CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Aggregated Set Membership Proofs

Aggregated Signature Based Set Membership Proofs Implemented to Verify Clients in Client and Server Verifiable Additive Homomorphic Secret Sharing

Master's thesis in Computer science and engineering

Hanna Ek

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Aggregated Set Membership Proofs

Aggregated Signature Based Set Membership Proofs Implemented to Verify Clients in Client and Server Verifiable Additive Homomorphic Secret Sharing

Hanna Ek

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Aggregated Signature Based Set Membership Proofs Implemented to Verify Clients in Client and Server Verifiable Additive Homomorphic Secret Sharing
Hanna Ek

Supervisor: Georgia Tsaloli and Katerina Mitrokotsa, Department of Computer Science and Engineering
Examiner: Katerina Mitrokotsa, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2021

Aggregated Signature Based Set Membership Proofs Implemented to Verify Clients in Client and Server Verifiable Additive Homomorphic Secret Sharing
Hanna Ek
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Abstract text about your project in Computer Science and Engineering.

Keywords: Aggregates Set Membership Proofs, Bulletproof, VAHSS, Range Proofs, cryptography, thesis.

Acknowledgements

Here, you can say thank you to your supervisor(s), company advisors and other people that supported you during your project.

Hanna Ek, Gothenburg, May 2021

Contents

List of Figures

List of Tables

1
Introduction

The digitalisation of our society leads to a need of cryptographic protocols to obtain online security and privacy. In many online applications users are required to provide some information to legitimise themselves. This could for example be to provide your membership number to verify that you are a member of a site.

In many application however it is sufficient to share more abstract information. Considering the example with the membership number. When proving that you are a member of a group instead of sharing your personal membership number, it would be sufficient to prove that you are one of the members, not specifying which one.

To illustrate the above idea consider two parties, Alice and Bob. Alice is a subscriber to Bobs paper. Alice wishes to convince Bob that she is a subscriber, but without revealing who she is. Bob has a list of all subscribers but without knowing who Alice is, he does not feel certain that she is on the list. Therefore Alice constructs a proof that her name is on the list. Bob receives this proof an checks its validity. Bob is now convinced that Alice's name is on the list, although Bob does not know which of the names it is, and allows Alice to get access to read the paper. For the remaining Alice will be denote as the prover and Bob as the verifier.

A cryptographic construction that allows a prover to convince a verifier that a secret is in a set without revealing the secret is *set membership proofs*. Constructing set membership proofs are computationally expensive. Consequently a lot of research has been done to reduce the computation required to construct and verify set membership proofs. Usually the constructions are optimised considering one prover and one verifier, as in the example above with Alice and Bob. Considering instead the setting: one verifier is responsible to verify multiple provers. An example of such a construction could be verification of clients in a VAHSS protocol [16]. In such an applications, using set membership proofs, the computational complexity of the verification grows linearly with the number of provers. This is due to that the verifier has to verify all received proofs separately.

A method for reducing the computation required by the the verifier, is to aggregated the proofs beforehand, such that the verifier only needs to verify the aggregated proof. The aggregated proof should be such that its validity should convince the verifier that all individual proofs are valid.

The aim of this paper is to investigate the possibilities to aggregate set membership, in order to reduce the computational complexity for verification of multiple set membership proofs.

Purpose

This primary purpose of this paper is to explore the possibilities to aggregate set membership proofs. This to have the computational complexity of the verification independent of the number of provers being verified. At first a general description of an aggregate set membership proofs is seeked and then the goal is, given such a description to provide a concrete construction of an aggregated set membership proofs.

A secondary purpose is to implement an aggregated set membership proof in a VAHSS construction, to verify the clients. Then compare such an implementation in terms of runtime with using the the state of the art range proofs, Bulletproofs, to verify clients in a VAHSS construction. To obtain this goal a method for extending the VAHSS construction, [16], to verify clients is has to be obtained.

Limitations

A limitation is that the exploration of obtaining a concrete construction of an aggregated set membership proofs is limited to an investigation if one specific set membership proof can be aggregated.

Contribution

Given the purpose above the results obtained in this paper was:

- A General description of the a aggregated set membership proof construction and the completeness, soundness and zero-knowledge properties such a construction should satisfy.
- A construction and implementation of specific construction of an aggregated set membership proof. Specifically a partial aggregation of the signature based set membership proof, presented in [4] is presented. The presented construction is proved to satisfy the stated correctness, soundness and zero-knowledge properties for an aggregated set membership proof.
- The VAHSS construction using homomorphic hash functions to verify servers, presented i [16], is modified to additionally verifying clients. The clients are verified using either a range proof or a set membership proof. A construction such that aggregated set membership proofs can be used to verify the clients is presented and its security discussed.
- Implementing of all proposed construction in Golang and runtime comparison for all construction results

Organisation

In chapter 2 the theoretical background is presented. First cryptographic principles is treated then a more detailed description of set membership proofs and range proofs is given. In chapter 3 a general description of aggregated set membership proofs is

defined. Based on this definition chapter 4 presents an aggregation of signature based set membership proofs. This construction is implemented and compared i runtime for different settings and to the state of the art Bulletproofs in chapter 5. Chapter 6 presents a client and server VAHSS, where clients are verified using for instance aggregated signature based set membership proofs. In the last chapter chapter 7, a summary is given and some final conclusions.

2
Background

This chapter will present the theory that will be used in the remainder of the paper. The chapter will be structured as, first notation, theorems, definitions and assumptions are stated. Then the cryptographic preliminaries used in this paper are explained. In section 2.2 two different types of zero knowledge proofs are presented, more precisely range proofs and set membership proofs.

2.1 Preliminaries

This section will introduce some notations and then cryptographic assumptions and definitions followed by several cryptographic preliminaries that will be relevant for this paper. The purpose of this section is to state information that will be refereed to and used as building blocks for the remaining of this paper.

Notation

To make the text more comprehensible notation that is used throughout the paper is introduced and defined here. Let $\mathbb{F} = \mathbb{Z}_p$ denote a finite field, where p is a large prime and let $\mathbb{G}$ denote a unique subgroup of order q. Define $g \in \mathbb{G}$ to be a group generator and $h \in \mathbb{G}$ a group element, such that $log_g\, h$ is unknown and h co-prime to p. The notation $y \in_R \mathbb{Y}$, means that an element y in chosen at random from the set $\mathbb{Y}$.

Definitions, Theorems and Assumptions

In this section definitions, theorems and assumption that will be refereed to and used throughout this paper is stated. The assumptions given here are the assumptions that all cryptographic constructions in this paper will rely on. The discrete logarithm assumption and the q-strong Diffie Hellman assumption define below does not hold in the presence of quantum computers. Thus the cryptographic constructions presented in this paper are not guaranteed to be secure post quantum.

Definition 1 (**Pseudorandom Function (PRF)**). *Let S be a distribution over $\{0,1\}^l$ and $F_s : \{0,1\}^m \to \{0,1\}^n$ a family of functions indexed by a string s in the support S. It is defined that $\{F_s\}$ is a pseudo random function family if, for every PPT (probabilistic polynomial time) adversary $\mathcal{A}$, there exists a negligable function*

ε such that:

$$|Pr[\mathcal{A}^{F_s}(\cdot) = 1] - Pr[\mathcal{A}^{R}(\cdot) = 1]| \leq \varepsilon,$$

where s is distributed according to S and R is a function sampled uniformly at random from the set of all functions mapping from $\{0,1\}^n$ to $\{0,1\}^m$.

Definition 2 (**Euler's totient function**). *The function $\Phi(n)$ is defined as the counter of the number of integers that are relative primes to n in the set $\{1, ..., n\}$. Note if n is a prime number $\phi(n) = n - 1$.*

Theorem 1 (**Euler's Theorem**). *For all integers x and n that are co-prime it holds that: $x^{\Phi(n)} = 1\,(mod\ n)$, where $\Phi(n)$ is Euler's totient function.*

From Theorem 1 it follows that for arbitrary y it holds that $x^{y\Phi(n)} = 1\,(\text{mod n})$.

Assumption 1 (**Discrete logarithmic assumption**). *Let $\mathbb{G}$ be a group of prime order q, further let $g \in \mathbb{G}$ be a group generator of $\mathbb{G}$ and $y \in \mathbb{G}$ be an arbitrary group element. Then it is infeasible to find $x \in \mathbb{Z}_q$, such that $y = g^x$*

Assumption 2 (**q-strong Diffie Hellman Assumption**). *Given a group $\mathbb{G}$, a random generator $g \in \mathbb{G}$ and powers $g^x, ..., g^{x_q}$, for $x \in_R \mathbb{F}$ and $q = |\mathbb{G}|$. It is then infeasible for an adversary to find $(c, g^{\frac{1}{x+c}})$, where $c \in \mathbb{F}$.*

Homomorphic Secret Sharing

Homomorphic secret sharing (HSS), first mentioned in [12], hides a secret x by splitting it into shares, such that any subset, $\mathcal{S}$, of shares smaller than a threshold τ, i.e $|\mathcal{S}| < \tau$, reviles no information about the original value of x. Let a secret x be split into m shares denoted x_i s.t $i \in \{1, ..., m\}$, then to reconstruct the value x at least τ shares has to be combined, this is called a (τ, m)-threshold scheme. In this paper the threshold will be equal to the number of shares, $\tau = m$.

Verifiable Additive Homomorphic secret sharing

In this paper *Verifiable Additive* homomorphic secret sharing (VAHSS) will be of interest.

Before describing VAHSS, AHSS will be desribed. The additivity property for a HSS means that the secret is reconstructed by computing the sum of at least τ shares, i.e $x = \sum_{i=1}^{\tau} x_i$. This is denoted Additive Homomorphic Secret Sharing (AHSS).

VAHSS is a construction of AHSS where many parties, refereed to as servers, collaborates to compute the sum of multiple clients secrets. Each client split their secret into the same amount of shares as there are serves and sends one share to each servers. The servers then computes and outputs a partial sum of all received shares. The final sum is computed by summing the servers outputs. For VAHSS constructions a proof σ is constructed that verifies that final sum is the correctly

computed sum of the clients secrets. In section 6.1 a specific construction of VAHSS is presented.

Homomorphic hash functions

Let $\mathcal{H}$ be a cryptographic hash function, $\mathcal{H} : \mathbb{F} \mapsto \mathbb{G}$. Any such function should satisfy the following two properties:

- **Collision-resistant** It should be hard to find $x, x' \in \mathbb{F}$ such that $x \neq x'$ and $\mathcal{H}(x) = \mathcal{H}(x')$.
- **One-Way** It should be computationally hard to find $\mathcal{H}^{-1}(x)$.

A homomorphic hash function should also satisfy the following property:

- **Homomorphism** For any $x, x' \in \mathbb{F}$ it should hold that $\mathcal{H}(x \circ x') = \mathcal{H}(x) \circ \mathcal{H}(x')$, where $\circ$ is either " $+$ " or " $*$ ".

A homomorphic hash function that satisfies the thee properties is the function $\mathcal{H}_1(x) : \mathbb{F} \mapsto \mathbb{G}$ and $\mathcal{H}_1(x) = g^x$ [17].

Pedersen Commitment scheme

A *Pedersen commitment* is a commitment to a secret $x \in \mathbb{F}$, defined as $\mathbb{E}(x, R) = g^x h^R$, where $R \in_R \mathbb{F}$, [11]. The Pedersen commitment satisfies the following theorem:

Theorem 2. *For any $x \in \mathbb{F}$ and for $R \in_R \mathbb{F}$, it follows that $\mathbb{E}(x, R)$ is uniformly distributed in $\mathbb{G}$. If two commits satisfys $\mathbb{E}(x, R) = \mathbb{E}(x', R')$ $x \neq x'$ then it must hold that $R \neq R' \bmod q$ and*

$$x - x' = (R - R') \log_g h \bmod p \quad \textit{Where } p \textit{ is the prime underlying the field } \mathbb{F}. \tag{2.1}$$

Proof. The statements of the theorem follows from solving for $log_g(h)$ in $\mathbb{E}(x, R) = \mathbb{E}(x', R')$ □

Theorem 2 implies that if someone knows the discrete logarithm of h with respect to g, i.e $log_g(h)$, this person is able to provide two equal commits, $\mathbb{E}(x, R) = \mathbb{E}(x', R')$ such that $x \neq x'$. Since the $log_g h$ is assumed to be unknown it is impossible to construct two equal commits hiding different secrets. This means that the Pedersen commitment scheme is computational binding under the discrete logarithm assumption, it is also perfectly hiding of the secret x, [11].

The Pedersen commitment is homomorphic. Hence for arbitrary messages $x_1, x_2 \in \mathbb{F}$, random values $R_1, R_2 \in_R \mathbb{F}$ and the commits $C_i = \mathbb{E}(x_i, R_i)$, $i \in \{1, 2\}$, it holds that $C_1 \cdot C_2 = \mathbb{E}(x_1 + x_2, R_1 + R_2)$.

Note that the Pedersen commitment is similar to the hash function $\mathcal{H}_1$ and the hash function can be seen as a generalisation of the Pedersen commitment.

