Schemes

A type of signature scheme that allows the participation of multiple users is the *Multisignature scheme*[5]. Here each user of the signature scheme holds a part of the secret key. By jointly participating in the signing protocol, a valid group signature can be generated. Any user can then verify the validity of the signature using the public key of the group. Multisignature schemes can be divided into two categories. First there are the multisignature schemes with *Serial Signing Order*. For serial signing, the signing order can be detected by a verifier from a signature. Different signing orders result in different multisignatures. For parallel signing, the signing order cannot be detected by a verifier from a signature. The multisignature is group independent.

Threshold Signature Schemes

A form of a multisignature scheme with parallel signing order is the *Threshold Signature Scheme*. This concept was first introduced by *Desmedt*[8] in 1987. In a (t, n)-threshold signature scheme, a group signature can be created when atleast t out of n members participate in the signing protocol. Again, each member holds a part of the secret key and any user can then verify this group signature by using the public key of the group.

El Gamal type Signature Schemes

In 1985 *El Gamal* introduced a public key cryptosystem and signature scheme based on *Discrete Logarithms*[9]. The security of both systems relies on the difficulty of computing discrete logarithms over finite fields. Here the keypair can be defined as follows: for a secret key x, let p be a large prime and g be a primitive element $mod p$, now the public key can be defined as $y = g^x mod p$. It is also explained that p needs to be chosen such that $p-1$ has at least one prime factor, else computing discrete logarithms is easy.

Park and Kurosawa's Threshold Signature Scheme

Park and Kurosawa introduced an El Gamal Type Threshold Signature Scheme in 1996[21]. The paper first introduces a variant of the *Digital Signature Standard*[14]. This is an El Gamal type signature scheme which requires only a linear combination of two shared secrets when applied to the (t, n)-threshold scenario. Then they modify this signature scheme to a (t, n)-threshold signature scheme.

Their variant of the DSS is formulated as follows. Let p and q be large primes such that q divides $p-1$ and g generates the subgroup G_q of $\mathcal{Z}_p$ of order q. Let h be a one way hash function whose range is $\{1, ..., q-1\}$.

Public Key $p, q, g, y (= g^x \text{ mod p})$

Secret Key $x \in \mathcal{Z}_q$

Signature (t, w) such that
$w = (g^e \text{ mod p}) \text{ mod q}$
$t = wx + h(m)e \text{ mod q}$
where $e \in \mathcal{Z}_q$ is a random number.

Verification (t, w) is a valid signature for m if and only if
$w = (g^{t/h(m)} y^{-w/h(m)} \text{ mod } p) \text{ mod } q$

2.1.2. Public-Key Infrastructure (PKI)

In Public-Key Cryptography, the encryption key and decryption key are two separate keys. The encryption key is made public, therefore called the public key, and can be used widely to encrypt messages. The decryption key is kept secret to the owner, therefore called the secret key, and is used to decrypt the messages. This is useful if person A wants to send a message to person B and wants to ensure that only person B is able to read the message.

However, to ensure that a public key is then actually owned by specific person, it necessary to have a verifiable connection between the public key and a person. This is what a Public-key Infrastructure is used for. A Public-key Infrastructure enables the connection between a public key and a identity. A so called *Certificate Authority (CA)* is able to issue certificates which states that a public key belongs to an identity. Signatures, as explained in the previous subsection, can be used as certificates. By signing a message that includes the public key and the identity, a verifiable signature can be created. This signature then functions as a certificate. It is important to understand that this Certificate Authority is a Trusted Third Party. If the CA is not trusted, the certificate can be deemed invalid by a verifier.

2.2. Blockchain Technology

The *Blockchain* model is an interesting technology that was first introduced by *Nakamoto*[17]. Because of the success of *Bitcoin*, people try to utilise Blockchain technologies in many fields.

The basic goal of the blockchain is to create a distributed database where every user in the system agrees on the content of this database in a trustless manner. The database is defined as a ledger consisting of blocks. These blocks contain the information that needs to be stored on the blockchain. Every now and then, a block is defined and information is stored on that block. Then, using a proper consensus mechanism, an attempt is made to achieve consensus between the users of the system regarding the contents of the block. If consensus is achieved, the block can be added to the blockchain. When a block is defined and when it is added to the block is dependent on the implementation of the Blockchain Technology.

Key Characteristics of the Blockchain

Chain of Blocks One of the key characteristics in the Blockchain model is the actual chain of blocks. Each block contains a pointer to its previous block, hence creating a chain. This basic idea is a defining factor in the model and can be used very efficiently.

Merkle Tree The concept of the Merkle Tree data structure is quite simple. The idea is to hash several items until a single hash remains which is called the *Merkle Root*[17]. The algorithm starts by hashing all items. Next these hashes are paired. Each pair is combined and then hashed again, creating a single hash. Then the remaining hashes are again paired, combined and hashed. This is continued until one hash remains, see Figure 2.1. Here m_i is an item to be stored and h_i is a hash.

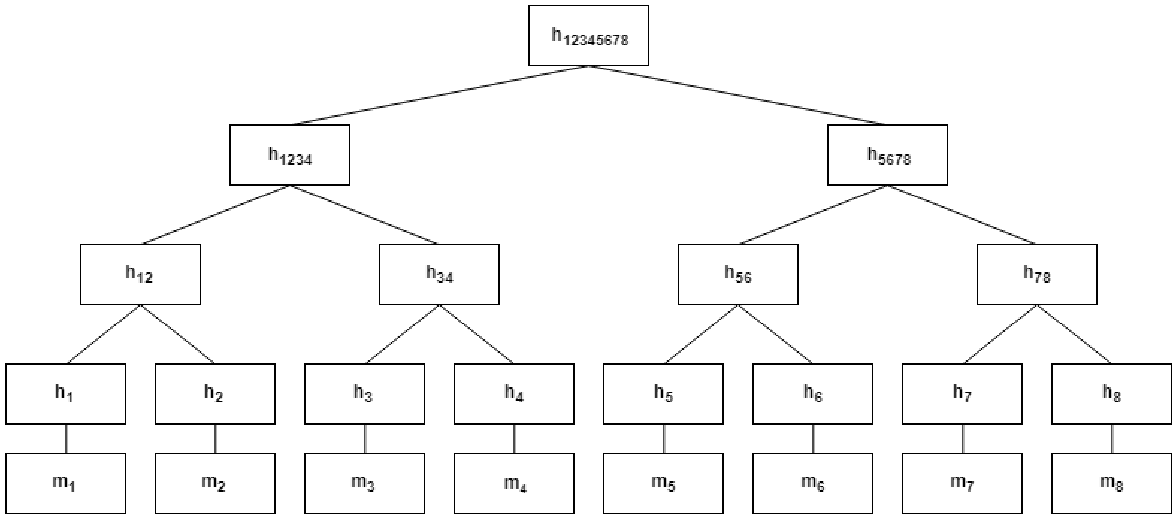


Figure 2.1: Example of a Merkle Tree

In the original idea of blockchain created for Bitcoin, this concept is used to store the transactions in a block. In order to prove that a transaction is contained in the block, one needs to show that the hash of the transaction can be used to create the Merkle Root. In other words, this is used to prove that a transaction is part of the group of transactions in the corresponding block.

Consensus Mechanism In the original blockchain paper from Nakamoto[17], which forms the basis of the *Bitcoin cryptocurrency*, the consensus mechanism used is called *Proof-of-Work* and is explained as follows. When a block is created, a timestamp needs to be defined. This is done by first adding a nonce to the block before hashing it. The resulting hash needs to have a certain amount of leading zero bits. The nonce is changed until a hash is found with the required leading zero bits. This hash is then called the *Proof-of-Work*.

This idea works based on the assumption that the majority of the miners, measured in terms of computational power to solve the Proof-of-Work puzzles, are honest and try to extend the longest existing chain[15]. The security is then guaranteed because the majority will be extending the honest chain faster than any corrupt minority that might try to change previous blocks or double-spend currency, due to the higher computational power. Therefore, in case there are multiple chains due to forking, the longest chain is always considered valid.

However, in the Bitcoin context, the moment a block is created, all miners immediately start working on the Proof-of-Work for the new block. This is an important *requirement* for this concept to work. If this is not the case, corrupt miners could start changing previous blocks or double-spend currency, while the honest miners are idle.

2.3. The Backlogging Problem

When a signature scheme for a dynamic coalition environment is in place, a problem is introduced. If a certain coalition exists, the ability for a non-active composition of this coalition to create valid signatures is referred to as *Backlogging*. This is a concept that needs to be avoided, since this is unwanted behaviour. An example will be provided to understand how backlogging works in practice and to understand why this is unwanted behaviour.

Let's take the scenario where the coalition ABC exists, containing the members A, B and C. The coalition uses an appropriate distributed signature scheme, so each member contains a part of the secret key and a proper signing protocol is available, enabling the coalition to jointly create a signature for some message m. Additionally, group changes are supported. Now ABC signs message m_1 with their secret key x_{ABC} using the signing protocol, resulting in the signatures $Sig(ABC, m_1)$. This situation is depicted in figure 2.2.

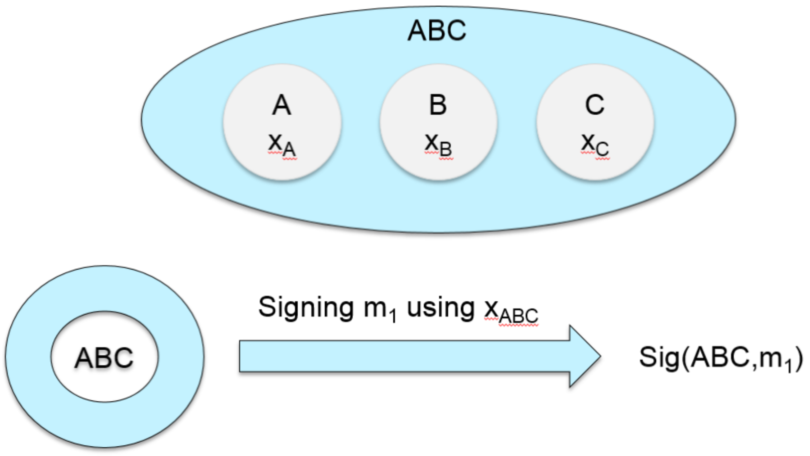


Figure 2.2: Coalition ABC Signing message m_1

Now a group change occurs, where member D joins the coalition. This results in a new coalition composition $ABCD$. The new member D has its own secret keyshare and the new secret key x_{ABCD} is determined. Now $ABCD$ signs message m_2 with their secret key x_{ABCD} using the signing protocol, resulting in the signatures $Sig(ABCD, m_2)$. This situation is depicted in figure 2.3.

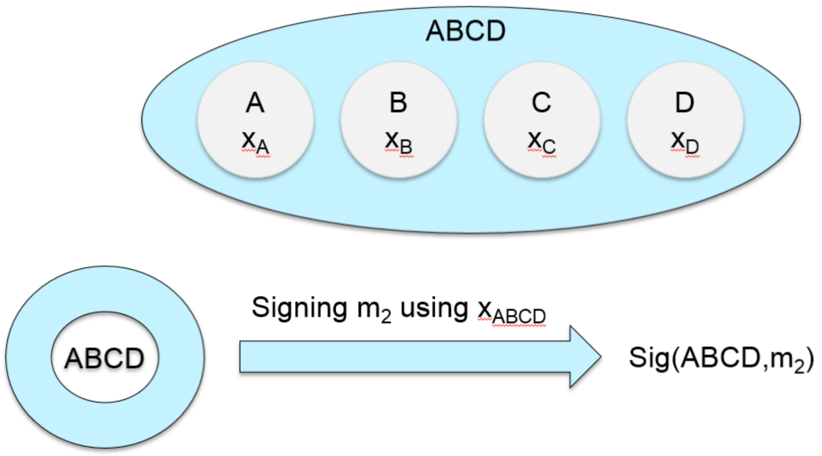


Figure 2.3: Coalition ABCD Signing message m_2

Now the Backlogging problem can be defined. Take the situation that is depicted in 2.4. Even though $ABCD$ is the current composition of the coalition, ABC signs message m_3 using their old secret key x_{ABC}. This results in the signature $Sig(ABC, m_3)$. The problem is that member D is not able to tell if the signature $Sig(ABC, m_3)$ was created before or after the group change. If it was created before D joined the coalition, it was created during the time period ABC was active, so the signature $Sig(ABC, m_3)$ is valid. If it was created after D joined the coalition, it was created during the time period $ABCD$ was active, hence the signature $Sig(ABC, m_3)$ is an instance of backlogging and is not valid.