A Pedersen commitment scheme can also be defined for vectors and is then called *Pedersen vector commitment*. Consider a n dimensional vector $\mathbf{x} \in \mathbb{F}^n$, let $\mathbf{g} = (g_1, ..., g_n) \in \mathbb{G}^n$ and $h \in \mathbb{G}$ where $\mathbb{G}$ is a group of order q as above. A commitment to the vector $\mathbf{x} = (x_1, ..., x_n)$ with the random value $R \in_R \mathbb{F}$ is then defined as $\mathbb{E}(\mathbf{x}, R) = \mathbf{g}^{\mathbf{x}} h^R = h^R \prod_{i=1}^{n} g_i^{x_i}$ and the commitment is a value in the one-dimensional group $\mathbb{G}$.

Bilinear mapping

Bilinear mapping (also commonly called bilinear pairing) maps two group elements from one groups to an element in another group. In this paper admissible bilinear mapping fulfilling Definition 3 will be used. In the definition given here two elements from the same group are mapped. Generally in the definition of admissible bilinear mapping, two elements from different groups are mapped to a third group, i.e $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, but in this paper it will always hold that $\mathbb{G}_1 = \mathbb{G}_2$ and hence the definition is given on this form.

Definition 3 (**Admissible Bilinear Map**). *Let $\mathbb{G}_1, \mathbb{G}_T$ be two multiplicative cyclic groups of prime order p such that there exist an admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$. Let $\mathbb{G}_1^* = \mathbb{G}_1 \backslash \{1\}$. Then the bilinear map e fulfils:*

- *Bilinear: for any group element $g \in \mathbb{G}_1^*$ and $a, b \in \mathbb{Z}_p$,*

$$e(g^a, g^b) = e(g, g)^{ab}$$

- *Non-degenerated: $e(g, g) \neq 1$*
- *The bilinear map is efficiently computable*

Bohen-Boyen Signatures

Considering a bilinear map e, defined in the previous section. The bilinear property of the mapping e can be used to create digital signatures. Bohen-Boyen presented a signature scheme that exploits the bilinear property of the mapping e to verify signatures [3]. In short the scheme works as, the signer knows the secret key x and distributes the public key g^x. To sign a message m the signer uses the secret key, x, and computes $\sigma = g^{1/(x+m)}$. This signature is $q - secure$ to forgery under the q-Strong Diffie Hellman Assumption. Verification is done by checking that $e(\sigma, y \cdot g^m) = e(g, g)$, which holds due to the bilinearity of e.

Zero knowledge proof

Zero-knowledge proofs (ZKP) is a cryptographic primitive that was first presented in [8]. The idea behind a ZKP is that after successfully performing a ZKP a certain statement about a secret x has been verified to be true (or false) without having revealed any other information about the secret x beyond the statement. Here only non-interactive ZKP that ensures proof of knowledge (PoK) is considered. Before closer defining what this means the set up and environment of ZKP protocols is defined

A ZKP consists of two parties a *prover* and a *verifier*, further assume both parties has access to the protocol parameters generated by a SetUp algorithm and a language $\mathcal{L} \in$ NP, additionally the prover knows a secret $x \in \mathcal{L}$. The prover constructs a proof that x belongs to $\mathcal{L}$, by using a witness w of x. Given the proof the verifier can in polynomial time determine if the proof is valid or not. For a ZKP to be non-interactive means that there is no communication required between the prover and verifier during the construction of the proof. PoK means that the verifier is not only convinced there exist a witness w but also that the prover knows such a witness.

A ZKP should fulfil the thee properties in Definition 4, informally the definitions states that: a correctly constructed proof of an instance $x \in \mathcal{L}$ should be accepted with probability 1, an incorrect proof of an instance $x \notin \mathcal{L}$ should have a negligible probability of being accepted and the verifier should learn nothing about the secret beyond the statement being proved.

Definition 4. *First define the two algorithms;* `Prove`(x, w)*,the algorithm for generating a ZKP of instance* $x \in \mathcal{L}$ *and witness* w*, and* `Verify`*, the verification algorithm of the ZKP. Such a ZKP scheme should fulfil the three properties:*

- ***Completeness*** *Given a witness* w *satisfying the instance* $x \in \mathcal{L}$*, it should hold that* `Verify`$($`Prove`$(x, w)) = 1$*.*
- ***Soundness*** *If the witness* w *does not satisfy the instance* $x \notin \mathcal{L}$*, then the probability Prob*[`Verify`$($`Prove`$(x, w)) = 1] < \varepsilon$*, for a sufficiently small* ε*.*
- ***Zero-knowledge*** *A proof system is honest verifier zero-knowledge if there exist a PPT algorithm* `Simulator` *having access to the same input as the algorithm* `Verify` *but not the provers input, such that output from the Simulator and* `Prove` *is indistinguishable, i.e have the same distribution given that* $x \in \mathcal{L}$*.*

This paper will consider zero knowledge range proof (ZKRP) and zero knowledge set membership proofs (ZKSM) where the statement that the prover convinces the verifier of is that the secret belongs to a predetermined range or set.

Fiat-Shamir heuristic

The Fiat-Shamir heuristic [1] can be used to convert an interactive protocol into a non-interactive. In this paper it will be used to construct non-interactive ZKP. A non interactive ZKP requires no communication between the prover and verifier during the construction of the proof. In interactive constructions the verifier sends a challenge $c \in_R \mathbb{F}$ to the prover that is included in the proof in order to convince the verifier that the prover did not cheat. The Fiat-Shamir heuristic replaces the random challenge sent by the verifier with the output of a hash-function of the partial-proof up to this point. The Fiat-Shamir heuristic converts an interactive ZKP to non-interactive while preserving security and full zero-knowledge relying on the random oracle model (ROM) [1] .

2.2 Set Membership Proof and Range Proofs

Zero knowledge set memberships proof (ZKSM) allows a prover to convince a verifier that the value of a secret is in an allowed set. Zero knowledge set membership proofs (ZKSM) does this without revealing any other information about the secret beyond the fact that the secret belongs to the set. Formally (ZKSM) prove the following statement:

$$\{(g, h \in \mathbb{G}, C; x, R \in \mathbb{F}) \ : \ C = g^x h^R \wedge x \in \Phi\}, \tag{2.2}$$

where Φ is some known set.

Zero knowledge range proofs (ZKRP) will also be considered, they prove that a secret belongs to a range, instead of a set, which is not as general as a set since

it has to be continuous. Zero knowledge range proofs (ZKRP) does not revealing any other information about the secret beyond the fact that the secret belongs to the range. Formally the ZKRP presented in this paper are constructed to prove the following statement about a secret x:

$$\{(g, h \in \mathbb{G}, C; x, R \in \mathbb{F}) \ : \ C = g^x h^R \wedge x \in \{\textit{"predetermined allowed range"}\}. \quad (2.3)$$

The range which x is proved to belong to in ZKRPs may vary between different constructions and will be more precisely defined below for the separate constructions.

Note that in the above statements it is assumed that x is the secret hidden in a Pedersen commitment, which is not a general requirement for range proofs and set membership proofs however only such proofs will be studied in this paper.

All set membership proofs and range proofs considered in this paper satisfied Definition 4.

Signature-based Set membership proof and range proof

In this section a specific implementation of a ZKSM is described. It will then be explained how this ZKSM can be extended to a ZKRP. The set membership proof described in this section will be refereed to as *signature based set membership* and the derived range proof will be refereed to as *signature based range proof*. Both the set membership proof and the derived range proof was originally presented in [6]. Both the proofs are modified compared to the original construction according to the Fiat-Shamir heuristic to be non-interactive.

Set membership proof

The idea behind the ZKSM is that for each element in the allowed set Φ there exist a public commitments, denoted A_i, $\forall i \in \Phi$. These commitments are made public in the SetUp phase by the verifier or some other party (not the prover). The prover aims to prove that the secret hidden by a pre-published Pedersen commitment, denoted C, is in the allowed set Φ. To prove this the public commitment representing the secret x, i.e A_x is selected and used to compute the value $V = A_x^\tau$, where $\tau \in_R \mathbb{F}$. The prover has to convince the verifier that 1) the published value V is indeed equal to A_x^τ where A_x is a signature of an element from the allowed set 2) the secret in the Pedersen commitment C is the same as the secret hidden by V.

In Construction 1 a detailed description of the PPT algorithms constructing the ZKSM described above is explained. The notation $e(\cdot, \cdot)$ in the construction refers to an admissible bilinear mapping as defined previously in section 2.1.

The ZKSM construction can be turned into a efficient zero knowledge range proof by rewriting the secret x in base u such that,

$$x = \sum_{j=0}^{l-1} x_j u^j.$$

Using this notation it follows that if $x_j \in [0, u)\ \forall j \in \mathbb{Z}_l$, then $x \in [0, u^l)$. A remark is that the subscript j goes between the number $[0, l-1]$ and not $[0, l]$. This ZKRP

Construction 1 : Non interactive set membership proof

Goal: Given a Pedersen commitment $C = g^x h^R$ and a set Φ, prove that the secret x in the commitment belongs to the set Φ without revealing anything else about x.

- **SetUp** $(1^\lambda, \Phi) \rightarrow (sk, pp)$
  Pick uniformly at random $\chi \in_R \mathbb{F}$ and put the secret key $sk = \chi$. Define $y = g^\chi$ and $A_i = g^{\frac{1}{\chi+i}}\ \forall i \in \Phi$, output y and $\{A_i\}_{i\in\Phi}$ and define the public parameters $pp = (y, \{A_i\}_{i\in\Phi})$.
- **Prove** $(g, h, C, \Phi) \rightarrow \Sigma = (V, a, D, z_x, z_\tau, z_R)$
  Pick uniformly at random $\tau \in_R \mathbb{F}$, choose from the set $\{A_i\}$ the element A_x and calculate $V = A_x^\tau$. Pick uniformly random three values $s, t, m \in_R \mathbb{F}$. Put $a = e(V, g)^{-s} e(g, g)^t$ and $D = g^s h^m$. Then use these two values to compute the challenge $c = \text{Hash}(C, V, a, D)$. Then given this challenge compute $z_x = s - xc$, $z_R = m - Rc$ and $z_\tau = t - \tau c$ then construct and publish the proof, $\Sigma = (V, a, D, z_x, z_\tau, z_R)$.
- **Verify** $(g, h, C, \Sigma) \rightarrow \{0, 1\}$
  Check if $D \stackrel{?}{=} C^c h^{z_R} g^{z_x} \wedge a \stackrel{?}{=} e(V, y)^c e(V, g)^{-z_x} e(g, g)^{z_\tau}$. If the equality holds the prover has convinced the verifier that the secret $x \in \Phi$ thus return 1 otherwise return 0.

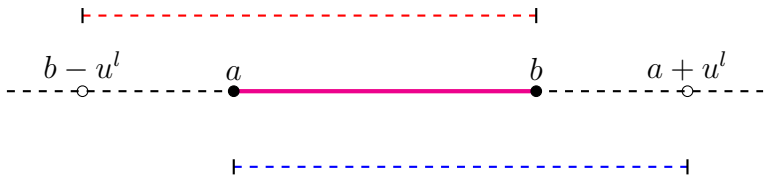


Figure 2.1: Illustration of generalisation to arbitrary intervals $[a, b]$ for range proofs

construction can be generalised to prove membership to an arbitrary interval $[a, b]$ where $a > 0$ and $b > a$, by showing that $x \in [a, a + u^l)$ and $x \in [b - u^l, b)$, since then the secret belong to the range $[a, b]$. Figure 2.1 illustrates the intuition and correctness of the statement. Proving $x \in [a, a + u^l)$ and $x \in [b - u^l, b)$ can easily be transferred into proving $x - a \in [0, u^l)$ and $x - b + u^l \in [0, u^l)$, since both a, b are public. In [7] an optimised implementation of the arbitrary range signature based range proof is presented, reducing the complexity with a factor 2. This rather small reduction is important when a verifier is required to check the range of multiple secret simultaneously.

Bulletproofs

Bulletproofs is a state of the art range proof that is integrated in many real-world protocols, for example they are often used in in cryptocurrencies. Bulletproof will be used in this paper for runtime comparison to an state of the art construction.

The Bulletproof was originally presented in [5]. Bulletproof are range proofs that a secret belongs to the range $[0, 2^n]$,m and are of logarithmic in n in size. The Bulletproof construction is based on the inner product argument. The inner product argument is an argument of knowledge that $\mathbf{s}, \mathbf{q}$ in a Pedersen vector commitment

$P_v = \mathbf{g}^{\mathbf{s}}\mathbf{h}^{\mathbf{q}}$ satisfies a given inner product denoted c.

The basic idea behind the construction of Bulletproofs is that a prover, given a Pedersen commitment $C = g^x h^R$, of the secret x, convince a verifier that the secret belongs to the interval $[0, 2^n)$. This is achieved by convincing the verifier that $\boldsymbol{x} \in \{0,1\}^n$ is the binary representation of the secret x, or equivalently that $x = \langle \boldsymbol{x}, \mathbf{2}^n \rangle$ and that the prover knows $\boldsymbol{x}$.

In this paper Bulletproof modified, compared to the original construction given in [5], according to the Fiat-Shamir heuristic to a non-interactive implementation is considered. A complete description of the construction of a non interactive Bulletproof is given in Appendix A in Construction 5. A non-interactive construction of the inner product argument is given in Construction 6.

3

Aggregated Set Membership Proofs

In applications where one verifier is responsible for verifying multiple provers set membership proofs the computational complexity of the verification is important to keep low. If the set membership proofs are proof of the statement in equation (2.2) the verifier would have to verify each proof individually. This would then leads to that the runtime for the verification grows linearly in the number of provers. The linear dependence of the number of provers for the computational complexity of the verification algorithm results in that this algorithm quickly becomes a bottleneck for such an application.

This motivates the question if an aggregation of set membership proofs can be provided to decrees the computations required to verify multiple provers simultaneously.

An aggregated set membership proof is a zero knowledge proof of the statement,

$$\Big\{(g, h \in \mathbb{G}, \{C_i\}_{i\in\mathcal{S}}; \{x_i\}_{i\in\mathcal{S}}, \{R_i\}_{i\in\mathcal{S}} \in \mathbb{F}^n) \; : C = \prod_{i\in\mathcal{S}} C_i \\ \wedge \prod_{i\in\mathcal{S}} C_i = g^{\sum_{i\in\mathcal{S}} x_i} h^{\sum_{i\in\mathcal{S}} R_i} \\ \wedge x_i \in \Phi \forall i \in \mathcal{S}\Big\}. \tag{3.1}$$

The statement implies that after successfully having performed an aggregated set membership proof the verifier is convinced that or all $i \in \mathcal{S}$ the Pedersen commitment C_i is a commitment to a secret belonging to the set Φ .

Further an construction of an aggregated set membership proof is a 5-tuple of PPT algorithms as described in Definition 5.

Definition 5 (**Aggregated non-interactive set membership proof**). *An aggregated set membership proof proves a statement on the form* (3.1) *and is a* 5*-tuple of PPT-algorithms (**SetUp**, **Prove**, **Aggregate**, **CalculateChallanges**, **Verify**.)*

- ***SetUp*** $(1^\lambda, \Phi) \rightarrow (pp, sk)$
  On the input 1^λ*, where* λ *is the security parameter , and the set* Φ *the algorithm outputs a secret key sk and public parameters pp.*
- ***Prove*** $(pp, i, C_i, \Phi) \rightarrow \Sigma_i$
  On input the public parameters pp, $i \in \mathcal{S}$ *denoting the index of the prover* p_i *and a Pedersen commitment of the* p_i*'s secret* x_i*. The algorithm outputs a*

polynomial time verifiable zero knowledge proof, denoted Σ_i*, that the secret* x_i *is in the set* Φ*.*

- ***Aggregate*** $(\{pp, \Sigma_i\}_{i\in\mathcal{S}} \rightarrow \Sigma_a$
  Given a set of set membership proofs $\{\Sigma_i\}_{i\in\mathcal{S}}$ *the algorithm aggregates the proofs into one zero knowledge proof of the statement in equation* (3.1) *denoted* Σ_a*.*
- ***CalculateChallenges*** $(\{C_i\}_{\in\mathcal{S}}, \{\sigma_i\}_{i\in\mathcal{S}}) \rightarrow \{c_i\}_{i\in\mathcal{S}}$
  On the input $\{\Sigma_i\}_{i\in\mathcal{S}}$ *the algorithm computes and outputs the challenges* $c_i = Hash(\Sigma_i)$ *for all* $i \in \mathcal{S}$*.*
- ***Verify*** $(pp, \Sigma_a, \{C_i\}_{i\in\mathcal{S}}, \{c_i\}_{i\in\mathcal{S}}) \rightarrow \{0, 1\}$
  On input the aggregated set membership proof public parameters, pp, aggregates proof, Σ_a*, and every provers Pedersen commitment,* $\{C_i\}_{i\in\mathcal{S}}$*, and challenge ,*$\{c_i\}_{i\in\mathcal{S}}$*, the algorithm outputs either* 1 *or* 0*.*

The algorithms (**SetUp**, **Prove**, **Aggregate**, **CalculateChallanges**, **Verify**) should satisfy the correctness, soundness and zero knowledge requirements in Defintion 6. These requirements can be seen as a modification of Defnition 4 to an aggregated ZKP of the statement in equation (3.1).

Definition 6. *The algorithms in Definition 5 should be such that they fulfil the completeness, soundness and zero-knowledge properties:*

- ***Completeness*** *Given a witness* Σ_i *satisfying the instance* $x_i \in \Phi$*, where* C_i *is a Pedersen commitment of*x_i*, for all* $i \in \mathcal{S}$*, it should hold that* $\texttt{Verify}(\texttt{Aggrgeate}(\{\texttt{Prove}(pp, i, C_i, \Phi)\}_{i\in\mathcal{S}})) = 1$*.*
- ***Soundness*** *If for any* $i \in \mathcal{S}$ *the witness* Σ_i *does not satisfy the instance* $x_i \notin \mathcal{L}$*, then the probability Prob*$[\texttt{Verify}(\texttt{Aggrgeate}(\{\texttt{Prove}(pp, i, C_i, \Phi)\}_{i\in\mathcal{S}})) = 1] < \varepsilon$*, for a sufficiently small* ε*.*
- ***Zero-knowledge*** *A proof system is honest verifier zero-knowledge if there exist a PPT algorithm* `Simulator` *having access to the same input as the algorithms* `Verify` *and* `Aggregate` *but not the provers input, such that output from the Simulator and* `Prove` *is indistinguishable, i.e have the same distribution given that* $x \in \mathcal{L}$*.*

4

Aggregated Signature Based Set Membership Proofs

In the previous chapter general description and definition of aggregated set membership proofs was described. In this chapter this general formulation will be used to propose a construction to aggregated a specific construction of set membership proof. More precisely it is investigated if the non-interactive construction of the set membership proposed in [4] can be aggregated according to Definition 5 such that it satisfies the completeness, soundness and zero knowledge in Definition 6. The specific set membership that will be considered is the signature based set membership in Construction 1.

4.1 Aggregation of signature based set membership proof

The considred set membership proofs can be used to construct a signature-based range proof as discussed previously and proposed in [4]. The strong similarity between the set membership proof and the derived range proof results in that the results obtained for the public signature based set membership proofs can easily be adjusted to also hold for the for the signature-based range proof. This would then result in an aggregated range proof construction. Such a construction is not further investigates in this paper but it is stressed that the results follow directly from the results for the public signature based set membership proofs.

To aggregate the public signature based set membership proofs possible constructions for the algorithm **Aggregate** is evaluated. The starting point for designing such a construction is that the aggregation of public signature set membership proofs according to the algorithm **Aggregate** should result in construction that satisfies the completeness requirement in Definition 6. Remark that it is assumed that the verification of the aggregated proof should be designed in the same way as the algorithm **Verify** in Construction 1. Having found an implementation of the algorithm **Aggregate** that satisfies the completeness it will then be examined under what assumptions the soundness and zero knowledge in Definition 6 is satisfied.

To investigate different designs of the algorithm **Aggregate** and whether it provides an aggregated set membership proof that satisfies the completeness the following is considered.

Two set membership proof Σ_1 and Σ_2 generated by the algorithm **Prove** in construction 1, recall that the proofs are on the form $\Sigma_i = (V_i, a_i, D_i, z_{x_i}, z_{\tau_i}, z_{R_i},)$

for $i = 1, 2$. In addition to the proofs also assume that C_i for $i = 1, 2$ are two Pedersen commitments published by the provers, p_1 and p_2, hiding the secrets x_1,x_2. The public parameters, pp, are generated by the algorithm **SetUp** in Construction 1 and that the challenges c_i, $i = 1, 2$ which can be computed as a hash of the proofs.

Since it will be assumed that the verification of an aggregated public signature based set membership proofs is performed as the algorithm **Verify** in Construction 1 the completeness property holds if an aggregates proof $\Sigma_a = (C, a, D, z_x, z_\tau, z_R)$ satisfies the two equalities $D = C^c h^{z_R} g^{z_x} \wedge a = e(V, y)^c e(V, g)^{z_x} e(g, g)^{z_\tau}$, where $C = C_1 C_2$ and $c = c_1 c_2$. Note that to reduce notation subscripts on the proof elements is omitted,

The simplest design of the algorithm **Aggregate** for the two considered set membership proofs would be define the aggregated set membership proof as the element-wise product of the two set membership proofs. This yields an aggregated set membership proof on the form, $\Sigma_a = (V, a, D, z_x, z_\tau, z_R)$, where $V, a, D, z_x, z_\tau, z_R$ are computed as,

$$\begin{aligned}
V =& V_1 V_2 = g^{\frac{\tau_1}{\chi + x_1}} g^{\frac{\tau_2}{\chi + x_2}} = g^{\frac{\tau_1}{\chi + x_1} + \frac{\tau_2}{\chi + x_2}} \\
a =& a_1 a_2 = \Big(e(V_1, g)^{-s_1}) e(g, g)^{t_1}\Big)\Big(e(V_2, g)^{-s_2}) e(g, g)^{t_2}\Big) \\
=& \Big(e(g, g)^{\frac{-s_1 \tau_i}{\chi + x_1}} e(g, g)^{t_1}\Big)\Big(e(g, g)^{\frac{-s_2 \tau_2}{\chi + x_2}} e(g, g)^{t_2}\Big) \\
D =& D_1 D_2 = (g^{s_1} h^{m_1})(g^{s_2} h^{m_2}) = g^{s_1 + s_2} h^{m_1 + m_2} \\
z_x =& z_{x_1} + z_{x_2} = (s_1 - c_1 x_1) + (s_2 - c_2 x_2) \\
z_R =& z_{R_1} + z_{R_2} = (m_1 - c_1 R_1) + (m_2 - c_2 R_2) \\
z_\tau =& z_{x_1} + z_{x_2} = (t_1 - c_1 \tau_1) + (t_2 - c_2 \tau_2)
\end{aligned} \tag{4.1}$$

Further assume that the challenges c_1 and c_2 has been calculated according to $c_i = Hash(C_i, V_i, a_i, D_i)$, $i = 1, 2$.

Next it will be investigated if the aggregated proof Σ_a derived as above satisfies the completeness in Definition 6. For the considered set membership proof this means that it must hold that, 1) $D \stackrel{?}{=} C^c h^{z_R} g^{z_x}$ and 2) $a \stackrel{?}{=} e(V, y)^c e(V, g)^{-z_x} e(g, g)^{z_\tau}$. Evaluating the first of the two equalities it follows that,

$$\begin{aligned}
LHS &= D = D_1 * D_2 = g^{s_1 + s_2} * h^{m_1 + m_2} \\
RHS &= C^c h^{z_R} g^{z_x} = (C_1 * C_2)^{c_1 c_2} h^{z_{R_1} + z_{R_2}} g^{z_{x_1} + z_{x_2}} \\
&= (g^{x_1} h^{R_1} g^{x_2} h^{R_2})^{c_1 c_2} h^{m_1 - R_1 c_1 + m_2 - R_2 c_2} g^{s_1 - x_1 c_1 + s_2 - x_2 c_2} \\
&= g^{c_1 c_1 (x_1 + x_2) + s_1 + s_2 - x_1 c_1 - x_2 c_2} h^{c_1 c_2 (R_1 + R_2) + m - R_1 c_1 - R_2 c_2}
\end{aligned}$$

$\implies$ LHS $\neq$ RHS.

The completeness property is not satisfied given an aggregation on the form in equation (4.1). Studying the LHS and RHS above it is noted that the equality does not hold since $c_1 c_2 (x_1 + x_2) = c_1 c_2 x_1 + c_1 c_2 x_2 \neq x_1 c_1 + x_2 c_2$. Further note that if $c_1 = c_2$ then this would be easy to fix. Clearly it cannot be guaranteed that the challenges for two different set membership proofs will be equal since they depend on randomness in the proof construction. However having pinpointing this source of inequality an aggregation that circumvent this issue is seeked.

Again consider the two set membership proofs Σ_1, Σ_2 as defined above, but now restrict the aggregation to only concern half the set membership proof. Namely only terms participating in the the first equality check, $D \stackrel{?}{=} C^c h^{z_R} g^{z_x}$, in the verification.