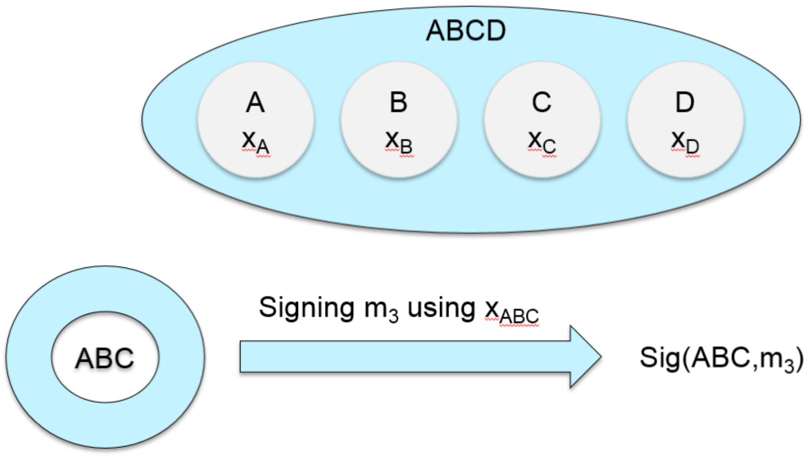


Figure 2.4: Coalition ABC Signing message m_3

Let's apply the problem scenario, provided in section 1.2, to this example. The main goal of the coalition is to be able to agree on missions and its details. Let's take the scenario where A, B and C agree on some mission and its details, but D disagrees on this. This means the mission should not be executed, since not all members of the coalition agree on it. However, taking the situation provided in figure 2.4, A, B and C can use their old secret key to sign the mission and its details and claim that it was signed during the time period ABC was the active composition of the coalition. Since D is not able to prove the resulting signature is an instance of backlogging, the signature is deemed valid, hence the mission and its details is agreed on. This example shows that this is an important problem that needs to be tackled.

3 Prior Art

Developing signature schemes that are able to perform in a dynamic coalition environment has been researched throughout the years. In this Chapter we will discuss prior art regarding the topic of this thesis. Section 3.1 discusses a signature scheme proposed by *Van der Lubbe et al.*[25]. This paper is one of the contributions of the Master Thesis conducted by *Merel J. de Boer* at the *Delft University of Technology*. The proposed signature scheme can be expanded and tackles backlogging by using two *Oblivious Third Parties*. *Van Elsas et al.*[26] then improved this signature scheme by enabling the signature scheme to be reduced and by replacing the Oblivious Third Parties by *One-Way Accumulators*, dismissing the need for Third Parties in general. Our signature scheme is strongly based on this signature scheme. Section 3.2 discusses the signature scheme introduced by *Van Elsas et al.*

3.1. A Signature Scheme for a Dynamic Coalition Defence Environment without Trusted Third Parties

In 2014 *J.C.A. van der Lubbe, M. de Boer* and *Z. Erkin* introduced a *Signature Scheme for a Dynamic Coalition Defence Environment without Trusted Third Parties*[25]. This paper is related to the research conducted by *Merel J. de Boer* for her Master Thesis. The aim of their paper is to introduce a distributed (n, n)-signature scheme, that can be expanded to a distributed $(n+1, n+1)$-signature scheme, without the use of Trusted Third Parties. The solution presented in this paper is a modification of *Park and Kurosawa's* signature scheme[21], that is explained in section 2.1.1. In this section we will discuss the contents of this paper.

Problem Scenario

The problem scenario in this paper is similar to the problem scenario provided in section 1.2. However, where in our problem scenario the focus is set on *mission agreement*, their focus is set on creating a proper *Public Key Infrastructure (PKI)*. The concept of a PKI is explained in Section 2.1.2. Their scenario considers a combined joint network. To implement a PKI for for this network, the dynamic coalition functions as a distributed *Certificate Authority (CA)*. This distributed CA is then able to provide verifiable credentials to the users of this network, ensuring the users have proper authorisation and access rights in the network. Because the network is placed in a dynamic coalition environment, the expansion of the distributed CA is considered. However, due to the scope of the paper, the authors do not consider reduction.

Preliminaries

The proposed signature scheme uses a couple of building blocks to ensure a secure expansion of the signature scheme. These building blocks will be briefly explained.

Oblivious Third Parties (OTP) An OTP is a compromise between a complete independent calculation and the use of a TTP. It is used to perform required calculations on input and it is assumed that the output is always correct. OTPs are entities outside the coalition, meaning they are not influenced by any of the coalition members. Because the OTPs are not trusted, information send to them will be encrypted.

Homomorphic Encryption A homomorphic cryptosystem has the property $E(x) \oplus E(y) = E(x \oplus y)$ for some operation $\oplus$. This is used to send an OTP information without revealing the information to the OTP, but ensure that the OTP is able to perform operations on it.

EED-model The EED-model is a sequence of encryption and decryption. Assume the existence of OTP$_A$ and OTP$_B$. A secret share is first encrypted with the public key of OTP$_B$. This encrypted share is then encrypted again with the coalition's homomorphic public key, resulting in a double encrypted secret share. This secret share will then be sent to OTP$_B$. However to enhance the EED-model and keep OTP$_B$ truly ignorant, the double encrypted secret share is first send to OTP$_A$, which encrypts this with its own public key before sending it to OTP$_B$.

Distributed El Gamal (n,n)-signature scheme
The distributed (n,n)-signature scheme used in the paper is defined as follows.

Let p and q be large primes such that q divides $p-1$ and generator g generates the subgroup G_q of $\mathcal{Z}_p$ of order q.

Protocol to Generate Random Numbers (PGRN)

1. Each member P_i chooses $x_i \in \mathcal{Z}_q$, at random and broadcasts $y_i = g^{x_i} \bmod p$
2. Every P_i computes $y \triangleq \prod_{i=1}^{n} y_i$. Note that $x \triangleq \sum_{i=1}^{n} x_i$ and thus $y = g^x$.

Key Generation Protocol

1. $P_1, ..., P_n$ execute the PGRN protocol. Let the public output be y and the secret key of P_i be x_i.
2. The public verification key of the group is (p,q,g,y).

Signature Issuing Protocol Let m be a message and h be a one way hash function

1. $P_1, ..., P_n$ execute the PGRN protocol. Let the public output be $v(= g^{\beta} \bmod p)$.
2. Each P_i reveals $\gamma_i \triangleq wx_i + h(m)\beta_i \bmod q$.
3. Each P_i verifies that for $\forall l, g^{\gamma_i} = (y_l)^w (v_l)^{h(m)}$. The validity of the signature (t,w) is verified by $w \equiv (g^{t/h(m)} y^{-w/h(m)} \bmod p) \bmod q$. Note that $w = (g^{\beta} \bmod p) \bmod q$.

Expanding the Signature Scheme
For expansion, the public keys (p,q,g,y) remain valid as well as the secret key $x = \sum_{i=1}^{n} x_i$. The authors of the paper explain that when the signature scheme is expanded to include a new member, the new member does not get a share of the secret key. Instead all members generate a random number and register it using the OTPs to make it verifiable that all members joined the signing process.

The expansion consists of two protocols: The *Initialisation Protocol*, which is used to generate and register the random numbers, and the *Modified Signature Issuing Protocol*, which needs not only the secret key, but also the random numbers to generate a valid signature.

The Initialisation protocol uses two OTPs, OTP$_A$ and OTP$_B$, the EED-model and homomorphic encryption to generate the random numbers in an appropriate way. At the end of the protocol, OTP$_A$ keeps the secret r and calculates $R = h(g^{-r})$, which is then published. This value is then also used when verifying the validity of a signature. The Modified Signature Issuing protocol ensures that a valid signature is only created if all random secrets are used, hence participation of the new members is required.

It is important to understand that this solution does not completely tackle the Backlogging problem, since it is still possible for earlier compositions to either use old random numbers or just the secret key to create signatures and backdate them. The authors explain that a *time-stamping mechanism* is necessary to prevent backlogging completely. However this has not been implemented in the proposed solution.

Limitations
OTP_A has knowledge of r. This introduced a big security risk. If one of the old members obtains knowledge of this value, the old members can create valid signature without the participation of the new members.

Additionally, it is an important requirement that the OTPs do not work together. At a certain point in the Initialisation protocol, the OTP has encrypted knowledge of the random secrets. Because the EED-model is used, the OTP can not decrypt this information, however the members of the distributed CA can. If the original members of the coalition can obtain knowledge about the random values of the new members, they can forge signatures, which should be avoided.

As earlier stated, this solution does not fully prevent backlogging. The authors explain the need of a *time-stamping mechanism*, however this has not been implemented.

Lastly, even though the signature scheme is designed for a dynamic coalition environment, it is not possible to reduce the signature scheme, to exclude a member.

3.2. A Dynamic Digital Signature Scheme Without Third Parties

As a respond to the previously discussed paper[25], *M. van Elsas, J.C.A. van der Lubbe* and *J.H. Weber* introduced in 2015 a *A Dynamic Digital Signature Scheme Without Third Parties*[26]. This solution improved the previous signature scheme, by enabling reduction of the signature scheme and removing the need of Third Parties in general. The signature scheme proposed in this paper forms the basis of our Blockchain-based signature scheme.

The Dynamic Signature Scheme The authors first introduce the signature scheme with a static group. This is similar to the signature scheme explained in section 2.1.1, but uses keyshares to enable the participation of multiple users. Then the protocols are extended to facilitate a dynamic group. Two protocols are added to the signature scheme enabling expansion and reduction.

The ability to perform as a dynamic signature scheme is reflected in the keypair and keyshares used. The secret key used is based on the secret keyshares of the active members. The secret key is equal to the product of these secret keyshares. For expanding the signature scheme, the *Joining Protocol* is used. This protocol lets the new member define its secret keyshare and lets the new keypair be defined to include this keyshare. Hence a new keypair is determined and the new composition of the coalition can be defined. In a similar way, reduction of the signature scheme is defined. For reducing the signature scheme, the *Leaving protocol* is used. Because the members of the new composition already have the keyshares available, its only necessary to define the new keypair to exlude the member that wishes to leave. The new keypair is determined and the new composition of the coalition is defined.

Tackling the Backlogging Problem Due to the way the dynamism of the signature scheme is implemented, backlogging is possible. Because a group change is reflected in the keypair used, an older composition of the coalition can use their old keypair to create signatures and claim the signatures were created in the time period where they were active, as is explained in section 2.3.

However the authors created a method to make an instance of backlogging detectable. They do this using a combination of a *memberlog* and a *One-Way Accumulator*[19]. The basic idea is to keep a set of all messages signed by a group composition. Whenever a signature is suspected to be an instance of backlogging, it is checked if the corresponding message is contained in the set of the messages that were signed by the corresponding group composition. If this is the case it was signed during the time period the composition was active, so it is a valid signature. If this is not the case, it was signed during a time period the composition was not active, so it is an instance of backlogging.

This has been implemented in the following way. First of all, whenever a group change occurs, so either the Joining protocol or the Leaving protocol is issued, a special signature is issued by the current group composition. The message that needs to be signed includes the identities of all members of the new group composition, the newly determined public key and the set of all signatures issued by the current group composition. This signature is then stored in a collection called the *memberlog*. Hence the memberlog contains information about all group compositions that have existed. A separate signature issuing protocol has been defined for this case called the *Group Signature Issuing Protocol*. Note that because in order to sign a message, all members need to agree on signing the message, and the aforementioned message contains information of the new coalition, it is ensured that all members of the current group composition agree on the group change. Hence the signature scheme ensures that group changes are performed in a secure way.

Additionally, as earlier stated, the messages that were signed by a group composition are stored in a set. This set is defined as a *One-Way Accumulator*[19]. A One-Way Accumulator is a one way membership function. Depending on the implementation, in order to prove that an element is part of an OWA, either only an identifier or a combination of an identifier and a witness value of the element needs to be provided to prove that the element is part of the OWA. Whenever a group change occurs or the first coalition is defined, a new

OWA is also defined. The *Regular Signature Issuing Protocol* is defined in such a way, that at the end of the protocol, the message signed is added to the currently used OWA.