Note that if only a part of the proofs are aggregated it still leads to reduction of computational complexity for the verifier. The goal is now design the aggregation such that the aggregated proof satisfies $D = C^c h^{z_R} g^{z_x}$. Define the aggregated proof as, $\Sigma_a = (D, z_x, z_R)$, this means the proof does not contain the bilinear maps output a and the group element V, z_τ. Further let the the aggregated proof be computed according to,

$$\begin{aligned} D &= D_1^{c_2} \cdot D_2^{c_1} = (g^{s_1} h^{m_1})^{c_2} \cdot (g^{s_2} h^{m_2})^{c_1} = g^{s_1 c_2 + s_2 c_1} h^{m_1 c_2 + m_2 c_1} \\ z_x &= c_2 z_{x_1} + c_1 z_{x_2} = c_2(s_1 - x_1 c_1) + c_1(s_2 - x_2 c_2) = s_1 c_2 + s_2 c_1 - c_1 c_2 (x_1 + x_2) \\ z_R &= c_2 z_{R_1} + c_1 z_{R_2} = c_2(m_1 - R_1 c_1) + c_1(m_2 - R_2 c_2) = m_1 c_2 + m_2 c_1 - c_1 c_2 (R_1 + R_2) \end{aligned} \tag{4.2}$$

This construction of the aggregation will result in that the challenges always appears in a product which resolves the problem of them being different. Additionally also define the product of the challenges and commitments as,

$$\begin{aligned} c &= c_1 c_2 \\ C &= C_1 C_2 = (g^{x_1} h^{R_1})(g^{x_2} h^{R_2}) = g^{x_1 + x_2} h^{R_1 + R_2}. \end{aligned}$$

The aggregated proof Σ_a, computed according to equation (4.2) satisfies the equality $D = C^c h^{z_R} g^{z_x}$, this is seen below,

$$\begin{aligned} LHS = D &= D_1^{c_2} \cdot D_2^{c_1} = g^{s_1 c_2 + s_2 c_1} h^{m_1 c_2 + m_2 c_1} \\ RHS = C^c h^{z_R} g^{z_x} &= (C_1 C_2)^{c_1 c_2} h^{c_2 z_{R_1} + c_1 z_{R_2}} g^{c_2 z_{x_1} + c_1 z_{x_2}} \\ &= (g^{x_1 + x_2})^{c_1 c_2} h^{m_1 c_2 + m_2 c_1} g^{s_1 c_2 + s_2 c_1 - c_1 c_2 (x_1 + x_2)} \\ &= g^{(x_1 + x_2) c_1 c_2 - c_1 c_2 (x_1 + x_2) + s_1 c_2 + s_2 c_1} h^{m_1 c_2 + m_2 c_1} = g^{s_1 c_2 + s_2 c_1} h^{m_1 c_2 + m_2 c_1} \end{aligned}$$

$\implies$ LHS = RHS.

Aggregation of the two set membership proof defined according to equation (4.2) results in an aggregation that satisfies the completeness. The next step is to check if this aggregation can be extended to aggregate an arbitrary number of set membership proofs and still satisfy the completeness.

Consider $|\mathcal{S}|$ provers and consequently $|\mathcal{S}|$ set membership proofs denoted Σ_i, $i \in \mathcal{S}$. The aggregation in equation (4.2) extended to aggregate all proof results in the

aggregation procedure below,

$$
\begin{aligned}
D &= \prod_{i\in\mathcal{S}} D_i^{\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j} = \prod_{i\in\mathcal{S}} (g^{s_i}h^{m_i})^{\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j} = g^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i} h^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i} \\
z_x &= \sum_{i\in\mathcal{S}}\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\Big)z_{x_i} = \sum_{i\in\mathcal{S}}\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\Big)s_i - \Big(\prod_{j\in\mathcal{S}} c_j\Big)\sum_{i\in\mathcal{S}} x_i \\
z_R &= \sum_{i\in\mathcal{S}}\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\Big)z_{R_i} = \sum_{i\in\mathcal{S}}\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\Big)m_i - \Big(\prod_{j\in\mathcal{S}} c_j\Big)\sum_{i\in\mathcal{S}} R_i
\end{aligned}
\tag{4.3}
$$

Let $C = \prod_{i\in\mathcal{S}} C_i$ be the product of all Pedersen commitments and $c = \prod_{i\in\mathcal{S}} c_i$ the product of the challenges. The partly aggregated set membership proof $\Sigma_a = (D, z_x, z_r)$ computed according to equation 4.3 satisfies $D = C^c h^{z_R} g^{z_x}$,

$$
\begin{aligned}
LHS =& D = g^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i} h^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i} \\
RHS =& C^c h^{z_R} g^{z_x} = \Big(\prod_{i\in\mathcal{S}} C_i\Big)^{\prod_{i\in\mathcal{S}} c_i} h^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i} \\
& g^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i - \left(\prod_{j\in\mathcal{S}} c_j\right)\sum_{i\in\mathcal{S}} x_i} \\
=& \Big(g^{\sum_{i\in\mathcal{S}} x_i} h^{\sum_{i\in\mathcal{S}} R_i}\Big)^{\prod_{i\in\mathcal{S}} c_i} h^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i - \left(\prod_{j\in\mathcal{S}} c_j\right)\sum_{i\in\mathcal{S}} R_i} \\
& g^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i - \left(\prod_{j\in\mathcal{S}} c_j\right)\sum_{i\in\mathcal{S}} x_i} = g^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i} h^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i}
\end{aligned}
$$

$\implies$ LHS = RHS.

Above it has been seen that arbitrary many set memberships proofs can be aggregated according to equation (4.3) such that $D = C^c h^{z_R} g^{z_x}$, holds after the aggregation. This results in a technique to aggregate set membership proofs such that when verifying the signature based set membership proofs the first equality, $D = C^c g^{z_x} h^{z_R}$, only need to be verified once instead of once once for each proof.

Next examine if the entire construction of public signature based set membership proofs can be aggregated. This implies that in addition to the above aggregation the second equality should also be satisfied after aggregating. This leads to that aggregated proof should be equal to $\Sigma_a = (a, V, D, z_x, z_\tau, z_R)$ where the values a, V, z_x, z_τ satisfies the equation $a = e(V, y)^c e(V, g)^{-z_x} e(g, g)^{z_\tau}$.

Again consider two set membership proofs Σ_1, Σ_2 computed according to the algorithm **Prove** in Construction 1. Examine if an aggregated proof $\Sigma_a = (a, V, D, z_x, z_\tau, z_R)$ computed according to equation (4.1) satisfies the second equality of the verification under the assumption $c_1 = c_2$. If so it would imply that the same trick as above, namely aggregating such that the challenges appear only as a product, would yield a method for fully aggregating the public signature based set membership proofs.

After fully extending the expression $a \stackrel{?}{=} e(V,y)^c e(V,g)^{-z_x} e(g,g)^{z_\tau}$, see Appendix B for the details of the calculations, it is realised that the terms,

$$e(V_1,g)^{z_{x_2}} e(V_2,g)^{z_{x_1}} = e(g,g)^{\frac{\tau_1}{\chi+x_1}(-s_2+x_2 c)+\frac{\tau_2}{\chi+x_2}(-s_1+x_1 c)},$$

on the right side of the equality are not cancelled leading to that the equality in the verification is not satisfied.

This concludes that even if $c_1 = c_2$ the completeness property will not hold after aggregating according to equation (4.1). This implies that in order to aggregate the whole set membership proof additional tricks are required. Several attempts to construct an aggregation that satisfy the completeness property has been made without any success. Therefore whether it can be done or not is left unanswered. What can be seen as an indicator to that it is not be possible is that in the algorithm **Verify** in the derived signature based range proof the bilinear mappings are checked separately for each $j \in \mathbb{Z}_l$ [4].

4.2 Construction

The previous section proposed a design of the algorithm **Aggregate**, such that the public signature based set memberships proofs could be partly aggregated. A full description of the aggrgeated public signature based set membership in given in Construction 2. The algorithm **Prove** in the aggregated set membership proof is run by all proving parties wanting to prove that their secret is in the set. Then the aggregating party takes all the set membership proofs as input and outputs an aggregated proof. The verifier takes the aggregated proof as input, outputs either 1 or 0. Due to the aggregation the verifier checks the equality $D = C^c h^{z_R} g^{z_x}$ once instead of once for each proving party in the protocol.

In Construction 2 the challenges are given as input to the verifier, unlike the original set membership proof where the verifier calculates the challenge $c_i = Hash(a_i, D_i)$. The reason for this is that the values arguments required to compute tha challenges are not not known to the verifier. This raises the question about which party that should be responsible for calculating the challenges. It is not desired that the party performing the aggregation also computes the challenger since this opens up for cheating, for the same reason should the proving party not compute the challenge. Thus, the challenges are computed by a different party, independent of both the proving-parties and the aggregating party, and finally multiplied together by the verifier. Another possibility is to include the values entire set membership proofs $\{\Sigma_i\}_{i\in\mathcal{S}}$ in the input to the verification, leading to that the verifying party can compute the challenges. This would consequently increase the computations for in the verification. For the rest of this paper, if noting else mentioned, it is assumed that the algorithm **CalculateChallanges** is performed according to Construction 2 by a trusted party.

Many aggregating parties

The computational complexity for the algorithm **Aggregate** is linear in the number of set membership proof that are aggregated. To reduce the computation load of the

Construction 2 : Aggregation of non interactive set membership proof

Goal: Given the Pedersen commitments $C_i = g^{x_i}h^{R_i}$ for $i \in \mathcal{S}$. The goal is to prove that all secrets x_i belongs to the set Φ. Without revealing anything else about the secrets x_i.

- **SetUp** $(1^\lambda, \Phi) \rightarrow (sk, pp)$
  Let g be a generator of the group $\mathbb{G}$ and h and element in the group such that $log_g(h)$ is unknown. Pick uniformly at random $\chi \in_R \mathbb{F}$ and put $sk = \chi$. Define $y = g^\chi$ and $A_i = g^{\frac{1}{\chi+i}}$ $\forall i \in \Phi$, output $pp = (g, h, y, \{A_i\}_{i\in\Phi})$.
- **Prove** $(pp, i, C_i, \Phi) \rightarrow \Sigma_i$
  Pick uniformly at random $\tau_i \in_R \mathbb{F}$, choose from the set $\{A_i\}$ the element A_{x_i} and calculate $V = A_{x_i}^{\tau_i}$. Pick uniformly random three values $s_i, t_i, m_i \in_R \mathbb{F}$. Put $a_i = e(V_i, g)^{-s_i}e(g, g)^{t_i}$, $D = g^{s_i}h^{m_i}$, and $c = \text{Hash}(C_i, V_i, a_i, D_i)$. Finally compute $z_{x_i} = s_i - x_i c_i$, $z_{R_i} = m_i - R_i c_i$ and $z_{\tau_i} = t_i - \tau_i c_i$ then construct and publish $\Sigma_i = (V_i, a_i, D_i, z_{x_i}, z_{\tau_i}, z_{R_i})$.
- **Aggregate** $(pp, \{\Sigma_i\}_{i\in\mathcal{S}}) \rightarrow \Sigma_a$
  Given a set of set membership proofs $\{\Sigma_i\}_{i\in\mathcal{S}}$. Aggregate the values $(\{D_i\}_{i\in\mathcal{S}}\{z_{x_i}\}_{i\in\mathcal{S}}\{z_{r_i}\}_{i\in\mathcal{S}}) \mapsto D_a, z_{x_a}, z_{R_a}$ according to equation (4.3). Then construct and publish the aggregated proof $\Sigma_a = (\{V_i\}_{i\in\mathcal{S}}, \{a_i\}_{i\in\mathcal{S}}, D_a, \{z_{x_i}\}_{i\in\mathcal{S}}, z_{x_a}, \{z_{\tau_i}\}_{i\in\mathcal{S}}, z_{R_a})$.
- **CalculateChallenges** $(\{C_i\}_{i\in\mathcal{S}}\{\Sigma_i\}_{i\in\mathcal{S}}) \rightarrow \{c_i\}_{i\in\mathcal{S}}$
  For all $i \in \mathcal{S}$ parse the proof Σ_i, then compute the challange $c_i = Hash(C_i, V_i, a_i, D_i)$. Finally output the set of all challenges $\{c_i\}_{i\in\mathcal{S}}$.
- **Verify** $(pp, \Sigma_a, \{C_i\}_{i\in\mathcal{S}}, \{c_i\}_{i\in\mathcal{S}}) \rightarrow \{0, 1\}$
  Compute the product of the challenges $c = \prod_{i\in\mathcal{S}} c_i$. Check if $D_a \stackrel{?}{=} \left(\prod_{i\in\mathcal{S}} C_i\right)^c h^{z_{Ra}} g^{z_{xa}} \wedge a_i \stackrel{?}{=} e(V_i, y)^c_i e(V_i, g)^{-z_{x_i}} e(g, g)^{z_{\tau_i}}$ for all $i \in \mathcal{S}$. If the equalities holds the provers has convinced the verifier that $x_i \in \Phi$ for all x_i such that $i \in \mathcal{S}$ return 1 otherwise return 0.

aggregating party it is possible to split the aggregation between multiple aggregating parties.

Let the set $\mathcal{K}$ in represents the set of parties performing the aggregation, and the set $\mathcal{S}_k$ for $k \in \mathcal{K}$ represents the assigned set membership proofs to aggregate to the k^{th} aggregating party. The sets $\mathcal{S}_k$ are such that $\bigcup_{k\in\mathcal{K}} \mathcal{S}_k = \mathcal{S}$ and $\mathcal{S}_i \bigcap \mathcal{S}_j = \emptyset$ for all $i, j \in \mathcal{K}$ such that $i \neq j$.

To split the aggregation between the aggregating parties in the set $\mathcal{K}$, the algorithm **Aggregate** is performed once for each party in the set $\mathcal{K}$ on the input $(pp, \{\Sigma_i\}_{i\in\mathcal{S}_k})$ and the output is Σ_{a_k}. Then the verification algorithm gets the input $(pp, \{\Sigma_{a_k}\}_{k\in\mathcal{K}}, \{C_i\}_{i\in\mathcal{S}}, \{c_i\}_{i\in\mathcal{S}})$ and performs the verification once for each aggregated proof. A construction of this is seen in appendix C in Construction 7. By letting the set of aggregating parties $\mathcal{K}$ consist of one element and $\mathcal{S}_k = \mathcal{S}$ construction 2 and 7 would be equal.

An illustrative figure of the aggregation procedure will look is given in Figure 4.1. Each client generates a set membership proof, and than sends it to its assigned server. Then each server will aggregate the received set membership proofs according to the algorithm **Aggregate** in Construction 2 and send the aggregated proof to the verifier. Finally the verifier verifies all the aggregates proofs and the severs computations.

In Figure 4.1 the computation of the challenges is done by an independent party, next two approaches to compute the challenges without introducing any new parties. The first alternative is that the verifier computes the challenges, this would require that the verifies additionally gets $\{D_i\}_{i\in\mathcal{N}}$ as input. The second alternative is that the servers computes the challenges, however as discussed above the same party that aggregates the proofs should not compute the challenges. To use the server for the computation of challenges assume that the servers are linked in a closed chain. Then each server computes the challenges for the set membership proofs aggregated by the consecutive server.

4.3 Completeness, Soundness and Zero-Knowledge

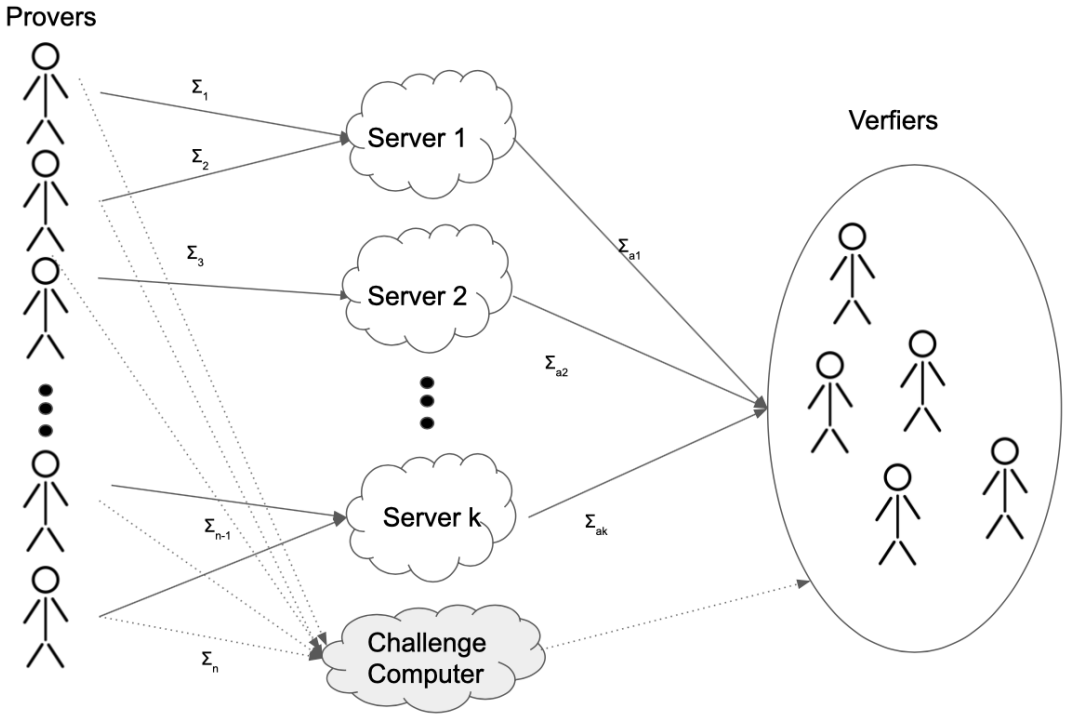
Trusted aggregating party

This section will investigate if the proposed aggregated set membership proof presented in Construction 2 fulfils the completeness, soundness and zero-knowledge requirements stated in Definition 6 under the assumption that the aggregation has been done according to equation (4.3) by one trusted party.

Completeness follows from the argument given above, where it has been seen that $D \stackrel{?}{=} C^c h^{z_R} g^{z_x}$ given all parties where honest, and the completeness of the public signature based set membership proof. Thereby Construction 2 satisfies the completeness in Definition 6.

Zero-knowledge follows directly from the zero knowledge property of Construction 1 and by realising that multiplying and adding elements which perfectly hides the secret x_i, with other elements independent of the secret will not reveal any information about the secret. This leads to that it remains to see if the soundness property holds for Construction 2.

Figure 4.1: Provers outsourcing their set membership proofs to their assigned servers that in turn outputs aggregated set membership proof.



The verification in Construction 1 consists of two equality checks. The first equality check $D \stackrel{?}{=} C^c h^{z_R} g^{z_x}$ convinces the verifier that the commitment and the proof element z_x hides the same secret. The second equality check $a \stackrel{?}{=} e(V,y)^c (V,g)^{z_x} e(g,g)^{z_\tau}$ convinces the verifier that the secret used to compute z_x is the same as the secret hidden in V and the security of Bohen-Boyen signatures gives that this secret is a member of the set Φ.

Considering the second equality check of the set membership proof, that has not been aggregated, the soundness follows directly from the soundness of construction 1. This is since this part of the proof is send directly from the provers to the verifier.

The equality check $D \stackrel{?}{=} C^c h^{z_R} g^{z_x}$ in the original set membership construction as seen above servers the purpose of ensuring the verifier that the secret hidden in the commitment C is the same as the secret used to construct the value z_x. This property need to be checked that it remains after the aggregation. Meaning that the verifier should be sure that all secrets hidden in the commitments, are equal to the secrets used to compute the values z_{x_i}, in turn used in the aggregation to construct z_x.

To investigate this, consider two set membership proofs, Σ_1 and Σ_2, on the form $(V_i, a_i, D_i, z_{x_i}, z_{\tau_i}, z_{R_i})$ for $i = 1, 2$. Given these proofs and the Pedersen commitment published by the provers the trusted aggregating party computes $\Sigma_a = (\{V_1, V_2\}, \{a_1, a_2\}, D, z_x, \{z_{x_1}, z_{x_2}\}, \{z_{\tau_1}, z_{\tau_2}\}, z_R))$, according to algorithm **Aggregate** in construction 2. Given this set up, it will be investigated if it can hold that: $D = (C_1 C_2)^c h^{z_R} g^{z_x}$, where $C_i = g^{\tilde{x}_i} h^{\tilde{R}_i}$ and $z_{x_i} = s_i - c_i x_i$ such that $x_i \neq \tilde{x}_i$ for i

equal to either 1, 2 or both. Resulting in that the below equations must be equal while assuming that $\tilde{x}_i \neq x_i$,

$$\begin{aligned} LHS =& g^{s_1c_2+s_2c_1}h^{m_1c_2+m_2c_1} \\ RHS =& g^{c_1c_2(\tilde{x}_1+\tilde{x}_2)}h^{c_1c_2(\tilde{R}_1+\tilde{R}_2)}g^{c_2(s_1-c_1x_1)+c_1(s_2-c_2x_2)}h^{c_2(m_1-c_1R_1)+c_1(m_2-c_22)} \\ =& g^{s_1c_2+s_2c_1}h^{m_1c_2+m_2c_1}g^{c_1c_1(\tilde{x}_1+\tilde{x}_2)-c_1c_2(x_1+x_2)}h^{c_1c_1(\tilde{R}_1+\tilde{R}_2)-c_1c_2(R_1+R_2)}. \end{aligned}$$

Under the assumption that the aggregation was done correctly, one of the following cases has to hold in order for $LHS = RHS$,

1. $x_1 = \tilde{x}_2$ and $x_2 = \tilde{x}_1$
2. $(x_1 + x_2) = (\tilde{x}_1 + \tilde{x}_2) \bmod \Phi(N)$

In the first case the provers are cheating, since they do not use the same secret in the commitments as in z_{x_i}. Despite cheating provers the result from the protocol is unaffected. This is since a cheating parties in this case only achieves to having their secret being committed to by another prover, as oneself commits to this prover's secret. It is not of relevance who committed what as long as all secrets is verified to be in the set.

The second case shows that two provers can collaborate in such a way that they ensured the verifier that the secrets x_1 and x_2 are in the set, by falsely proven that the secrets in x_1, x_2 are equal to $\tilde{x}_i, \tilde{x}_2$. I.e the equality $D \stackrel{?}{=} (\prod_{i=1}^{2} C_i)^c h^{z_R} g^{z_x}$ holds true without $\tilde{x}_i = x_i$. Thereby using $\tilde{x}_1$ and $\tilde{x}_2$ belong to the set Φ and x_1, x_2 not being in the set, the provers has successfully cheated. Thus it has to be assumed that provers cannot communicate in order for the soundness property to hold for the aggregated set membership.

The arguments above for the soundness of Construction 2 is formally stated and proved in Theorem 3.

Theorem 3. *Assumed that the aggregation is done according to algorithm* ***Aggregate*** *in Constrcution 2 by a trusted party, that provers cannot communicate and that the set membership proof in Construction 1 is sound according to Definition 4. Then the aggregated set membership proof in Construction 2 satisfies the soundness in Definition 6.*

Proof. Let T denote the index-set of malicious provers. Assume that a prover c_i, $i \in T$ can construct a set membership proof for the Pedersen commitment $C_i = g^{\tilde{x}_i}h^{\tilde{R}_i}$ such that $\tilde{x}_i \neq x_i$ and $z_{x_i} = s_i - c_i x_i$.

The soundness of set membership proofs implies that the values of the malicious provers proofs used in the non aggregated equality check has to be computed according to the algorithm **Prove** in Construction 2. Assume further that all honest provers computes their set membership proof according to **Prove** and for the malicious provers the set membership proof and Pedersen commitment must, due to the

soundness of Construction 1, fulfil:

$$\begin{aligned} V_i &= A_{x_i}^{\tau_i} \\ a_i &= e(V_i, g)^{-s_i} e(g, g)^{t_i} \\ D_i &= g^{\tilde{s}_i} h^{\tilde{m}_i} \\ z_{x_i} &= s_i - x_i c_i \\ z_{\tau_i} &= t_i - \tau_i c_i \\ z_{R_i} &\in \mathbb{F} \\ C_i &= g^{\tilde{x}_i} h^{\tilde{R}_i}, \end{aligned}$$

where $\tilde{x}_i \neq x_i$ and $x_i \in \Phi$. It is not required that $\tilde{s}_i = s_i$, $\tilde{m}_i = m_i$, $\tilde{R}_i = R_i$ nor required that they are not equal.

In order for the algorithm **VerifyAggregated** to validate true it has to hold that $D = C^c h^{z_R} g^{z_x}$, where D, z_R, z_x is the aggregation according to equation (4.3) of all provers, honest and malicious, set membership proofs. For this to be an equality it has to hold that the left hand side and right hand side below are equal,

$$LHS = D = g^{\sum_{i\in T}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)\tilde{s}_i + \sum_{i\in\mathcal{S}\setminus T}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i} h^{\sum_{i\in T}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)\tilde{m}_i + \sum_{i\in\mathcal{S}\setminus T}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i}$$

$$\begin{aligned} RHS = C^c h^{z_R} g^{z_x} &= \left(g^{\sum_{i\in T}\tilde{x}_i + \sum_{i\in\mathcal{S}\setminus T} x_i} h^{\sum_{i\in T}\tilde{R}_i + \sum_{i\in\mathcal{S}\setminus T} R_i}\right)^{\prod_{j\in\mathcal{S}} c_j} h^{\sum_{i\in T}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right) z_{R_i}} \\ &\quad h^{\sum_{i\in\mathcal{S}\setminus T}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i - \sum_{i\in S\setminus T}\left(\sum_{i\in\mathcal{S}} c_j\right)R_i} g^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i - \left(\prod_{j\in\mathcal{S}} c_j\right)\sum_{i\in\mathcal{S}} x_i} \\ &= g^{\prod_{j\in\mathcal{S}} c_j\left(\sum_{i\in T}\tilde{x}_i + \sum_{i\in T} x_i\right) + \sum_{i\in T}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i} h^{\left(\prod_{j\in\mathcal{S}} c_j\right)\sum_{i\in T}\tilde{R}_i + \sum_{i\in T}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right) z_{R_i}} \\ &\quad g^{\sum_{i\in\mathcal{S}\setminus T}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i} h^{\sum_{i\in\mathcal{S}\setminus T}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i} \end{aligned}$$

Both sides of the equality above can be interpreted as a Pedersen commitments. It is assumed, under the discrete logarithm assumption, that two Pedersen commitments cannot be equal unless their arguments are equal. This implies that for the LHS to be equal to the RHS above the exponents of g and h has to be equal on both sides. Consider the exponent of g this leads to,

$$\sum_{i\in T}\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\Big)\tilde{s}_j = \prod_{j\in\mathcal{S}} c_j\Big(\sum_{i\in T}\tilde{x}_i + \sum_{i\in T} x_i\Big) + \sum_{i\in T}\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\Big)s_i. \tag{4.4}$$

It remain to argue that equality in equation (4.4) cannot hold unless $\tilde{x}_i = x_i$, which would lead to an contradiction.