Now lets refer back to the Backlogging Example provided in Section 2.3 and take the scenario depicted in figure 2.4. Because of the methods used in this signature scheme, D is able to determine if the signature $Sig(ABC, m_3)$ is an instance of backlogging. It first checks if the corresponding group composition, ABC, is a valid group composition, by checking if an entry of this group composition exists in the memberlog. If this is the case, D checks if m_3 is included in the corresponding OWA. If this is the case it was signed during the time period ABC was active, so it is a valid signature. If this is not the case, it was signed during a time period ABC was not active, so it is an instance of backlogging.

Limitations The authors explain that the OWA implementation is not the most efficient. It requires memory linear to the amount of elements. Additionally it requires a predefined size. This means that either a group composition is limited to signing a certain amount of messages or everytime this size is reached, a new OWA has to be defined. The authors chose for the latter, by defining an extra protocol that created a new OWA whenever the current OWA is full.

Additionally, even though the signature scheme is completely defined, the signature scheme is not very detailed. Some definitions can therefore be quite challenging when applying the signature scheme in the real-world. For example, the signature scheme uses identifiers to define members of a group composition. However these identifiers need to be verifiable and trusted. If the identifiers can be spoofed, the security of the signature scheme can be broken.

4 The Blockchain-based Signature Scheme

In this Chapter we propose our signature scheme, the *Blockchain-Based Signature Scheme for Dynamic Coalitions*. The Blockchain-Based Signature Scheme uses the same idea as the signature scheme introduced by *van Elsas*, which is discussed in section 3.2. However the protocols and the process has been defined to resemble the Blockchain model. This Chapter is organised as follows. First, an important decision regarding the consensus mechanism used is highlighted and explained. Then a basic expplanation of how the Blockchain-based Signature Scheme works is provided. Finally, a detailed description of our signature scheme, including the protocol descriptions, is provided. For convenience, for the remainder of this thesis report, the paper written by *van Elsas et al.*[26] will be referred to as the *original paper*.

4.1. Achieving Consensus

The original method of achieving consensus in the Bitcoin environment[16], the *Proof-of-Work* model, is not applicable in this context. The Proof-of-Work concept is very interesting and works well in the *Bitcoin* environment. This is due to the incentive the users of the Bitcoin model have to perform the work and how well the time-aspect tackles the double spending challenge in this context. However, this does not apply to a secure dynamic coalition environment.

The double-spending problem can be translated in this context to a member wanting to change the composition of an earlier coalition or adding/deleting a message signed by this coalition. This is obviously an issue that needs to be tackled. However the requirement, mentioned in Section 2.2, for this concept to work, that finding the Proof-of-Work of the next Block is immediately started when a Block is created, is not applicable in this context unless changes in the coalition happen frequently enough.

Also in the Bitcoin environment, members have an incentive to perform the Proof-of-Work, because of the *transaction fee*[16]. However in the secure dynamic coalition environment, members do not have a direct incentive. Hence a member can decide not to cooperate, with the idea that another member will complete the task.

Unless these two problems are tackled, the Proof-of-Work concept will not work in this context. But the goal of the Proof-of-Work concept is to achieve *consensus* and make a Block *immutable*, since it is explained that changing a Block after consensus is achieved is hard. In this context this can also be achieved by *signing* a Block. This is because in order to create a valid signature, participation is needed by all members. This means consensus is needed in order to create a signature. If all members of the coalition agree on the Block and its contents, the Block can be signed, ensuring the consensus of all members. Then the Block can be added to the blockchain. This changes the Proof-of-Work to a new concept, *Proof-of-Consensus*.

4.2. Basic Idea

The first thing that needs to be noted is that because the Proof-of-Work is changed to Proof-of-Consensus, when a Block is signed, it results into a signature. A *Signed Block* is defined as the combination of the signature and the Block that has been signed. Note that the Signed Block replaces the *Hashed Block* from the original Blockchain model. Similarly to the original paper, the period between changes in the group composition is called a *phase*. A distinction is made between *regular signature issues*, signing normal messages, and *Block signature issues*, signing Blocks. A secret key is denoted by x and a public key is denoted by y.

During each phase, every member creates an empty Merkle Tree. Whenever a regular signature is issued, each member adds the corresponding message to the Merkle Tree of that phase. In a similar way as the One-Way Accumulator in the original paper, this is used to prevent backlogging. Note that because each phase has a Merkle Tree and a phase is defined per group composition, each group composition has its own Merkle Tree. Due to the absence of a shared database, each member has to define its own Merkle Tree. However, if the protocols are followed correctly by every member, the Merkle Trees will be identical.

Once a change in the group composition is requested, the public key y of the new coalition is determined and the Block is defined. The Block contains the following items:

- The y value of the new coalition
- The y value of the current coalition
- The signature of the previously Signed Block
- A Merkle Root from the current Merkle Tree

Next, the Block is signed. As earlier stated, the combination of the resulting signature and the Block is called a *Signed Block*. Signing the Block ensures that all members of the current coalition approve the change, hence ensuring a new member that all current members accept its admittance or ensuring a leaving member that all current members accept its departure. It also ensures that all members have knowledge and agree on the signatures that have been issued by the current group composition, because the Block also contains the Merkle Root. Signing the Block makes the Block immutable, since for a change to the Block, approval is needed by all members. And approval by all members makes the change valid. A Block contains the signature of the previously Signed Block. This functions as a pointer to the previous Block, hence creating a chain. If an earlier Block is changed, the corresponding signature is also changed, resulting in a change in all the subsequent Blocks.

The first Block, in the initialisation phase, only contains the y value of the coalition in the making. Since there was no previous coalition, there exists no previously Signed Block or Merkle Tree. This Block is then signed with the newly formed y value, to ensure that all members agree on forming a coalition.

4.2.1. Preventing Backlogging

If a member finds a signature that is made with a y value that corresponds to a previous coalition and suspects backlogging, it only needs to find the Block that is signed using this y value and check if the message is contained in the Merkle Tree from that Block. If this is not the case, the signature is an instance of backlogging. It should be noted that every time a signature is being verified with a y value that is not equal to the y value used by the latest coalition, this check needs to be done to prevent backlogging completely.

4.3. Detailed Description

This section describes the protocols of the signature scheme. The protocols are similar to the ones described in the original paper, however some changes are applied. First an overview of the definitions used in the protocols is provided. Then the five protocols of the signature scheme are provided and a brief explanation is given to elaborate on how the protocol works. The five protocols discussed are the *Initialisation Protocol*, the *Joining Protocol*, the *Leaving Protocol*, the *Regular Signature Issue Protocol* and the *Block Signature Issue Protocol*.

4.3.1. Definitions

$\mathcal{N}$ is the current set of members of this coalition. The scheme enables members to leave and join this set. Next we have p and q which are large primes such that q divides $p-1$. g generates the group G_q which is a subgroup of $\mathcal{Z}_p$, of order q. We assume that p, q and g are publicly known. For the signature issue protocols, m is always the message that is going to be signed. The hash value of m is denoted by $h(m)$, where h is a publicly known hash function with a range from 1 to $q-1$. Lastly, the period between changes in the group composition is called a phase and is denoted by o.

A *Block* is denoted by $B_o(y_{o+1}, y_o, sig_{o-1}, Troot_o)$, where:

- o = the phase the Block was created

- y_{o+1} = the y value of the new group composition that is going to be used in the $o+1th$-phase
- y_o = the y value of the current group composition that is used in the oth-phase
- sig_{o-1} = the signature $sig_{o-1}(t, w, y)$ contained in the Signed Block SB_{o-1}, signed in the $o-1th$-phase
- $Troot_o$ = the Merkle Root corresponding to the current Merkle Tree T_o that is used in the oth-phase.

Every phase o, each member defines a Merkle Tree T_o. As stated in section 4.2, due to the absence of a shared database, it is necessary that each member defines its own Merkle Tree, but consistency between these Merkle Trees should be present if the protocols are followed correctly by all members. When a reference is made to a Merkle Tree T_o, it is of no importance which of the T_o Merkle Trees is taken, since it can be assumed that the Merkle Trees for all members are identical. All messages of the regular signature issues in phase o are stored in Merkle Tree T_o. When a group change occurs in phase o, Block B_o is defined. First the new group composition agrees on a new y value, that is going to be used in phase $o+1$, hence has the label y_{o+1}. Then the Block is defined, containing: the y value of the new group composition, the y value of the current group composition, the signature contained in the previously Signed Block and the Merkle Root corresponding to the Merkle Tree of the current phase. The Block has two functionalities. First of all, the Block contains the y value of the new group composition, so when the Block is signed, all participating members are ensured that all members of the current group composition agreed on that value and therefore agreed on the group change. Secondly, the Block contains the Merkle Root of the Merkle Tree of the current phase. When the Block is signed, all current coalition members are also ensured that the Merkle Tree is accepted, thus meaning all current coalition members agree that the messages stored in the Merkle Tree are valid.

A *Signed Block* is denoted by $SB_o(B_o, sig_o(t, w, y))$, where:

1. o = the phase the Block was signed in
2. B_o = the Block that was defined in the oth-phase
3. $sig_o(t, w, y)$ = the signature in the oth-phase corresponding to Block B_o with signature values t, w and y

When a group change occurs in phase o, Block B_o is defined. In order to achieve consensus and make the Block immutable, a Block signature is issued with Block B_o as message. The resulting signature $sig_o(t, w, y)$ is prove that the Block has been signed. Hence the combination of $(B_o, sig_o(t, w, y))$ is the *Signed Block*. To verify the validity of Block B_o, one only needs to verify the validity of the corresponding signature $sig_o(t, w, y)$. Once the Signed Block SB_o is created, phase $o+1$ starts.

The first block in the chain is a special block. This is because there exist no previous phase, hence there are no previous blocks or values. This Block is denoted by $B_0(y_1)$. It only contains the y value the first composition of the coalition has agreed upon. Once this Block is signed, the first phase, phase 1, starts.

4.3.2. Initialisation Protocol

For initialisation, that happens in phase 0, the following protocol will be applied:

1. Each member $i \in \mathcal{N}$ chooses a random secret x_i from $\mathcal{Z}_q$.
2. Each member $i \in \mathcal{N}$ broadcasts $z_i = g^{x_i} \bmod p$ to all other members.
3. Each member in $\mathcal{N}$ computes $y_1 = \prod_{i \in \mathcal{N}} z_i = g^x \bmod p$.
4. The Block $B_0 := B_0(y_1)$ is defined and a Block signature is issued by the members in $\mathcal{N}$ with $m := (B_0)$, resulting in the Signed Block $SB_0 := SB_0(B_0, sig_0(t, w, y_1))$.
5. Each member in $\mathcal{N}$ defines a new empty Merkle Tree T_1.
6. The first phase starts with Merkle Tree T_1 and Signed Block SB_0, containing the approved y_1 value.

The shared secret is defined as follows:

$$x \triangleq \sum_{i \in \mathcal{N}} x_i$$

The shared secret is not known by any member. If a member joins or leaves the coalition, the shared secret changes according to the individual shares of the secret key of the new composition of the coalition.

The *Initialisation Protocol* is used to form the coalition. This happens in phase 0 and is done by creating and signing the first Block B_0. First the members in $\mathcal{N}$ create their secret keyshares and decide on a y value. This is stored in the first Block B_0. Then the Block is signed using the Block Signature Issue Protocol resulting in the Signed Block SB_0. Lastly, the first Merkle Tree T_1 is defined by each member. The first phase then starts with Merkle Tree T_1 and the Signed Block SB_0, containing the y_1 value. Notice that usually the y value in the signature of a Signed Block is the y value of the current phase, but in this special case, where there is no y value yet, the Block is signed with the y_1 value, that is used in the next phase.

4.3.3. Joining Protocol

When a new member k wants to join the coalition in the current phase o, the following protocol will be applied. The extended group $\mathcal{N} \cup \{k\}$ is denoted by $\mathcal{N}'$.