First consider the case that the set T only consist of one element, implying that there is only one malicious client. Without loss of generality assume that this is client c_k. Under this assumption equation (4.4) can be rewritten to,

$$\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq k}} c_j\Big)\tilde{s}_k = \Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq k}} c_j\Big)c_k\Big(\tilde{x}_k + x_k\Big) + \Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq k}} c_j\Big)s_k \implies \tilde{s}_k = c_k\Big(\tilde{x}_k + x_k\Big) + s_k$$

If it would be possible to choose $\tilde{s}_k = c_k\big(\tilde{x}_k + x_k\big) + s_k$, it would be possible to cheat in the set membership in Construction 1. Thereby if $|T| = 1$, it must hold that $\tilde{x}_k = x_k$

Instead assume that $|T| > 1$ and that c_k is a malicious client. Under the assumption that the provers cannot communicate and collaborate other clients set membership proof, $\{\Sigma_i\}_{\substack{i\in\mathcal{S}\\ i\neq k}}$, can be seen as random values for prover c_k during the construction of Σ_k. Therefore under this assumptions equation (4.4) can be rewritten to,

$$\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\Big)\tilde{s}_k + \overbrace{\sum_{\substack{i\in T\\ i\neq k}}\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\Big)\tilde{s}_j}^{\text{Random}} = \Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq k}} c_j\Big)c_k\Big(\tilde{x}_k + x_k\Big) + \Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq k}} c_j\Big)s_k$$
$$+ \overbrace{\prod_{j\in\mathcal{S}} c_j\Big(\sum_{\substack{i\in T\\ i\neq k}}\tilde{x}_i + \sum_{\substack{i\in T\\ i\neq k}} x_i\Big)}^{\text{Random}}$$
$$+ \overbrace{\sum_{\substack{i\in T\\ i\neq k}}\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\Big)s_i}^{\text{Random}}$$

If $\tilde{x}_k \neq x_k$, this would imply that the set membership proof can be broken by adding random values to D and z_x in Construction 1. This is a contradiction to the soundness of set membership proofs. This implies that $\tilde{x}_k = x_k$.

It has been proved that the aggregated set membership proof, presented in Construction 2, fulfils the soundness in Definition 6 under the assumptions that the aggregation is performed according to equation (4.3) by a trusted party, that provers cannot communicate and that the set membership proof in Construction 1 is sound. □

Untrusted Aggregating party

In the previous section it was assumed that the algorithm **Aggregate** was performed correctly by a trusted party. Under the assumption that the proving parties cannot communicate is was argued that the aggregated set membership proof in Construction 2 satisfies the correctness, soundness and zero-knowledge property in Definition 6.

In this section it will be investigated if the trusted party assumption is necessary or if some weaker assumption is sufficient alternatively if the aggregation can be checked. It will be assumed that proving parties cannot communicate between themselves and that proving parties cannot communicate with the aggregating party.

Note that since all input to the algorithm **Aggregate** is public, the aggregation can be done by anybody. Therefore an aggregation performed can always be checked if it is correct by running the algorithm **Aggregate** on one's own and then examine if the results are the same. To aggregate the set membership proofs is an expensive operations and hence arguments on an aggregations correctness without having to redo the aggregation would be useful.

Before looking into how the aggregation party might cheat and depart from the aggregation in equation (4.3), it will be cleared out what values the aggregation party can affect and what requirements that has hold. The aggregation party need to provide values for the proof parameters D, z_x, z_R such that $D = C^c h^{z_R} g^{z_x}$. The party performing the aggregation can not modify the values of the product of commitments, denoted C, nor the value of the product of the provers challenges, denoted c. If the aggregation can be performed such that the values D, z_x, z_R are not computed according to equation (4.3) and the verification verifies true, the aggregating party has deceived the verifier.

An important distinction is that in the original set membership construction the value of the challenge, c_i, used is not know when constructing C_i, V_i, D_i, a_i since it is a function of these values. In the aggregated set membership the challenges depends on the values publicly known for the aggregating party. This leads to that this knowledge can be taken advantage of to cheat. This results in that besides having a party independent of the aggregation, possibly the verifier, computing the product of the challenges, it must also be verified that $D \neq C^c$, where $C = \prod_{i \in \mathcal{S}} C_i$ and $c = \prod_{i \in \mathcal{S}} c_i$.

If it is not checked that $D \neq C^c$ it would be possible to choose the values D, z_x, z_R according to,

$$\begin{aligned} D =& \prod_{i \in \mathcal{S}} C_i^{\prod_{i \in \mathcal{S}} c_i} \\ z_x =& \phi(p) \\ z_R =& \phi(p), \end{aligned}$$

where p is the prime underlying the field $\mathbb{F}$. For the above choice of D, z_x, z_R the equation, $D \stackrel{?}{=} \left(\prod_{i \in \mathcal{S}}^{n} C_i \right)^{\prod_{i \in \mathcal{S}}^{n} c_i} h^{z_R} g^{z_x}$ holds trivially true, independent of whether the commitment C_i and the values z_{x_i} hides the same secret for all $i \in \mathcal{S}$. Henceforth it will be assume that it is checked that $D \neq C^c$.

This leads to that the aggregated values D, z_R, z_x must be such that $D \neq C^c$ and that left hand side equals the right hand side in the below equation system:

$$\begin{aligned} LHS =& D \\ RHS =& \Big(\prod_{i \in \mathcal{S}} C_i \Big)^{\prod_{i \in \mathcal{S}} c_i} h^{z_R} g^{z_x} = \Big(g^{\sum_{i \in \mathcal{S}} x_i} h^{\sum_{i \in \mathcal{S}} R_i} \Big)^{\prod_{i \in \mathcal{S}} c_i} h^{z_R} g^{z_x} \\ =& \Big(g^{(\prod_{i \in \mathcal{S}} c_i) \sum_{i \in \mathcal{S}} x_i)} h^{(\prod_{i \in \mathcal{S}} c_i) \sum_{i \in \mathcal{S}} R_i)} \Big) h^{z_R} g^{z_x}. \end{aligned}$$

In order to have equality between the left hand side and the right hand side, the terms derived from the commitments must be cancelled. The aggregating party does not know the secrets x_i or the random values R_i for any $i \in \mathcal{S}$ nor the sums $\sum_{i \in \mathcal{S}} x_i$ and $\sum_{i \in \mathcal{S}} R_i$, this follows from the assumption that the provers cannot communicate with each-other or with the party performing the aggregation.

Under the assumption discussed above, Theorem 4 states that aggregation must be performed according to the algorithm **Aggregate** for the algorithm **Verify** to verify true in Construction 2.

Theorem 4 (**Security of aggregation without trusted third party assumption**). *Let $\mathcal{A}$ be a PPT adversary. Assume $\mathcal{A}$ knows the set membership proofs Σ_i for $i \in \mathcal{S}$, where $\Sigma_i = (V_i, a_i, D_i, z_{x_i}, z_{\tau_i}, z_{R_i})$ calculated according to the algorithm **Prove** in Construction 2. In addition the adversary has also knowledge of the Pedersen commitments C_i and the challenges $c_i = Hash(D_i, a_i)$ for all $i \in \mathcal{S}$.*

Assume that the adversary $\mathcal{A}$ cannot communicate with the provers and that the provers cannot communicate with each other,. This leads to that the $\mathcal{A}$ has no knowledge of the secrets x_i or the randomness R_i in the Pedersen commitment, nor the sum of the secrets and random values.

Under these assumptions the adversary $\mathcal{A}$ has a negligible probability of choosing D, z_x, z_R such that: $D = C^c g^{z_x} h^{z_R}$, where $D \neq C^c$, $C = \prod_{i \in \mathcal{S}} C_i$ and $c = \prod_{i \in \mathcal{S}} c_i$ and $z_x \neq \sum_{i \in \mathcal{S}} \left(\prod_{j \in \mathcal{S}, j \neq i} c_j \right) z_{x_i}$.

Proof. Assume that the values D, z_x, z_R may be chosen by the adversary, such that $D \in \mathbb{G}$, $z_x, z_R \in \mathbb{F}$ and $z_x \neq \sum_{i \in \mathcal{S}} \left(\prod_{j \in \mathcal{S}, j \neq i} c_j \right) z_{x_i}$. Assumes without loss of generality that $D = g^a h^b$, where $a, b \in \mathbb{F}$. Given this the adversary $\mathcal{A}$ needs to choose the values a, b, z_x, z_R such that:

$$g^a h^b = C^c g^{z_x} h^{z_R},$$

the values of $C^c = \left(g^{\sum_{i \in S} x_i} h^{\sum_{i \in \mathcal{S}} R_i} \right)^{\prod_{j \in \mathcal{S}} c_j}$ can not be modified by the adversary since they are send directly from the provers to the verifier and used in the verification.

Expanding the right hand side of the equality it follows that,

$$g^a h^b = g^{c \sum_{i \in \mathcal{S}} x_i + z_x} h^{c \sum_{i \in \mathcal{S}} R_i + z_R}.$$

If $a \neq c \sum_{i \in \mathcal{S}} x_i + z_x$ and $b \neq c \sum_{i \in \mathcal{S}} R_i + z_R$, the above is contradiction to the assumption that it is not possible to construct two equal Pedersen Commitments hiding different secrets.

Thus the adversary must choose a, b, z_x, z_R such that $a = c \sum_{i \in \mathcal{S}} x_i + z_x$ and $b = c \sum_{i \in \mathcal{S}} R_i + z_R$. Considering the first equation it can be rewritten $a - z_x = c \sum_{i \in \mathcal{S}} x_i \bmod \Phi(p)$ and then the second equation $b - z_R = c \sum_{i \in \mathcal{S}} R_i \bmod \Phi(p)$. The right hand side of these equations is assumed to unknown, thereby the adversary has a negligible probability of choosing a, b, z_x, z_R such that the above is satisfied. Thereby it follows that $z_x = \sum_{i \in \mathcal{S}} \left(\prod_{j \in \mathcal{S}, j \neq i} c_j \right) z_{x_i}$, which contradicts the assumption and proves the theorem.

□

Note thate the Theorem would hold even if several untrusted parties would aggregate subsets if the proofs and then one verifier would verify all the aggregated proofs.

5
Implementation and Evaluation

In this chapter the aggregated signature based set membership proposed in the previous chapter will be implemented and evaluated in terms of runtime. The construction will be compared with itself for different settings and compared to the state of the art Bulletproofs.

5.1 Implementation

A prototype implementation of the aggregated signature based set membership proof has been implemented in Golang. The implementation is based on the code for the signature based set membership available on GitHub, [10]. Both for the construction where one aggregating party is considered and where multiples parties aggregates subsets of the proves proofs are implemented. The code for the aggregated signature based set membership proof is available at [**?**].

Implementation parameters

This section will definte the parameter setting used for the implementation. The number of provers will be fixed to 100, unless otherwise stated, and the verification will always be performed by one single party. The number of aggregating parties, $|\mathcal{K}|$, will vary between $|\mathcal{K}| = 1, 5, 10, 20$ and it will always be assumed that all aggregating parties aggregates the same amount of proofs. I.e if $|\mathcal{K}| = 10$ each aggregating party will aggregate $100/10 = 10$ proofs and if $|\mathcal{K}| = 5$ each aggregating party will aggregate $100/5 = 20$ proofs.

The set Φ will consist of 182 elements in the interval $[0, 1000]$.

The signature based set membership proofs are based on elliptic curve group, libsecp256k1 library available in Go-Ethereum. To obtain a 128-bit security the underlying field has to be of size $\sim$ 256-bits since the fastest known algorithm to solve elliptic curve discrete logarithm problem (ECDLP) requires $\mathcal{O}(\sqrt{n})$ steps. Therefore the finite field $\mathbb{F} = \mathbb{Z}_p$, where p is a 256-bit prime number.

The hardware used throughout the entire paper is: the computer used has a 1.6 GHz Dual-Core Intel Core $i5 - 5250U$ CPU, 8GB 1600 MHz DDR3 RAM and running macOS 10.15.

Table 5.1: Timing in seconds for algorithms **Aggregate** and **VerifyAggregated** in Construction 2. 100 provers are considered and one trusted party performing the aggregation of all set membership proofs.

$\lvert\mathcal{K}\rvert$	**Aggregate**	**Verify**
1	58.84[s]	8.09
2	19.10 [s]	8.15
5	2.22 [s]	8.16
10	0.60 [s]	8.33
Not aggregated	-	9.29

5.2 Trade-off between Aggregation and Verification

The aggregation of set membership proofs is computational heavy. Henceforth to reduce the computations for a single party the aggregation can be split between several aggregating parties. Such a generalised construction of the aggregated signature based set membership has been given in Appendix C in Construction 7. It has also been noted that if the set $\mathcal{K}$ in Construction 7 consists of one single element and $\mathcal{S}_k = \mathcal{S}$, then Construction 2 and 7 are the same. Given this henceforth this section will focus on Construction 7.