1. k requests the Signed Blocks $SB_0, ..., SB_{o-1}$ from a member $i \in \mathcal{N}$.
2. The member i sends the Signed Blocks $SB_0, ..., SB_{o-1}$.
3. k verifies the validity of the chain.
4. Each member $i \in \mathcal{N}$ sends $z_i = g^{x_i} \bmod p$ to k.
5. k chooses a random x_k from $\mathcal{Z}_q$.
6. k broadcasts $z_k = g^{x_k} \bmod p$ to each member $i \in \mathcal{N}$.
7. Each member in $\mathcal{N}'$ computes $y_{o+1} = \prod_{i \in \mathcal{N}'} z_i$.
8. The Block $B_o := B_o(y_{o+1}, y_o, sig_{o-1}, Troot_o)$ is defined and a Block signature is issued by the members in $\mathcal{N}$ with $m := (B_o)$ resulting in a new Signed Block $SB_o := SB_o(B_o, sig_o(t, w, y_o))$.
9. A member $i \in \mathcal{N}$ sends SB_o to k.
10. k checks if SB_o is valid by verifying the validity of the signature of SB_o and if sig_{o-1} contained in Block B_o equals the signature of Signed Block SB_{o-1}.
11. Each member in $\mathcal{N}'$ defines a new empty Merkle Tree T_{o+1}.
12. The new phase $o+1$ starts with the group $\mathcal{N}' : \mathcal{N} := \mathcal{N}'$, Merkle Tree T_{o+1} and Signed Block SB_o, containing the approved y_{o+1} value.

When a new member k requests an admittance to the coalition in phase o, the *Joining Protocol* is issued. The member k requests the Signed Blocks $SB_0, ..., SB_{o-1}$. Upon retrieval, k verifies the validity of the chain. How this is done is explained in the next paragraph. Member k then creates its secret keyshare and the new y_{o+1} value is determined by the members of the new coalition, including member k. The Block B_o is then defined, containing the new y_{o+1} value, the currently used y_o value, the signature sig_{o-1} of the previously Signed Block SB_{o-1} and the Merkle Root $Troot_o$ of the current Merkle Tree T_o. This Block is signed using the Block Signature Issue Protocol, resulting in the Signed Block $SB_o(B_o, sig_o(t, w, y_o))$. The newly Signed Block is send to k, which then verifies the validity of its signature and checks if the signature of the previously Signed Block in Block B_o is correct. Lastly, every member of the new group composition $\mathcal{N}'$ defines a new empty Merkle Tree T_{o+1}. Now the new phase $o+1$ starts with Merkle Tree T_{o+1} and the Signed Block SB_o, containing the y_{o+1} value.

Verifying the Validity of the Chain As earlier stated, upon retrieval, member k verifies the validity of the signatures of the Signed Blocks and that the chain is build in a correct manner. This is done to ensure member k that the coalition has been performing the group change protocols correctly and no malicious attempt has been made to change the blocks in an incorrect manner.

This is done as follows. Starting with Signed Block SB_0, it is checked if SB_0 contains a valid signature. For all subsequent Signed Blocks SB_i, it is first checked if the signature of the previously Signed Block contained in Block B_i, sig_{i-1}, actually matches the signature of Signed Block SB_{i-1}. This ensures that the chain is build correctly. Then it is checked if the Signed Block SB_i contains a valid signature.
The validity of the signature $sig_o(t, w, y)$ is verified by

$$w \equiv (g^{t/h(m)} y^{-w/h(m)} \bmod p) \bmod q$$

4.3.4. Leaving Protocol

When a member j wants to leave the coalition in the current phase o, the following protocol will be applied. The reduced group $\mathcal{N} \setminus \{j\}$ is denoted by $\mathcal{N}'$.

1. Each member in $\mathcal{N}'$ computes $y_{o+1} = \prod_{i \in \mathcal{N}'} z_i$.

2. The Block $B_o := B_o(y_{o+1}, y_o, sig_{o-1}, Troot_o)$ is defined and a Block signature is issued by the members in $\mathcal{N}$ with $m := (B_o)$ resulting in a new Signed Block $SB_o := SB_o(B_o, sig_o(t, w, y_o))$.

3. Each member in $\mathcal{N}'$ defines a new empty Merkle Tree T_{o+1}.

4. The new phase $o+1$ starts with the group $\mathcal{N}' : \mathcal{N} := \mathcal{N}'$, Merkle Tree T_{o+1} and Signed Block SB_o, containing the approved y_{o+1} value.

When a member j wishes to leave the coalition in phase o, the *Leaving Protocol* is issued. First the new y_{o+1} value is determined by the members of the new coalition, excluding member j. Since every member already has the individual z_i values of the members, this is a trivial task and no communication is needed. Then the Block B_o is defined, containing the new y_{o+1} value, the currently used y_o value, the signature sig_{o-1} of the previously Signed Block SB_{o-1} and the Merkle Root $Troot_o$ of the current Merkle Tree T_o. This Block is signed using the Block Signature Issue Protocol, resulting in the Signed Block $SB_o(B_o, sig_o(t, w, y_o))$. The Block is signed with the members of $\mathcal{N}$, ensuring that the member j also agrees on the departure. Lastly, every member of the new group composition $\mathcal{N}'$ defines a new empty Merkle Tree T_{o+1}. The new phase $o+1$ starts with Merkle Tree T_{o+1} and the Signed Block SB_o, containing the y_{o+1} value.

4.3.5. Regular Signature Issue Protocol

When a regular signature is issued in the current phase o, the following protocol is applied.

1. Each $i \in \mathcal{N}$ chooses a random β_i from $\mathcal{Z}_q$.

$$\beta \triangleq \sum_{i \in \mathcal{N}} \beta_i$$

   Here β is the shared random secret, not known by any member.

2. Each $i \in \mathcal{N}$ broadcasts $c_i = g^{\beta_i} \bmod p$ to all other members.

3. Each $i \in \mathcal{N}$ reveals $a_i = g^{\gamma_i}$ where $\gamma_i \triangleq wx_i + h(m)\beta_i \bmod q$. Here w is equal to $v \bmod q$ with $v = \prod_{i \in \mathcal{N}} c_i = g^{\beta} \bmod p$.

4. Each member in $\mathcal{N}$ verifies that $\forall j, a_j = (z_j)^w (c_j)^{h(m)}$.

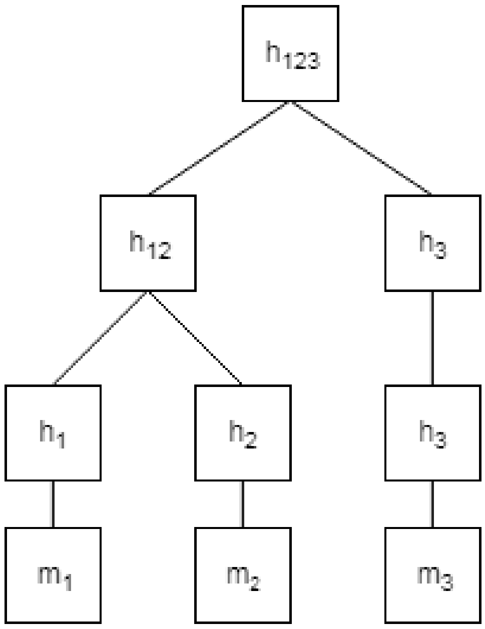
5. Each member in $\mathcal{N}$ computes $a = \prod_{i \in \mathcal{N}} a_i = \prod_{i \in \mathcal{N}} g^{\gamma_i} = g^{\sum_{i \in \mathcal{N}} \gamma_i} = g^t$ where $t = wx + h(m)\beta \bmod q$.

6. Each member in $\mathcal{N}$ adds m to its Merkle Tree T_o.

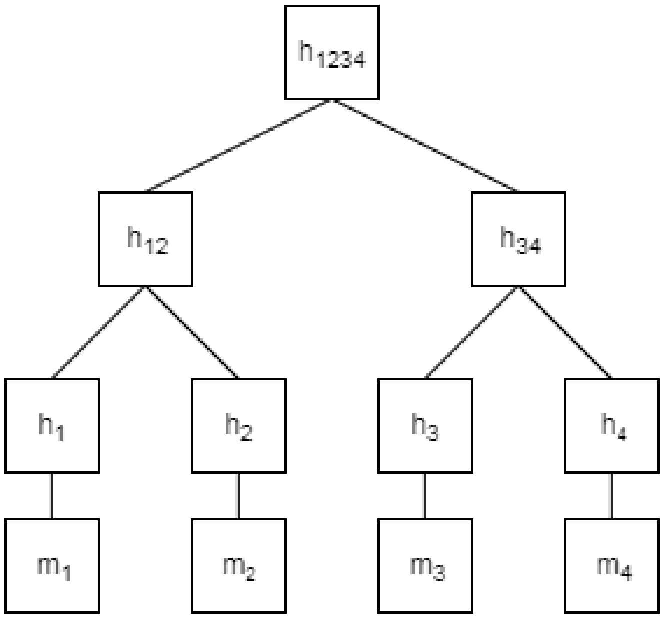
The validity of the signature $sig_o(t, w, y)$ is verified by

$$w \equiv (g^{t/h(m)} y^{-w/h(m)} \bmod p) \bmod q$$

When the coalition wishes to sign a message m with a regular signature in phase o, the *Regular Signature Issue Protocol* is issued. A shared random secret is created and accordingly the signature values can be created. If a valid signature for the message m is created, the message is added by each member of the coalition to its Merkle Tree T_o. It is possible to verify the validity of the signature using the provided formula.



(a) A Merkle Tree with 3 messages



(b) A Merkle Tree with 4 messages

Adding messages to a Merkle Tree Take the scenario where a regular signature is issued in phase o and the kth-message m_k is added to the Merkle Tree T_o. In a Merkle Tree, all leaves are hashed messages $h(m_i)$ where $i \in 1,..,k-1$. Each parent is then defined as follows:

$$parent = hash(left\text{-}child || right\text{-}child)$$

This can be illustrated as follows. Let's take $k = 3$ and define $h_1 = h(m_1)$, $h_2 = h(m_2)$ and $h_3 = h(m_3)$. Now h_1 and h_2 are paired and hashed, resulting into $h_{12} = h(h_1||h_2)$. Because there is no pair for h_3, h_3 goes up in the tree. Now h_3 can be paired with h_{12}, resulting into $h_{123} = h(h_{12}||h_3)$. In this tree h_{123} is the *Merkle Root*. This can be seen in figure 4.1a.

Take the scenario where the message m_4 has to be added to the Merkle Tree. Again we define $h_4 = h(m_4)$. Similarly as the previous situation, h_1 and h_2 are paired and hashed, resulting into $h_{12} = h(h_1||h_2)$. But now h_3 has a pair, since h_4 is introduced. This results into $h_{34} = h(h_3||h_4)$. Then h_{12} can be paired with h_{34}, resulting into $h_{1234} = h(h_{12}||h_{34})$, which is the new Merkle Root. This can be seen in figure 4.1b.

For clarity, in the figures the conversion from m_i to $h_i = h(m_i)$ is also given.

4.3.6. Block Signature Issue Protocol

When a Block signature is issued with message $m := (B_o)$ in the current phase o, the following protocol is applied.

1. Each $i \in \mathcal{N}$ chooses a random β_i from $\mathcal{Z}_q$.

$$\beta \triangleq \sum_{i \in \mathcal{N}} \beta_i$$

   Here β is the shared random secret, not known by any member.

2. Each $i \in \mathcal{N}$ broadcasts $c_i = g^{\beta_i} \bmod p$ to all other members.
3. Each $i \in \mathcal{N}$ reveals $a_i = g^{\gamma_i}$ where $\gamma_i \triangleq wx_i + h(m)\beta_i \bmod q$. Here w is equal to $v \bmod q$ with $v = \prod_{i \in \mathcal{N}} c_i = g^{\beta} \bmod p$.
4. Each member in $\mathcal{N}$ verifies that $\forall j, a_j = (z_j)^w (c_j)^{h(m)}$.
5. Each member in $\mathcal{N}$ computes $a = \prod_{i \in \mathcal{N}} a_i = \prod_{i \in \mathcal{N}} g^{\gamma_i} = g^{\sum_{i \in \mathcal{N}} \gamma_i} = g^t$ where $t = wx + h(m)\beta \bmod q$.
6. Each member in $\mathcal{N}$ defines the signature $sig_o(t, w, y)$ for message m.

7. Each member in $\mathcal{N}$ defines the Signed Block $SB_o(m, sig_o(t, w, y))$ with the message $m := (B_o)$.

The validity of the signature $sig_o(t, w, y)$ is verified by

$$w \equiv (g^{t/h(m)} y^{-w/h(m)} \bmod p) \bmod q$$

This protocol is very similar to the regular signature protocol. When a group change occurs in phase o, either the *Joining Protocol* or the *Leaving Orotocol* is issued. These protocols include signing a Block B_o using the *Block Signature Issue Protocol*. This protocol is then issued with message $m := (B_o)$. A shared random secret is created and accordingly the signature values can be created. Now each member of the current group composition defines the signature $sig_o(t, w, y)$ that corresponds to the Block B_o and the Signed Block can be defined. Each member of the current group composition defines the new Signed Block $SB_o(m, sig_o(t, w, y))$ with $m := (B_o)$.

5 Analysis of Implementation Aspects

The second contribution of this thesis is an analysis of the aspects regarding implementing the *Blockchain-based Signature Scheme*. Our signature scheme uses several concepts that are open for interpretation or assumed to be implemented. These concepts could introduce problems during implementation of our signature scheme. Hence we will discuss these concepts and analyse them according to the given context. This chapter is constructed as follows. First, in Section 5.1, an analysis will be provided regarding *Unique Verifiable Identifiers*. In Section 5.2, potential implementations of *Publicly Known Variables* is provided. Next, in Section 5.3, the use of *Broadcasting* is discussed. In Section 5.4, an analysis of the use and requirements regarding *Hash Functions* is discussed. And lastly, in Section 5.5, several solutions for achieving *Data Consistency* is provided.

5.1. Unique Verifiable Identifiers

For the signature scheme in the paper written by *van Elsas et al.*[26], unique identifiers are defined for the members of the coalition. These are then used to define the coalition. But these identifiers have another important role, which is not stated in the paper, but also concerns our signature scheme. The identifiers are also very important when information is being shared by the members. For example, when the *Regular Signature Issue Protocol* is used, provided in Section 4.3.5, each member $i \in \mathcal{N}$ creates its part of the signature, a_i. In step 4 of the protocol, it is verified that for each member, a_i has been created correctly. This requires a linkage between the a_i a member provides and its public keyshare z_i every member of the coalition has stored. Having an identifier these values are being linked to, enables the linkage between these two values.

Using these identifiers in the communication network that is used by the coalition, members of the coalition are able to know the source of the messages that are being send in this network. But in order to improve security, these identifiers should be verifiable. It should not be possible to perform actions with a unique identifier other than your own. Hence it is required to couple these identifiers using a *Public-Key Infrastructure (PKI)*. The concept of a PKI is explained in Section 2.1.2. A member can then prove that it actually owns the unique identifier. Because secure communication is very important, it is necessary that the signature scheme is able to use a PKI.

It should be noted that in order for this scheme to be implemented in a secure way, such identifiers have to be in place, hence a PKI has to be in place. If this is not the case, users of the communication network, that is used to provide information to members of the coalition, can send information using an identifier of someone else and compromise the system. However implementing a PKI using the coalition is quite a difficult task. This is briefly discussed in section 8.2 as Future Work.

5.2. Publicly known variables

Our signature scheme makes use of several publicly known variables. These variables include the large primes p and q, the generator g and the hash function h. Each member in $\mathcal{N}$ should know or have access to these variables. Two potential solutions on enabling this are provided and analysed.

Shared/Public Database A database could be created that stores the public information. When needed, the database can be issued to obtain public information. It should be noted that this database should be public, since when a potential member wishes to join, it should have access to the public variables before it has been added to the coalition to be able to create its secret share. Keep in mind that a member has been officially added to the coalition if the Block containing the new values has been signed. However creating and managing such a database can be quite challenging, since the coalition has to do this in a distributed manner. Due to the context, it is not possible for one member to manage the database. However the concept of *Distributed Database Systems* has been researched thoroughly, where also security and privacy has been analysed [20][13]. But this often requires some sort of *Identity and Access Management*, which can be quite challenging due to the given context.

Reaching Agreement at Initialisation Another option is to make sure that all members during initialisation of the coalition agree upon the public variables and store these individually. Then when a potential member wishes to join the coalition, instead of only requesting the Signed Blocks from a current member of the coalition, it also requests the public variables in the Joining Protocol.

Reaching agreement in distributed systems is a big research field. *ByzCoin* uses *Scalable Collective Signing*, which enables a leader to request the validation and signing of statements by a decentralised group of witnesses [15]. This does require a *Public-Key Infrastructure* to be in place. But the biggest issue is that this method requires a leader to be chosen, which can be challenging due to the given context.

However, a simpler method using broadcasts and responses could be used. For example, when the public variables have to be decided on, a member proposes a set of public variables and broadcasts these to the other members. When a member receives a proposal and agrees with the variables, the member broadcasts a response, including these variables, to the proposal. When a member receives multiple proposals, because multiple members send different proposals simultaneously, a predefined choosing method is used to choose a proposal to respond to. When each member has received a response to the same proposal by all other members, it can safely be assumed that the variables are agreed upon and can then be stored and used.

5.3. Broadcasting

It is sometimes the case that a member wishes to broadcast information to the other members during a protocol. For example, during the Joining Protocol, provided in Section 4.3.3, the new member broadcasts its share of the new public key to the other members of the coalition, so everyone can calculate the new public key. It is understandable that broadcasting has to be done in a proper way.

It is obvious that a communication network has to be in place. Not only for broadcasting but also for the other required communication between members, such as requesting and providing the Signed Blocks. We assume this is in place. However for the characteristics of this communication network regarding broadcasting, requirements can be set.

It is important that it is ensured that broadcast messages are received by all members of the coalition. If, due to some communication error, a member of the coalition does not receive all parts of the public key, the coalition will not be able to create valid signatures. However this is easily achievable by implementing *'acknowledgements'*. Whenever a broadcast message is received, an acknowledgement is send back to the sender of the broadcast message by the receiver. Once the sender of the broadcast message has received an acknowledgement message from every member, it can safely assume that its broadcast message has been received by all members. If the sender of the broadcast message has not received an acknowledgement from all members within a reasonable time-span, it can either re-broadcast the message or re-send the message to the members that did not send an acknowledgement back. The latter is only possible if each member has an unique identifier, which is known by each member. The concept of identifiers has been discussed in section 3.4.1. It should be noted that the *order* of the broadcast messages is of no importance. In other words, *atomicity* of the broadcast messages is not required. This is because when broadcasting is used, e.g. constructing the public key or sharing the random secret when signing a message, it is only important that every member has all parts of the shared variable.

In order to implement a *secure* broadcasting system, it should be able for receivers of broadcast messages to verify that the message actually originates from the claimed source. This requires that *Unique Verifiable Identifiers*, discussed in Section 5.1, are in place. Having such identifiers in place, a source authentication protocol, such as the *TESLA Protocol* proposed by *Perrig et al.*, can be used to enable receivers of broadcast data to verify that the received data actually originates from the claimed source[22]. This also requires that a *Public-Key Infrastructure* is in place.

5.4. Hash Function

It is often the case that when hash functions are used in a system, the assumption is made that the hash function is an ideal hash function, also called a *random oracle*[4]. This implies that for any new query, the answer is uniformly distributed in the output set and is independent of the previous query/answer pairs. However, the currently available hash functions cannot in fact be random oracles but rather computationally indistinguishable from one. *Brickell et al.* also states that "if a signature scheme based on the Discrete Logarithm Problem can be broken by an existential forgery using adaptively chosen-message attack, then either the discrete logarithm problem can be solved, or some hash function can be distinguished from an ideal hash function, or multi-collisions can be found". Taking that statement into account, it is understandable that the hash functions used are important regarding the security of a signature scheme. Hence when implementing a signature scheme, it is important that proper hash functions have been set.

A proper hash function should meet several requirements and properties. The most important requirement is that a hash function should be *Collision Resistant*[1]. This means that it should be computationally infeasible to find two distinct input strings that result into the same hash value. In other words, a proper hash function should be computationally indistinguishable from an ideal hash function. This requirement is highly dependent on the range of a hash function and the distribution of the output of the hash function in the output set. A hash function takes an input of arbitrary length, which means that there is no limit to the amount of different inputs possible. Assuming the distribution of the output of the hash function in the output set is somewhat uniform, this means the greater the range of the hash function, the smaller the chance is that a collision will occur. The range of the hash function should be taken such that a collision can occur with negligible probability.

There are two different occurrences of hash functions throughout the proposed signature scheme. The first occurrence is in the signing protocols, where either a regular message or a block is hashed, depending on which signature protocol is issued. The second occurrence is in a Merkle Tree, where each element in the Merkle Tree is hashed.

In Section 4.3.1, it is mentioned that the hash function used has a range from 1 to $q-1$. This is due to the restrictions the chosen group G_q and the field $\mathcal{Z}_p$ introduce for the calculations regarding hashed values that are done in the signature protocols. This means that the security regarding the hash function is highly dependent on the value of q. However, it is also stated in the Section 4.3.1 that q is a large prime. The availability of very large primes should ensure that it is possible to take a value for q that is large enough so that the chance of collision regarding the hash function h is negligible.

To summarise, in order to minimise the risk regarding the security of the hash function, a proper hash function standard should be used, such that the output of the hash function is close to uniform distributed in the output set. Several hash function standards are available, such as the *Secure Hash Algorithm 3*[2]. Also the value of q should be taken such that the range of the hash function is large enough to ensure that the chance of collision is negligible. However it should also be considered that it is possible that a hash function standard does not support all ranges. Therefore when deciding on the value of q, the possible ranges the hash function standard supports should also be taken into account. Or the other way around, where a hash function standard is chosen according to the value of q that is being used.

5.5. Data Consistency

In this scheme it is important that there exist data consistency between the members of the coalition. Consistency regarding public variables has already been discussed in Section 5.2. However, some data are created and updated individually, but should be consistent with similar data kept by the other members. This type of data will be referred as *Replicated Data*. The most important instances of replicated data are the Blocks and the Merkle Trees inside the Blocks. Each member creates and updates its own Block and corresponding Merkle Tree during a phase. If the protocols are followed correctly by all members, the individual Blocks and Merkle Trees should be identical. However, if there does exist inconsistency in the individually kept Blocks, due to some kind of failure, the coalition will not be able to create a valid Signed Block. In order for the *Block Signature Issue Protocol*, provided in Section 4.3.6, to successfully provide a Signed Block, the message, the Block in this case, has to be the same for all participating members. Hence, it is important that data consistency should be ensured or data inconsistency should be handled correctly.

Two solutions to this problem will be provided and analysed. For these solutions, it is assumed that a *Unique Verifiable Identifiers*, which are discussed in Section 5.1, are used in the communication network. It is also assumed that data inconsistency is always detectable. This is assumed, because data inconsistency is only an issue if the actual replicated data is used. In this scheme, it currently the case that whenever replicated data is used, some data linked to the replicated data is also shared between members. It is then possible to detect data inconsistency based on the shared data. For example, if there exists data inconsistency in the individual Blocks of the members during a phase, this will be detectable during the *Block Signature Issue Protocol*. In step 3 of the protocol, each member defines a_i accordingly with $i \in \mathcal{N}$. If for some member j of the coalition, its Block is inconsistent compared to the Blocks of the other members, this a_j will be incorrect. This will be detectable by all other members during step 4 of the protocol, where it is verified that all a_i are created correctly with $i \in \mathcal{N}$. Hence inconsistency in the replicated data, the Block, can be found using the shared data, a_i with $i \in \mathcal{N}$. The two solutions will now be discussed.

Defining a Leader A solution to this problem is to first decide on a leader. One option is that this leader will be the only one to store the replicated data. Whenever such data is required, the leader is queried and provides the data accordingly. This can for example be done using *Scalable Collective Signing*, which has been discussed in section 3.4.2. This enables a leader to request the validation and signing of statements by a decentralised group of witnesses [15]. Another option is that whenever data inconsistency is detected, the version of the replicated data of the leader will be taken as the correct data. However, deciding on a leader is considered a hard task due to the given context. Hence this solution will not be seen as viable.

Comparing Data Versions This problem can also be solved using comparison of the versions of the replicated data whenever data inconsistency is detected. If data inconsistency is detected by a member, the member requests a halt of the currently performed protocol or process. A data inconsistency protocol is then started where each member of the coalition provides the other members with its version of the replicated data. Each member then compares each version of the corresponding data. The version that has the most occurrences will be deemed the correct version. To ensure each member chooses the same version, an extra check can be done where each member broadcasts the version it chose. If a member detects that another members has not chosen the same version, it can request a repeat of the data inconsistency protocol.

6 Complexity & Validation

In this Chapter a complexity analysis is provided regarding certain aspects of the Merkle Tree. Additionally a validation of the requirements set in Section 1.5.1 is provided and the security of our signature scheme is evaluated.