The reduced computation for any individual aggregating party obtained by having several parties aggregating subsets of the proofs will be compared to the increased verification time of having the verifier party verifying multiple aggregated proofs. Such a comparison of the runtime of the algorithm **Aggregate** and the algorithm **Verify** in Construction 7 is seen in Table 5.1. The runtime has been examined for number of aggregating parties varying between $1, 2, 5$ and 10, while the number of provers is held fix to 100. Each aggregating party has been assigned equally many proofs. Leading to that the each aggregating party is responsible to aggregate $100, 50, 20$ and 10 in the respective settings.

In Table 5.1 the runtime for the algorithm **Aggregate** is given per aggregating party and the runtime **Verify** if for performing the algorithm for all aggregated proofs.

5.3 Comparison to Bulletproofs

In this section the aggregated set membership proofs will be compared to the state of the art range proofs, Bulletproofs. The focus of comparison will be the runtime of verification of multiple proving parties executed by one verifier. Before presenting the results for the comparison some discussion about Bulletproofs will be made. This serves the purpose of clarifying what is being compared and explain the parameters used for the implementation of Bulletproofs.

Bulleproofs settings

The original paper about Bulletproofs [5] presents a method for aggregating Bulletproofs such that n parties, each having committed to a Pedersen commitment C_i, $i = 1, ..., n$, can generate a single Bulletproof verifying that each commitment hides a secret in an allowed range.

The proposed method for aggregation is an interactive construction. This construction will be interactive although the Fiat-Shamir heuristic is used to generate the challenges, since communication between the aggregating party and the provers is required during the construction of the aggregated Bulletproof. This is since it is required that all provers constructs their proofs using for the same challenges. If the provers where to use different challenges the verification would fail and the construction would not be complete.

Concluding, Bulletproof can be aggregated with the cost on an interactive construction. Since this paper aims to investigate non-interactive constructions, non aggregated Bulletproofs will be considered for the below comparison. Therefore the verifier will have to verify all Bulletproofs separately, i.e once for each prover.

The computational complexity for verification of Bulletproofs depends on the maximal upper bound of the range. This motivates to see how the runtime is affected by considering different upper bounds. The maximal upper bound of Bulletproofs is 2^n, for some n that is a power of 2. Two different values of n will be considered for evaluating the runtime, namely $n = 8$ and $n = 32$.

Bulletproofs can also be modified to allow arbitrary range, $[a, b]$, with a similar approach as presented for the signature based range proofs and illustrated in Figure 2.1. Further the range for the Bulletproofs will be fixed to $[18, 200]$.

Runtime Comparison

The runtime results presented is a comparison between how long the verification time is depending on which construction provers uses to prove that their secret is in an allowed range or set. Evidently the construction used to provide the proofs determines the construction used for the verification. Four different constructions will be considered, Bulletproofs with an upper bound equal to 2^8 and 2^{32}, and aggregated and not aggregated signature based set membership proofs.

Table 5.2 shows the runtime for the verification of 100 provers constructing their proof using the four considered constructions. The verification algorithm used is the verification corresponding to the algorithm used to provide the proof.

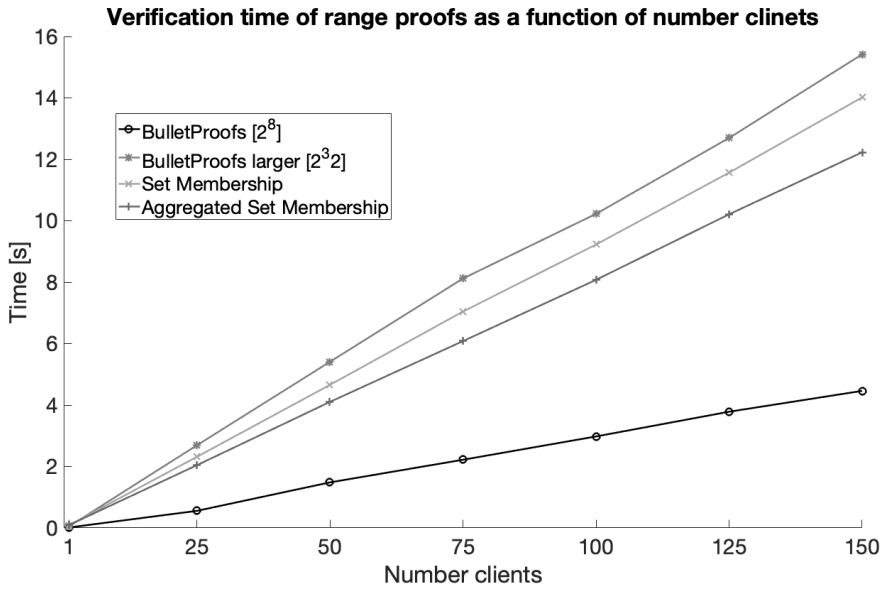
The Bulletproofs allowing for ranges of upper bound lower than 2^8 is considerable faster than all other constructions, however it is highly limited in the applications since the upper bound is fairly low. The runtime for Bulletproof of an upper bound equal to 2^{32} is longer than for both aggregated and non aggregated set membership proofs.

In Figure 5.1 the runtime for verification is given as a function of the number of provers for the four considered constructions. The runtime has been measured considering $1, 25, 50, 75, 100, 125$ respective 150 provers. From the figure it is seen that there is a almost a linear relationship between the number of provers and the runtime for verification. It is also seen that the runtime comparison between the

Table 5.2: The table shows the runtime for verification of 100 proofs, using four different constructions to provide the proofs and a verification algorithm compatible with the proof construction.

	Time
Bulletproofs $n = 8$	2.98[s]
Bulletproofs $n = 32$	10.22 [s]
Set Membership	9.29 [s]
Aggregated Set Membership	8.09 [s]

Figure 5.1: Runtime for the verification of set membership proofs and range proofs depending on the number of provers. The runtime is compared between aggregated and not aggregated set membership proofs and two different instances of Bulletproofs, the maximal upper bound of the range is equal to 2^8 respective 2^{32}.



different constructions seen in Table 5.2 appears to hold independent of the numbers of provers.

6
Application in VAHSS

In this chapter an application of the aggregated set membership proof will be presented and evaluated. As discussed in the introduction VAHSS constructions is one example of constructions where multiple data providers, henceforth denoted clients, participates. In this chapter a construction of VAHSS, where the verification is extended to also include the clients, will be presented. The verification of clients will be such that each clients has to prove that the shared secret is in an predetermined allowed set or range. After having modified the VAHSS construction such that clients are verified, different methods for verifying clients will be compared. This will mainly focus on comparing the previously derived aggregated set membership proof presented in Construction 2 to the state of the art Bulletproofs.

6.1 Construction of VAHSS Using Homomorphic Hash Functions

The construction of VAHSS that will be considered in this paper was originally presented in [16], and further implemented and benchmarked in [15]. In [16] three different constructions of VAHSS, was introduced in this paper the construction that makes use of homomorphic hash functions for verification of servers will be considered.

It will be assumed that n clients and m servers participated in the VHASS construction. To simplify notation the two sets $\mathcal{N} = \{1, ..., n\}$ and $\mathcal{M} = \{1, ..., m\}$ are introduced. The clients are denoted c_i for $i \in \mathcal{N}$ and their respective data x_i. The servers in the construction will be refereed to $s_j,\ j \in \mathcal{M}$. Each client c_i, splits their secret, x_i, into m shares, denoted $\{x_{ij}\}_{j\in\mathcal{M}}$. The clients then sends one share to each server. The servers receives shares from all n clients and computes the partial sum $y_j = \sum_{i\in\mathcal{N}} x_{ij}$ and publishes the result. The final sum can then be computed by any party by summing the public partial sums, this gives $y = \sum_{j\in\mathcal{M}} y_j = \sum_{j\in\mathcal{M}} \sum_{i\in\mathcal{N}} x_{ij} = \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{M}} x_{ij} = \sum_{i\in\mathcal{N}} x_i$.

The proof σ of the servers computations is obtained accordingly. Each client c_i, publishes a checksum $\tau_i = g^{x_i+R_i}$, where x_i is the secret hidden by the distributed shares and $R_i \in_R \mathbb{F}$ is such that $R_n = \phi(p)\lceil\frac{\sum_{i=1}^{n-1} R_i}{\phi(p)}\rceil - \sum_{i=1}^{n-1} R_i$. Each server s_j, computes a partial proof $\sigma_j = g^{y_j}$. Finally the verification is done by checking if $\prod_{j\in\mathcal{M}} \sigma_j = \prod_{i\in\mathcal{N}} \tau_i \wedge \prod_{i\in\mathcal{N}} \tau_i = g^y$. If this holds then the servers computations are proved to be done correctly. For a precise implementation and proof of correctness, security and verification the reader is refereed to the original paper, [16].

6.2 Additive homomorphic secret sharing with verification of both clients and servers

The VAHSS construction discussed in section 6.1 assumes honest clients and verifiers the servers. The aim of this section is to extended the VAHSS construction to verify both clients and servers, without the clients needing to reveal their secrets. This will achieved by first deriving a client and server VAHSS, where the clients are verified using range proofs or set membership proofs based on a Pedersen commitment. Then it will also be discussed how such an construction would need to be modified to use instead aggregates set membership proofs.

Remark that if a range proof or set membership proof is included in the VAHSS construction then potentially malicious clients can only have a limited influence on the computed sum. Range proofs and set membership proofs forces malicious input to still be part of the range or set. This leads to that the impact on the sum that a malicious client can have is bounded by the size of the range or the elements in the set.

Client and Server VAHSS

In this section it will be investigated how to combine the VHASS construction with a range proofs or set membership proofs. Only range proofs and set membership proofs emanate from a Pedersen commitment hiding a secret and generates a zero knowledge proof that this secret belongs to an pre-specified interval or set will be considered.

Note that to ensure honest clients it is not sufficient to construct and perform a set membership proof and VAHSS scheme separately. In such a protocol the verifier cannot be sure that the secret proven to be in the allowed set is the same as the secret hidden by the shares. The same holds considering range proofs.

Using the Pedersen commitment to link the VAHSS construction with a range proof or set membership proof, would result in a general construction of a client and server VAHSS compatible with any range proof or set membership proof enmeating from a Pedersen commitment. To obtain such a general construction of a client and server VAHSS the Pedersen commitment will be investigated if it can be implemented in the VAHSS construction.

More precisely a link between the shares generated in the algorithm **ShareSecret** in the VAHSS construction and the secret hidden in an Pedersen commitment is desired. Proving that the sum of the shares is equal to the secret in an Pedersen commitment and then providing a zero knowledge range proof or set membership proof for the commitment, would convince the verifier that the shares represents a secrets that is in the allowed range.

Publishing a Pedersen commitment of the secret itself does not provide any guarantee that it is the same secret that is hidden by the shares. It can also be seen that committing to the shares in Pedersen commitments does not ensure the verifier that the secret hidden in the shares are in the allowed set or range. This is since the individual shares themselves does not reveal information about the secret they are hiding. This leads to that there is not guarantee that proving a share belongs to

the allowed range (or set) implies that the secret does and the other way assuming that the secret to a range (or set) does not imply that the shares does.

Recall that the clients in addition to the shares also publishes the checksum τ_i for the secret x_i, further recall that the definition of the checksum is $\tau_i = g^{x_i+R_i}$, where R_i is chosen uniformly at random from the field $\mathbb{F}$ such that $R_n = \phi(p)\lceil\frac{\sum_{i=1}^{n-1} R_i}{\phi(p)}\rceil - \sum_{i=1}^{n-1} R_i$.

The checksum τ_i can be seen as a generalisation of a Pedersen commitment. Therefore the checksum may be used as a Pedersen commitment in the construction of a range proof or set membership proof. Formallt the checksum is a Pedersen commitment where $g = h$. However if $g = h$ the computationally binding property of a Pedersen commitment would not hold since $log_g(h) = log_g(g) = 1$ which leads to that the left hand side in equation (2.1) is equal to 1. Therefore to construct two commits $\mathbb{E}(x, R)$ and $\mathbb{E}(x', R')$ such that $\mathbb{E}(x, R) = \mathbb{E}(x', R')$ but $x \neq x'$ it is sufficient to solve solve for x' in,

$$R' - R = x - x' \, mod \, p.$$

In other words it is straightforward to create a false commitment hence also a false range proof and set membership proof.

Instead investigated if the checksum τ_i can be modified into a Pedersen commitment where $g \neq h$. This leads to that instead of the previously computed checksum τ_i, the clients compute and output $\pi_i = g^{x_i} h^{R_i}$, where x_i, R_i, g, h are as above.

Given the commitment π_i the clients can use it to construct a range proof or set membership proof of the secret in the commitment. It remains to argue that this method ensures the verifier that the secret hidden by the shares is the same secret as in the commitment π_i for $i \in \mathcal{N}$ that are used to prove honest clients.