6.1. Complexity Analysis of the Merkle Tree

The complexity of two aspects of the Merkle Tree are analysed. First it is analysed what the complexity is when a message is added to the Merkle Tree. Next, an analysis is provided regarding proving a message to be in the Merkle Tree.

Adding a Message to the Merkle Tree The Merkle Tree is used to store the messages that were signed by a composition of the coalition. When a regular signature is issued by the coalition, the corresponding message is added to the currently used Merkle Tree. The Merkle Trees that are used, are created in such way that they are *Balanced Binary Trees*. In Section 4.3.5, it is explained how messages can be added to a Merkle Tree. Taking a Merkle Tree with $N-1$ messages (and therefore $N-1$ leaves), to add a message m, it is only necessary to perform at most one operation per level of the Tree. That means that the amount of operations to calculate the new Merkle Root is at most equal to the Height H of the Merkle Tree. Because the Merkle Tree is balanced and binary, the H is equal to $\lceil log(N) \rceil$. This means that the worst-case time-complexity of adding a message to the Merkle Tree is equal to $\mathcal{O}(\lceil log(N) \rceil)$.

Proving a Message is in the Merkle Tree As part of the solution to the Backlogging problem, it is necessary to be able to prove if a message m is present in a Merkle Tree. The Merkle Tree is specifically designed to make this as easy as possible. To prove that a message is present in a Merkle Tree, it is only necessary to prove that the message can be used to create the Merkle Root. This is done by following the path from the message to the Merkle Root in the Merkle Tree and performing the corrresponding calculations. Because the Merkle Tree is a *Balanced Binary Tree*, it is only required to perform at most one operation per level in the Tree. This means that the amount of operations to prove that a message is in the Merkle Tree, is at most equal to the Height H of the Merkle Tree. For a Merkle Tree with N messages, this is equal to $\lceil log(N) \rceil$. Hence the worst-case time-complexity of proving that a message is in a Merkle Tree is equal to $\mathcal{O}(\lceil log(N) \rceil)$.

6.2. Validation

6.2.1. Requirement Validation

In section 1.5.1, it is stated what the requirements are regarding the signature scheme. For each requirement, the requirement will be provided and an explanation will be given of how the requirement has been met.

1. *'At initialisation the dynamic coalition consists of n members, which jointly generate a distributed secret key and a public key using a suitable protocol.'*
   When the coalition is being initialised, the *Initialisation Protocol* is issued. This protocol is provided in Section 4.3.2. Here the members in $\mathcal{N}$, the current set of members of the coalition, first choose their random secret keyshares. Then the public keyshares are defined and distributed. Now the members

of the coalition can compute the public key. So this protocol allows the coalition to jointly generate a distributed secret key and a public key. Note that the secret key exists, but is distributed, since each member owns a part of the secret key.

2. *'The n members are able to create a signature together using a suitable (n,n)-signature scheme. This scheme makes it possible to create a signature without revealing the secret keyshares. Note that it should only be possible to create a valid signature if all n members participate in the signing procedure. This ensures that the members of the dynamic coalition have a veto.'*
   The signature scheme contains two signature issue protocols: the *Regular Signature Issue Protocol* and the *Block Signature Issue Protocol.* The Block Signature Issue Protocol, which is provided in Section 4.3.6, is only used when a new Block has to be created, whereas the Regular Signature Issue Protocol, which is provided in Section 4.3.5, is used when the coalition wishes to sign a message. Hence this requirement involves the Regular Signature Issue Protocol. This protocol allows the members in $\mathcal{N}$, the current set of members of the coalition, to sign a message. If all members correctly participate in the protocol, this will result in a valid signature. The validity of the signature can then be verified using the formula provided in Section 4.3.5. During the protocol, the secret keyshares are not revealed. This protocol allows the members of the coalition to create a signature together, without revealing the secret keyshares. Note that because the used signature scheme is a distributed n,n-signature scheme, it is only possible to create a valid signature if all members participate in the signing protocol.

3. *'It must be possible to expand the (n,n)-signature scheme to a $(n+1,n+1)$-signature scheme. The new member generates a new secret keyshare. To retain the original secret keyshares, the secret keyshares must be combined with the secret keyshare of the new member to generate a new distributed secret key and public key.'*
   When a member wishes to join the current coalition, the *Joining Protocol* is issued. This protocol is provided in Section 4.3.3. The current members share their public keyshares with the new member. Then the new member defines its own secret keyshare, calculates its public keyshare and broadcast this public keyshare. Now all members, including the new member, can calculate the new public key and the coalition is expanded. Again, the secret key exists, but is distributed, since each member owns a part of the secret key. This protocol allows the expansion of the signature scheme from a (n,n)-signature scheme to a $(n+1,n+1)$-signature scheme.

4. *'It must be possible to reduce the (n,n)-signature scheme to a $(n-1,n-1)$-signature scheme. The original secret keyshares, excluding the secret keyshare of the leaving member, must be used to generate a new distributed secret key and public key.'*
   When a member wishes to leave the coalition, the *Leaving Protocol* is issued. This protocol is provided in Section 4.3.4. The members of the coalition exclude the leaving member by redefining the public key, excluding the public keyshare of the leaving member. There is no need for communication, since all members already have the public keyshares needed to calculate the new public key. Again, the secret key exists, but is distributed, since each member owns a part of the secret key. This protocol allows the reduction of the signature scheme from a (n,n)-signature scheme to a $(n-1,n-1)$-signature scheme.

5. *'After a change has occurred in the composition of the coalition, either an addition or removal of a member, the n old members may not be able to create a valid signature. In other words, instances of backlogging must be prevented. One can assume that no member will give his secret keyshare to another member, since this can be used against him. Note that the members of the dynamic coalition all want to have a veto, in this light they do not trust each other.'*
   This requirement was the hardest to satisfy and is the basis of this thesis. In order to satisfy this requirement, the *Blockchain Technology* has been implemented in the signature scheme. When a group change occurs, where either the *Joining Protocol* or the *Leaving Protocol* is issued, a Block is created. This Block contains information of the coalition and the group change. It stores information about the messages that were signed by the current coalition and information identifying the new coalition. If consensus is achieved by the members of the current group composition about the contents of the Block, the Block is made immutable by creating a signature for the Block. Whenever a member suspects backlogging of a signature for a message, it first checks if the signature is valid using the formula provided in section 4.3.6. Then it checks if the signature was created during the time period the group composition was active. This is done by finding the Block that was created during the time the group

composition that supposedly signed the message was active. Then it is checked if the message is included in the Block. If this is the case, the message was signed during the time period the group composition was active, so the signature is valid. If this is not the case, the message was signed during a time period the group composition was not active, so the signature is an instance of Backlogging and is deemed invalid. A more detailed explanation can be found in Section 4.2.

6. *'It must be impossible for any member to find the entire secret key throughout the entire lifetime of the dynamic coalition. One can assume that no member will give his secret keyshare to another member, since this can be used against him. '*
   As stated in the requirement, it can be assumed that no member will give his secret keyshare to another member. Hence the only way to find the entire key is through the information provided in the protocols. However in the protocols, the secret keyshares are never used directly. The secret keyshares are used in a public form using the *Discrete Logarithm Problem*.

   In the *Initialisation Protocol*, the *Joining Protocol* and the *Leaving Protocol*, the secret keyshares are used in a public manner using the Discrete Logarithm Problem, which are the public keyshares.

   In step 3 of both the *Regular Signature Issue Protocol* and the *Group Signature Issue Protocol*, the secret keyshares are used. But again the Discrete Logarithm Problem is used. Additionally the value γ_i, which contains the secret keyshare, also contains β_i, thus contains two secret values. This means it should be close to impossible to find any of these secret values.

   Because in the protocols, the secret keyshares are used in a secure way, it can be assumed that it is impossible for any member to find the entire key throughout the entire lifetime of the coalition.

6.2.2. Security Evaluation

There are several security notions regarding the signature schemes. The most important security notion for a signature scheme is *Existential Unforgeability under a Chosen Message Attack* or *EU-CMA*[12]. For this security notion, an adversary has knowledge of the public key and has the ability to generate messages and receive their valid signatures. This is called a *Chosen Message Attack*. The adversary has no knowledge of the secret signing key. Additionally, the goal of the adversary is to create a valid signature for some message of its own choice. This is called *Existential Forgery*. A signature scheme is EU-CMA secure if for any polynomial time adversary $\mathscr{A}$, the probability of creating a valid signature for some message of its own choice, using a Chosen Message Attack, is negligible.

The signature scheme proposed in this thesis is based on the signature scheme created by *Park et al.*[21]. They first created a variant of the *Digital Signature Standard*[18], referred to as D_1, and then modified this to create a $(t,n)-$Threshold Digital Signature Scheme, referred to as D_2. They also discuss the security of the threshold signature scheme. It is explained that the proposed threshold signature scheme is just as EU-CMA secure as their DSS variant.

They do this in the following way. They define two adversaries: $\mathscr{A}_1$ for D_1 and $\mathscr{A}_2$ for D_2. $\mathscr{A}_1$ is able to use a signer of D_1 as an oracle. $\mathscr{A}_2$ can corrupt up to $k-1$ signers of D_2 and have the signature issuing protocol executed by any k or more signers. It is then proven that if an adversary $\mathscr{A}_1$ against D_1 exists that is able to perform existential forgery using chosen message attacks with a certain probability, there also exists and adversary $\mathscr{A}_2$ against D_2 that can do this with the same probability. This is done by showing that if $\mathscr{A}_1$ can perform this forgery against D_1, $\mathscr{A}_2$ can use $\mathscr{A}_1$ as a subroutine to perform this forgery on D_2. Then it is proven that if an adversary $\mathscr{A}_2$ against D_2 exists that is able to perform existential forgery using chosen message attacks with a certain probability, there also exists and adversary $\mathscr{A}_1$ against D_1 that can do this with the same probability. This is done using the same method where it is shown that if $\mathscr{A}_2$ can perform this forgery against D_2, $\mathscr{A}_1$ can use $\mathscr{A}_2$ as a subroutine to perform this forgery on D_1. Hence the authors conclude that the proposed threshold signature scheme is as EU-CMA secure as their variant of DSS.

The signature scheme proposed in this thesis is a modification of the $(t,n)-$ Threshold Digital Signature Scheme from *Park et al.*, where $t = n$. The modifications are mostly due to the Backlogging Prevention methods and the fact that $t = n$. For Key Generation, the use of polynomials is not needed, since it is not required that a subset of the members is able to create a valid signature. Additionally, the use of polynomials is not needed in the Signature Issue Protocols for the same reason. This simplifies these protocols. To facilitate the dynamic coalition environment, the *Joining Protocol* and the *Leaving Protocol* are added to the signature scheme to expand and reduce the signature scheme respectively. These protocols only change the keypair used by the coalition. Also several steps are added to the protocols to prevent backlogging, regarding

Blockchain functionality, and an extra signature issue protocol has been added to create signatures for Blocks. However these modifications do not influence the way the signatures are created or the way the validity of the signatures are verified. It only enables the use of Blocks and Merkle Trees to store information. Therefore it can be seen that the underlying signature scheme is still the $(t, n)-$Threshold Digital Signature Scheme from *Park et al.*, where $t = n$. Hence it can be assumed that the security of this signature scheme is also equal to the $(t, n)-$Threshold Digital Signature Scheme from *Park et al.*, regarding EU-CMA.

A similar security proof as in [21] can be provided for this signature scheme. However this is beyond the scope of this thesis, so this is noted as Future Work.

7 Discussion

This Chapter provides a discussion of the *Blockchain-based Signature Scheme*. The Signature Scheme can be divided into two parts: the actual signature scheme and the Blockchain implementation. First a discussion is provided regarding aspects of the Blockchain part of our signature scheme in Section 7.1. Then, in Section 7.2, it is discussed if using the underlying signature scheme that is used as basis for our signature is a correct decision. In Section 7.3, a comparison is provided of our signature scheme and the signature scheme introduced by *Van Elsas et al*[26]. And lastly, in Section 7.4, a limitation of our signature scheme is discussed.

7.1. Discussion on Blockchain Aspects

7.1.1. Forking the Blockchain

An aspect that can be considered as an issue in our signature scheme is the fact that a composition of a coalition can create a *Fork* of the then currently used chain. Take the scenario when a coalition change occurs, because a new user wants to join the coalition. The *Joining Protocol* is issued and a new composition is formed. However the old composition can decide to also continue as the old composition, to keep signing messages with their old secret key, by forking the blockchain without the latest Signed Block. This means the coalition exists with two different compositions, one with and one without the new member. However, due to the given scenario, this is not seen as an issue. Using this scheme, it should not be prohibited for a specific combination of users to form a coalition. Hence it should always be possible for a group to form a coalition. A fork can be seen as just a coalition between the corresponding members. And even if forking is prevented, the composition can always use the scheme to create a new coalition and sign messages using this new formation, with the only disadvantage that information on the Blockchain of the other coalition will be lost.

Even though this is not deemed an issue, a solution for forking a Blockchain will be discussed. The solution involves keeping a *global status* of the coalitions formed. Keeping a global state, every (potential) member has knowledge of the compositions of all coalitions and can detect forks when needed. However, this solution introduces several new issues. First of all, this means a list of all possible users need to be available, which is quite unlikely in the given scenario, unless a scope is taken where parties like the *UN* or *NATO* are involved. Also this introduces privacy issues. Since it is given that the users do not fully trust each other, privacy is a sensitive topic. One can imagine that being in a list of all possible users and having it publicly known to which coalition each user is participating in, which also needs to be managed by some type of organisation, can be a privacy-intensive issue.

7.1.2. Majority Voting and Veto

The Bitcoin environment considers faulty nodes. Faulty nodes do not cooperate with creating a new honest block, due to for example a malfunction or malicious intent. To tackle the issue of faulty nodes, the Proof-of-Work concept implements a form of *majority voting*[17]. The majority decision is represented by the longest chain. This chain has the greatest Proof-of-Work effort invested in it. Assuming that the majority of the nodes are honest, the longest chain can be seen as the honest chain.

However, for the Blockchain-Based Signature Scheme, the Proof-of-Work concept has been replaced by the Proof-of-Consensus concept. Our signature scheme does not consider faulty nodes, hence the Proof-of-Consensus concept is not able to tackle the faulty node issue. This is done on purpose. Using this signature scheme, if a signature is created, it is signed under the name of the coalition. This means all members of the coalition need to agree on creating the signature. In the given context, a faulty node is a member that does not cooperate in the signing protocol, resulting in the coalition not being able to provide a valid signature. However, due to the given context, each member of the coalition should have a *Veto*. It should not be possible for the coalition to sign a document, if at-least one member of the coalition does not agree on signing it. Hence, the fact that a member is able to prevent the creation of a valid signature is considered correct behaviour of the signature scheme. This behaviour is also reflected in requirement 2 in Section 1.5.1.

7.1.3. Blocksize Controversy in Bitcoin

Bitcoin has a blocksize controversy[10]. Increasing the blocksize increases the throughput of the system, but decreases the fairness of mining a block. This is due to the fact how the *Proof-of-Work* is set up. An explanation of the Proof of Work concept can be found in Section 2.2. Large miners will have an advantage over small miners, resulting in centralisation of the mining power. This breaks the idea of having a decentralised system and results into a loss of mining power.

The Blockchain-based Signature Scheme uses the *Proof-of-Consensus* instead of the Proof-of-Work. This replaces the idea of mining blocks to creating a combined signature for the block. Hence the aforementioned issue, is not applicable for this system. There is no limit to the blocksize, except the input limit of the hash function used in the *Block Signature Issue Protocol*, to hash the Block. An explanation of the limits regarding the hash function can be found in Section 5.4.

7.1.4. Storing the Messages of the Coalition

For storing messages on the Blockchain, the Merkle Tree has been implemented. However another solution for storing the messages was considered. For this solution, instead of storing the messages from regular signatures in a Merkle Tree, each message is stored in a separate block. This option was evaluated, because instead of having two separate signature issuing protocols, only one signature issuing protocol is necessary to be defined, where first the message is stored in a block and then the block is signed. This is corresponds to the *Block Signature Issue Protocol* provided in Section 4.3.6.

However the Merkle Tree proved to be more efficient. This is due to the reason the messages are stored. The messages are stored to track which messages has been signed by a composition of the coalition. This means it is only necessary to prove that a message is present in a set of all messages signed by a certain group composition of the coalition. The idea of a Merkle Tree suffices for this particular problem. Let's take the scenario where for simplicity each composition of the coalition signed N messages and M different group compositions has been active. As explained in Section 6.1, proving a message to be in a Merkle Tree, has a time-complexity of $\mathcal{O}(log(N))$. Hence, proving that a message has been signed by a group composition using a Merkle Tree has a time-complexity of $\mathcal{O}(M * log(N))$. Finding a message using the other solution would be $\mathcal{O}(M * N)$, since there will be at most $M * N$ blocks to traverse. For this scheme, the more efficient time-complexity deemed more valuable than reducing the amount of signature protocols from two to one. Hence the Merkle Tree has been adapted.

7.2. Discussion on the Signature Scheme used as Basis

The signature scheme presented in this thesis is based on the El Gamal type threshold digital signature scheme of *Park et al.*[21]. *Van der Lubbe et al.*[25] introduced a signature scheme that is a modification of the signature scheme of *Park et al.*. *Van Elsas et al.*[25] then modified the signature scheme of *Van der Lubbe et al.*, adding functionality and improving the security. Our signature scheme proposed in this thesis report is a modification of the signature scheme introduced by *Van Elsas et al.*, hence it is also based on the signature scheme of *Park et al.*. However it could be evaluated if the decision of using the signature scheme of *Park et al.*, that was made in previous work, is still a valid decision for this research. In this subsection several other options will be evaluated and it will be discussed if using the signature scheme of *Park et al.* is still a valid option.

A suitable signature scheme needs to meet some requirements. First of all, it needs to facilitate multiple users and it should be possible to ensure that valid signatures can be created only if all users participate in the signing protocol. Secondly, it should be possible to adjust the signature scheme to include or remove a member with minimal effort. Lastly, the signature scheme should not have a form of centralisation involved.

There are several signature schemes that enable the involvement of multiple users. A type of signature scheme that enables this are *Group Signature Schemes*[6]. The goal of a group signature scheme is to allow a group member to sign messages anonymously on behalf of the group. This goes against the second requirement, so this type is not a viable option.

Another type of signature schemes that allows multiple users are *Aggregate Signature Schemes*[3]. An aggregate signature scheme enables the aggregation of n signatures on n distinct messages from n distinct users into one single short signature. This single signature can then convince a verifier that the n users did indeed sign the n original messages. It is explained that this is useful for reducing the size of certificate chains and the message size in secure routing protocols. Although aggregate signature schemes could be a viable option, there is unwanted additional functionality that aggregate signatures provide. It is only needed to sign a single message, whereas aggregate signature schemes enable the signing of n messages. This might result into unnecessary computational overhead.

Boneh et al.[3] explain that aggregate signature schemes are related to *Multisignature schemes*, which allows a set of users to sign the same message, resulting in a single signature. *Burmester et al.*[5] also explain that multisignature schemes have advantages over multiple iterations of a single signature scheme. The length of the signature is usually smaller and the computational cost of generating and verifying the signature is usually lower. Therefore multisignature schemes seem to be a better fit for this research.

Multisignature schemes can be divided into two categories regarding signing order: *Serial Signing* and *Parallel Signing*[5]. For serial signing, the signing order can be detected by a verifier from a signature. Different signing orders result in different multisignatures. For parallel signing, the signing order cannot be detected by a verifier from a signature. The multisignature is group independent. In the context of this research, it is not necessary for the signature scheme to have a signing order. Additionally a serial signing order requires a specific order of signing of the users, which can be challenging given the provided context. Therefore a parallel signing order seems better for this research.

Burmester et al.[5] states that parallel multisignature schemes essentially are *Threshold Signature Schemes*. A (t, n) threshold signature scheme allows the creation of a valid signature if at least t out of n users participate in the creation of the signature. By taking $t = n$, it can be ensured that all users are needed to create a valid signature. Hence we can conclude that Threshold Signature Schemes are best fit as basis for this research.

Van der Lubbe et al.[25] were able to successfully modify the threshold signature scheme of *Park et al.*[21], to facilitate the dynamic coalition environment. It can be evaluated if another threshold signature scheme would be a better fit for this research. *Burmester et al.*[5] introduced a *Parallel Multisignature Scheme* based on a modified El Gamal signature scheme. Apart from some constraints, the keypair and partial keypairs are identical to the keypair and partial keypairs of the signature scheme of *Park et al.*. Additionally the signing protocol and verification protocol are computationally very similar. Hence it can be assumed that using this signature scheme will result into negligible differences. *Shoup*[24] introduces a RSA threshold signature scheme. However this signature scheme makes use of a *Trusted Dealer* to generate the keypair and distribute the keyshares. This goes against the third requirement, since the dealer is a form of centralisation. But *Damsgård et al.*[7] improved this signature scheme, removing the need of a Trusted Dealer. But expanding this signature scheme to include a new user is quite challenging. This is mostly due to the relation between the RSA public exponent and the RSA private exponent. When a group change occurs, specifically when a member leaves the group, it is necessary that new keyshares are defined and distributed, because of this relation. Hence it is more efficient to use a El Gamal based threshold signature scheme, since it is possible for each member to keep their keyshares.

Looking at the different types of signature schemes, it can be assumed that a threshold signature scheme is the appropriate type for a secure dynamic coalition environment. Even though there is a possibility that another threshold signature scheme might be a better fit for this research, it can be concluded that using the signature scheme of *Park et al.* as basis is still a correct decision.

7.3. Comparison with the Signature Scheme of *Van Elsas et al.*

Because the memberlog used in the signature scheme of *Van Elsas et al*[26] has been replaced by the chain of blocks, maliciously changing information is more challenging. In a blockchain, Blocks are linked to each other. For the Blockchain-based Signature Scheme, this is done by storing the signature of the previous Signed Block in the current Block. This results in the fact that changing a Block also changes all subsequent Blocks. Hence changing for example the set of messages signed by a group composition is more difficult in our signature scheme compared to the signature scheme proposed in by *Van Elsas et al.*

Our signature scheme prevents backlogging by the use of Merkle Trees and the Blockchain Technology where *Van Elsas et al.*[26] uses a combination of One-Way Accumulators and a memberlog. The One-Way

Accumulator is explained to be a combination of a hash function and a boolean array with a predefined size. A Merkle Tree does not require a predefined size and because only the Merkle Root of the Merkle Tree is stored in a Block, there is also an increase in efficiency regarding memory usage. Additionally, instead of using and storing the One-Way Accumulators and the memberlog separately, the proposed scheme stores all the information on the Blockchain. Each Block contains information about the composition, the public key y, and the regular messages, the Merkle Root. This makes the proposed scheme more efficient than the scheme in the original paper.

7.4. Limitation of Our Signature Scheme

The immutability of the blocks in the blockchain are based on the signatures created by the various compositions of the coalition. This means that a set of members that can comprise all the compositions of the coalition up until the current one, will be able to change the information stored in a block that one of these compositions signed. Take the scenario where the coalition is in phase o, with the set of members $\mathcal{N}$. Member k wants to join the coalition, so a new block is created and signed by $\mathcal{N}$ and phase $o+1$ starts. Then member l wants to join, so a new block is created and signed by $\mathcal{N} \cup \{k\}$ and phase $o+2$ starts. Now the coalition wants to change the content of the block that was signed in phase o by $\mathcal{N}$. The coalition changes and signs the block. This is possible because the members of $\mathcal{N}$ are still in the coalition. But now the signature of this block is different, since the block has been altered. This means the signature contained in the block of phase $o+1$ is also incorrect, since this does not point to the previous block anymore. But the coalition replaces this signature with the new signature and resigns this block. This is also possible, since the block was signed by $\mathcal{N} \cup \{k\}$ and they are also still in the coalition. Now the coalition can safely continue in phase $o+2$ with the changed blocks. This can be seen as a fraudulent act, but since all members in the coalition agree on this change, one can argue that this is a valid change, since the system is only used by the members of the coalition.

8 Conclusion

It is important that in military environments coalitions can be formed in a secure way. The dynamism in these environments introduce challenges that make forming and managing coalitions quite hard. In this thesis we conducted research in the field of signature schemes that are able to perform in a dynamic coalition environment. Additionally, we utilised Blockchain technology to increase the security of the signature scheme and decrease the possibility of misuse in the system. To evaluate the use of the Blockchain-based Signature Scheme in the real-world, we analysed the important aspects that could prove difficult for implementation the scheme.

In this Chapter we reflect on the research question. We discuss how the research goal is achieved, by answering the research question. Due to the scope of this thesis, some aspects of improvement or clarification remain untouched. At the end of this Chapter, points of improvements or interesting aspects are discussed as Future Work.

8.1. Reflection on the Research Question

The main research question of this research, as stated in Section 1.5, is:

How to construct a signature scheme that facilitates the needs of a dynamic coalition environment, and tackles the backlogging problem by employing Blockchain technology?

We then broke down the research question into two subquestions:

1. *How to construct a signature scheme that facilitate the needs of a dynamic coalition environment?*
2. *How to employ Blockchain technology to tackle the Backlogging problem?*

For the remainder of this section, we will answer the the research subquestions by examining the functionalities and implementation of our main contribution, the *Blockchain-based Signature Scheme*.

To answer the first subquestion, we first start by looking at the paper of *Van der Lubbe et al*[25]. An extensive explanation of this paper can be found in Section 3.1. The goal of this paper is also to design a signature scheme for a dynamic coalition environment. They succeeded by introducing a distributed (n, n)-signature scheme that is expandable to a distributed $(n+1, n+1)$-signature scheme. However the solution lacked the ability to be reducible, hence does not fully perform in a dynamic coalition environment.

Van Elsas et al.[26] continued this research and introduced a dynamic signature scheme that is both expandable and reducible. Hence their signature scheme is able to fully perform in a dynamic coalition environment. This signature scheme is based on the signature scheme proposed by *Park et al*[21].

The research conducted for this thesis is a continuation of the research conducted by *Van Elsas et al.* The Blockchain-based Signature Scheme uses the same primitives as the signature scheme of *Van Elsas et al.*, hence it is also based on the signature scheme proposed by *Park et al.* Our signature scheme also uses the same methods to expand and reduce, by introducing two extra protocols: the *Joining Protocol* and the *Leaving Protocol*. The Joining protocol is able to expand the signature scheme, by enabling a new member to join the coalition. This new member defines a secret keyshare and the coalition is able to define a keypair to include this keyshare. Hence the member has been admitted to the coalition. The Leaving protocol is able to reduce the signature scheme, by enabling a member to leave the coalition. By enabling the remaining members of the coalition to redefine the keypair to exclude the keypair of the leaving member, the Leaving protocol is capable of excluding a member from the coalition. Hence the Blockchain-based Signature Scheme is able to perform in a dynamic environment.

Taking the Problem Scenario into account, provided in Section 1.2, the dynamic coalition environment also introduces trust issues. This is reflected in the need of decentralisation, since no trusted third party is allowed, and the right to have veto by each member in the coalition. The Blockchain-based Signature Scheme does not use any kind of Third Party. Additionally, it can be seen that no distinguishment is made between the members of the coalition. All members contain similar knowledge, namely the public variables and a secret keyshare, and have equal involvement in the protocols. Therefore decentralisation is assured. The explanation for requirement 2 provided in Section 6.2.1, explains that the members of the coalition have a veto. This means the Blockchain-based Signature Scheme handles the trust issues the dynamic coalition environment introduces.

Hence the Blockchain-based Signature Scheme is a solution that answers the first subquestion.

However, it should be noted that some implementation aspects could introduce some sort of centralisation. These aspects have been discussed in Chapter 5. But it is also explained that if the network that potentially uses the Blockchain-based Signature Scheme is setup correctly, meaning that proper solutions are implemented, the system can still be used in a decentralised manner.

The dynamic coalition environment introduced another issue, the *Backlogging problem*. An explanation of the Backlogging problem is provided in Section 2.3. Due to the increasing popularity of the Blockchain technology and the ledger capabilities it provided, the second subquestion arose. To answer this subquestion, we have to examine the protocols of the Blockchain-based Signature Scheme.

The basic idea of the Blockchain has been implemented in the Blockchain-based Signature Scheme to tackle the Backlogging problem. In section 2.2, the three key characteristics of the Blockchain Technology that have been used in our solution are explained: The Chain of Blocks, the Merkle Tree and a Consensus Mechanism. The chain of blocks is defined by the fact that each Block contains the signature of the previously Signed Block. The Merkle Tree is used to store the messages that were signed by a group composition. Each group composition defines therefore a Merkle Tree and that is stored in the Block they defined. Lastly, the consensus mechanism used is the Proof-of-Consensus concept. Here in order to achieve consensus, a Block is signed by the active group composition. This ensures that the group composition agree on the Merkle Tree, hence agree that the messages in the Merkle Tree are the messages they have signed togtether, and on the group change that is about to occur.

Now using these Blockchain characteristics, it can be checked if a signature is an instance of Backlogging. This is explained in Section 4.2. The basic idea is that it is possible to know if a message was signed by a specific group composition, because this information is stored on the blockchain. Then if the validitiy of a signature is being verified, it is possible to check if the signature was issued during the time the corresponding group composition was active, therefore if it can be checked if the signature is an instance of backlogging.

Hence the Blockchain-based Signature Scheme uses Blockchain technology to tackle the Backlogging problem, and is a solution that answers the second subquestion.

In conclusion, the Blockchain-based Signature Scheme is able to perform in a dynamic coalition environment and uses Blockchain technology to tackle the Backlogging problem. Our solution succesfully answers the main research question.

8.2. Future Work

During the course of this research, several aspects, concepts and implementations were discussed that were deemed beyond the scope of this thesis research. They will be mentioned in this section as potential future work.

Extending the Blockchain Functionality The Blockchain Technology is used to create a distributed database. Having this implemented in the signature scheme introduces an opening for implementing potential functionality. Further research can be done to find out if the Blockchain implementation can be extended to perform several tasks, instead of one. Blockchain functionality is usually defined by storing information in a Block. Because the Blocks are already implemented, it is most likely only necessary to define the information that needs to be stored in the Block and define the points in time a Block needs to be defined and signed, to extend the Blockchain functionality.

Verma et al.[27] discusses several potential implementations Blockchain technology can have in a dynamic coalition environment. One of these examples is the idea of *Tear Sheets*. Assuming some kind of hierarchy exists, for this idea, Blockchain technology is used to filter documents on the rank the user has. Especially due to the military setting the problem scenario has in this research, where hierarchy is a common concept, researching this topic can be interesting.

Implementing the Blockchain-based Signature Scheme In Chapter 5, an analysis is provided regarding implementing our signature scheme. However the signature scheme has not been actually implemented yet. This can be very beneficial to this research as it can provide more insight in how our signature scheme can be implemented and how it performs. However, as can be concluded by our implementation analysis, it can be quite difficult to create a full implementation that is both secure and works according to the requirements set for the signature scheme.

Proving the Security of The Signature Scheme A security analysis is provided in Section 6.2.2. However the Blockchain-based Signature Scheme lacks a proper security proof. As stated in the Section, this is deemed as beyond the scope of this thesis. But to make sure our signature scheme is secure, this proof is required. Hence it is seen as future work to provide a security proof for our signature scheme.

Implementing a PKI using the Coalition The idea of a Public-Key Infrstratucture is explained in Section 2.1.2. Implementing a PKI is quite trivial when the coalition is already formed. Since the coalition is able to provide signatures, it can function as a *Distributed Certificate Authority (CA)*. Public and private keypairs can be distributed by the coalition. It is however not trivial when the coalition is being initialised. For initialisation, secure communication is needed, because the individual keyshares have to be broadcasted by the members. But since the coalition is not yet formed, signatures can not be provided, so the coalition is not able to function as a Distributed CA. This requires the members to communicate in an insecure manner and means the coalition is very vulnerable during initialisation. The problem scenario in prior work, namely in [25], explained that the goal of their research is to enable the coalition to perform as a Distributed Certificate Authority. This scenario could still be considered when continuing this research. However for implementation, this introduces extra challenges.

Replacing the whole Signature Scheme with Blockchain Further research can be done to see if the Blockchain model can replace the complete signature scheme that is based on the signature scheme of *Park et al*[21]. The purpose of a signature scheme can be seen as being able to create an immutable, unforgeable and verifiable connection between an identity and a document/message. By for example storing tuples of messages and verifiable identities on a Blockchain, the Blockchain can potentially provide the functionality of a signature scheme. By using a proper consensus mechanism, the information stored on the Blockchain can be seen as immutable. However, for it to be secure, a proper consensus mechanism has to be defined, verifiable identities have to be defined and used and the issue of unforgeability has to be addressed. Especially the latter can be quite difficult to tackle.

List of Figures

Bibliography

[1] Timo Bartkewitz. Building hash functions from block ciphers, their security and implementation properties. *PhD. Thesis, submitted to Ruhr-University Bochum*, Feb 2009.

[2] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak specifications. *Submission to NIST (round 2)*, pages 320–337, Oct 2009.

[3] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 416–432. Springer, May 2003.

[4] Ernest Brickell, David Pointcheval, Serge Vaudenay, and Moti Yung. Design validations for discrete logarithm based signature schemes. In *International Workshop on Public Key Cryptography*, pages 276–292. Springer, January 2000.

[5] Mike Burmester, Yvo Desmedt, Hiroshi Doi, Masahiro Mambo, Eiji Okamoto, Mitsuru Tada, and Yuko Yoshifuji. A structured elgamal-type multisignature scheme. In *International Workshop on Public Key Cryptography*, pages 466–483. Springer, January 2000.

[6] David Chaum and Eugène Van Heyst. Group signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 257–265. Springer, April 1991.

[7] Ivan Damgård and Maciej Koprowski. Practical threshold rsa signatures without a trusted dealer. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 152–165. Springer, May 2001.

[8] Yvo Desmedt. Society and group oriented cryptography: A new concept. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 120–127. Springer, April 1987.

[9] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, July 1985.

[10] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 45–59, Mar 2016.

[11] Eric Freudenthal, Tracy Pesin, Lawrence Port, Edward Keenan, and Vijay Karamcheti. drbac: distributed role-based access control for dynamic coalition environments. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 411–420. IEEE, July 2002.

[12] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[13] Alan F Karr, William J Fulp, Francisco Vera, S Stanley Young, Xiaodong Lin, and Jerome P Reiter. Secure, privacy-preserving analysis of distributed databases. *Technometrics*, 49(3):335–345, 2007.

[14] Cameron F Kerry and Patrick D Gallagher. Digital signature standard (dss). *FIPS PUB*, pages 186–4, July 2013.

[15] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, August 2016.

[16] Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS*, volume 2013, pages 11–32, June 2013.

[17] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, 2008.

[18] CORPORATE NIST. The digital signature standard. *Communications of the ACM*, 35(7):36–40, July 1992.

[19] Kaisa Nyberg. Fast accumulated hashing. In *International Workshop on Fast Software Encryption*, pages 83–87. Springer, February 1996.

[20] M Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems*. Springer Science & Business Media, 2011.

[21] Choonsik Park and Kaoru Kurosawa. New elgamal type threshold digital signature scheme. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 79(1):86–93, Jan 1996.

[22] Adrian Perrig, Ran Canetti, J Doug Tygar, and Dawn Song. The tesla broadcast authentication protocol. *RSA Cryptobytes*, 5:2–13, 2002.

[23] Ricky A Sewsingh, Jan C A van der Lubbe, and Merel J de Boer. A blockchain-based signature scheme for dynamic coalitions. In *Proceedings of the 2018 Symposium on Information Theory and Signal Processing in the Benelux*, pages 182–190. WIC & IEEE, May 2018. ISBN 978-90-365-4570-9.

[24] Victor Shoup. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 207–220. Springer, May 2000.

[25] Jan C A van der Lubbe, Merel J de Boer, and Zeki Erkin. A signature scheme for a dynamic coalition defence environment without trusted third parties. In *International Conference on Cryptography and Information Security in the Balkans*, pages 237–249. Springer, Oct 2014.

[26] Maarten Van Elsas, Jan C A Van der Lubbe, and Jos H Weber. A dynamic digital signature scheme without third parties. In *Proceedings of the 36th WIC Symposium on Information Theory in the Benelux and the 5th Joint WIC/IEEE Symposium on Information Theory and Signal Processing in the Benelux*, pages 89–95, May 2015.

[27] Dinesh Verma, Nirmit Desai, Alun Preece, and Ian Taylor. A blockchain based architecture for asset management in coalition operations. In *SPIE Defense+ Security*, pages 101900Y–101900Y. International Society for Optics and Photonics, May 2017.