Assume that client c_k commits to the value $\hat{x}_k$ in the Pedersen commitment π_k and constructs shares, x_{kj} such that $x_k = \sum_{j \in \mathcal{M}} x_{kj} \neq \hat{x}_k$. This leads to that c_k generates a range proof that the secret hidden in the commitment belongs to the interval $[a, b]$ but the secret hidden by the shares does not necessarily belong to the range or set. Then the verification of the servers will not hold, since $\prod_{i=1}^{m} \pi_i \neq g^y$. this means that the verification will return false and the protocol will not succeed, even if all range proofs verifies true. Thus any cheating party will be detected, it will not be possible do determine which party that cheated and more precisely not even if the cheating party was a client or a server.

In Construction 3 the extended VAHSS, such that the construction ensures honest clients by including a range proof or set membership proof is described in detail. In order to clarify the modifications made to include a verification of the clients, the differences to the VAHSS construction presented in [16] are pointed out. The algorithms **ShareSecret** and **Verify** has been modified, and the algorithms **RangeProof** and **GenerateCommitment** have been added. More precisely in the algorithm **ShareSecret** does not output the checksum τ_i, instead the Pedersen commitment π_i is computed in the algorithm **GenerateCommitment**. The algorithm **GenerateCommitment** can be included in either **ShareSecret** or **RangeProof** instead of being viewed as a separate algorithm. In the implementation discussed later the commitment is generated while constructing the range proof and not explicitly. The algorithm **RangeProof** constructs a range proof (or set membership

proof) denoted Σ_i given the commitment π_i. It is not specified which range proof or set membership proof that is used to verify clients. Which range proof or set membership proof that is used does not affect the rest of Construction 3, as long as it emanate from a Pedersen commitment, hence it is left unspecified. In addition to the steps in the algorithm **Verify** for the VAHSS presented in [16], the algorithm **Verify** verifies the correctness of the proofs Σ_i and an additional `AND` operator to compute the total verification.

The algorithms **GenereateCommitment** and **RangeProof** are executed by the clients and the other algorithms are executed by the same party as in the VAHSS construction in [16].

Construction 3 : Client and Server Verifiable additive homomorphic secret sharing

Goal: Construct and share the sum $\sum_{i=1}^{n} x_i$, where x_i is a secret value known by client c_i, where $i \in \mathcal{N}$ without any client needing to revealing their individual secret. All parties are verified to be honest in the construction.

- **ShareSecret** $(1^\lambda, i, x_i) \mapsto \{x_{ij}\}_{j\in\mathcal{M}}$
  Pick uniformly at random $\{a_i\}_{i\in\{1,..,t\}} \in_R \mathbb{F}$ to be the coefficients to a t-degree polynomial p_i on the form $p_i(X) = x_i + a_1X + ... + a_tX^t$. Define the shares as $x_{ij} = \lambda_{i,j}p_i(\theta_{ij})$ for $j \in \mathcal{M}$, the parameters θ_{ij} and Lagrange coefficients λ_{ij} is chosen such that equation **??** is satisfied. Output $\{x_{ij}\}_{j\in\mathcal{M}}$.
- **GenereteCommitment**$(1^\lambda, i, x_i) \mapsto \pi_i$
  Let $\mathbb{E} : x, y \to g^x h^y$ be a Pedersen commitment . Let $R_i \in \mathbb{F}$ be the output of a PRF such that $R_n \in \mathbb{F}$ satisfies $R_n = \phi(N)\lceil \frac{\sum_{i=1}^{n-1} R_i}{\phi(N)} \rceil - \sum_{i=1}^{n-1} R_i$. Compute and output $\pi_i = \mathbb{E}(x_i, R_i)$.
- **RangeProof** $(pp, x_i, \pi_i) \mapsto \Sigma_i$
  Construct a range proof or set membership proof, denoted Σ_i, for the Pedersen commitment π_i of the secret x_i, on the range $[0, B]$ or a set Φ. All required public parameters, pp, needed to construction the proof Σ_i is assumed to be pre-shared and known by all parties.
- **PartialEval** $(j, \{x_{ij}\}_{i\in\mathcal{N}}) \to y_j$
  Compute and output $y_j = \sum_{i=1}^{n} x_{ij}$.
- **PartialProof** $(j, \{x_{ij}\}_{i\in\mathcal{N}}) \to \sigma_j$
  Compute and output $\sigma_j = \prod_{i=1}^{n} g^{x_{ij}} = g^{\sum_{i=1}^{n} x_{ij}} = g^{y_j} = H(y_j)$.
- **FinalEval** $(\{y_j\}_{j\in\mathcal{M}}) \to y$
  Compute and output $y = \sum_{j=1}^{m} y_j$.
- **FinalProof** $(\{\sigma_j\}_{j\in\mathcal{M}}) \to \sigma$
  Compute and output $\sigma = \prod_{j=1}^{m} \sigma_j = \prod_{j=1}^{m} g^{y_j} = g^{\sum_{j=1}^{m} y_j} = g^y = H(y)$.
- **Verify** $(\{\pi_i\}_{i\in\mathcal{N}}, x, y, \{\Sigma_i\}_{i\in\mathcal{N}}) \to \{0, 1\}$
  Compute and output $\sigma = \prod_{i=1}^{n} \pi_i \wedge \prod_{i=1}^{n} \pi_i = H(y) \wedge \{\textbf{VerifyClients}(\Sigma_i)\}_{i\in\mathcal{N}}$. Where **VerifyClients** is the verification algorithm associated with the algorithm used by the clients to construct the proofs $\{\Sigma_i\}_{i\in\mathcal{N}}$.

Theorem 5. *The client and server VAHSS presented in Construction 3 satisfies the correctness, security and verifiability requirements described in section 6.1, by*

replacing τ_i with π_i. Additionally it also satisfies the following extension of the verification requirement:

Let $\mathcal{A}$ denote any PPT adversary and T denote the set of corrupted clients. The extended verifiability property requires that any $\mathcal{A}$ who can modify the Pedersen commitments π_i to any $\pi_i^{'}$ $\forall i \in T$ has a negligible probability at choosing a commitment $\pi_i^{'}$ such that Verify$(\{\pi_i^{'}\}_{i\in\mathcal{N}}, x, y, \{\Sigma_i\}_{i\in\mathcal{N}}) = 1$.

Proof. To argue that that the correctness, security and verifiability properties for the VAHSS still holds after replacing τ_i with π_i for $i = 1, ..., n$, it is noted that,

$$\prod_{i=1}^{n} \pi_i = \prod_{i=1}^{n} g^{x_i} h^{R_i} = g^{\sum_{i=1}^{n} x_i} h^{\sum_{i=1}^{n} R_i} = g^{\sum_{i=1}^{n} h^{\phi(N)\left\lceil \frac{\sum_{i=1}^{n-1} R_i}{\phi(N)} \right\rceil}} = g^y$$

Hence it follows that: $\prod_{i=1}^{n} \tau_i = \prod_{i=1}^{n} \pi_i$.

Further the Pedersen commitment is perfectly hiding and computationally binding and hence it follows that the requirements is still fulfilled.

It remains to prove that Construction 3 also fulfils the extended verification requirement. This follows from the soundness of the range proofs or set membership proof used in the construction and by the argument above that the secret hidden in the commitment π_i must be the same as the secret obtain by combining the the shares $\{x_{ij}\}_{j\in\mathcal{M}}$ for all $i = 1, ..., n$.

□

Combining with aggregated set membership proof

When using the aggregated set membership to verify the clients in a VHASS construction Theorem 4 do not hold if the aggregation is performed by one untrusted server, since the assumptions are not fulfilled. The sum of secrets can be computed by any party after all servers has performed the algorithm **partialEval** and it is known that $h^{\sum_{i\in\mathcal{N}} R_i} = 1 \bmod \phi(p)$. Thereby although the individual secrets are unknown, it is possible to cheat by using the fact that $y = \sum_{i\in\mathcal{N}} x_i, h^{\sum_{i\in\mathcal{N}} R_i}$ are known. To illustrate how this information can be use to cheat consider, $D = C^{k+c}, z_R = k\phi(N)$ and $z_x = ky$, where $y = \sum_{i=1}^{n} x_i$ and $k \in_R \mathbb{F}$, it follows that,

$$\begin{aligned}
LHS =& D = C^{k+c} = \left(g^{k\sum_{i=1}^{n} x_i} h^{k\sum_{i=1}^{n} R_i)}\right)\left(g^{(\prod_{i=1}^{n} c_i)\sum_{i=1}^{n} x_i)} h^{(\prod_{i=1}^{n} c_i)\sum_{i=1}^{n} R_i)}\right) \\
LHS =& C^c h^{z_R} g^{z_x} = C^c h^{k\phi(N)} g^{ky} = \left(g^{(\prod_{i=1}^{n} c_i)\sum_{i=1}^{n} x_i)} h^{(\prod_{i=1}^{n} c_i)\sum_{i=1}^{n} R_i)}\right) h^{k\phi(N)} g^{ky} \\
=& \left(g^{(\prod_{i=1}^{n} c_i)\sum_{i=1}^{n} x_i)} h^{(\prod_{i=1}^{n} c_i)\sum_{i=1}^{n} R_i)}\right) g^{ky} h^{k\phi(N)\lceil \frac{\sum_{i=1}^{n-1} R_i}{\phi(N)} \rceil)} \\
&\implies LHS = RHS.
\end{aligned}$$

Thus it follows that Theorem 4 does not hold when one untrusted part aggregates all clients set membership proofs in a client and server VHASS.

If instead of one party that aggregates all set membership proofs, if multiple parties aggregates subsets of the proofs, Theorem 4 would hold. This follows from the fact that the sum of the secrets and random values would be unknown to the

Construction 4 : Client and Server Verifiable additive homomorphic secret sharing

Goal: Construct and share the sum $\sum_{i=1}^{n} x_i$, where x_i is a secret value known by client c_i, where $i \in \mathcal{N}$ without any client needing to revealing their individual secret. All parties are verified to be honest in the construction.

- **ShareSecret** $(1^\lambda, i, x_i) \mapsto \{x_{ij}\}_{j \in \mathcal{M}}$
  Pick uniformly at random $\{a_i\}_{i \in \{1,..,t\}} \in_R \mathbb{F}$ to be the coefficients to a t-degree polynomial p_i on the form $p_i(X) = x_i + a_1 X + ... + a_t X^t$. Define the shares as $x_{ij} = \lambda_{i,j} p_i(\theta_{ij})$ for $j \in \mathcal{M}$, the parameters θ_{ij} and Lagrange coefficients λ_{ij} is chosen such that equation **??** is satisfied. Output $\{x_{ij}\}_{j \in \mathcal{M}}$.
- **GenereteCommitment**$(1^\lambda, i, x_i) \mapsto \pi_i$
  Let $\mathbb{E} : x, y \to g^x h^y$ be a Pedersen commitment . Let $R_i \in \mathbb{F}$ be the output of a PRF such that $R_n \in \mathbb{F}$ satisfies $R_n = \phi(N)\lceil \frac{\sum_{i=1}^{n-1} R_i}{\phi(N)} \rceil - \sum_{i=1}^{n-1} R_i$. Compute and output $\pi_i = \mathbb{E}(x_i, R_i)$.
- **RangeProof** $(pp, x_i, \pi_i) \mapsto \Sigma_i$
  Construct a range proof, denoted Σ_i, for the commitment π_i to the secret x_i, on the set Φ using the algorithm **Prove** in Construction 2. All required pubic parameters a pp is assumed to be pre-shared and known by all parties.
- **PartialEval** $(j, \{x_{ij}\}_{i \in \mathcal{N}}) \to y_j$
  Compute and output $y_j = \sum_{i=1}^{n} x_{ij}$.
- **PartialProof** $(j, \{x_{ij}\}_{i \in \mathcal{N}}) \to \sigma_j$
  Compute and output $\sigma_j = \prod_{i=1}^{n} g^{x_{ij}} = g^{\sum_{i=1}^{n} x_{ij}} = g^{y_j} = H(y_j)$.
- **PartialAggregate** $(pp, k, \mathcal{S}_k, \{\Sigma_i\}_{i \in \mathcal{S}}) \to \Sigma_{a_k}$
  Given a subset of proofs $\{\Sigma_i\}_{i \in \mathcal{S}_k}$ where $\mathcal{S}_k \subseteq \{1, ..., n\}$. Aggregate the set of proof according to the algorithm **Aggregate** in Construction 2. Publish the aggregates proof Σ_{a_k}.
- **FinalEval** $(\{y_j\}_{j \in \mathcal{M}}) \to y$
  Compute and output $y = \sum_{j=1}^{m} y_j$.
- **FinalProof** $(\{\sigma_j\}_{j \in \mathcal{M}}) \to \sigma$
  Compute and output $\sigma = \prod_{j=1}^{m} \sigma_j = \prod_{j=1}^{m} g^{y_j} = g^{\sum_{j=1}^{m} y_j} = g^y = H(y)$.
- **Verify** $(\{\pi_i\}_{i \in \mathcal{N}}, x, y, \{\Sigma_{a_k}\}_{j \in \mathcal{K}}) \to \{0, 1\}$
  Compute and output $\sigma = \prod_{i=1}^{n} \pi_i \wedge \prod_{i=1}^{n} \pi_i = H(y) \wedge \{\textbf{VerifyClients}(\Sigma_{a_k})\}_{k \in \mathcal{K}}$. Where **VerifyClients** is the verification algorithm in Construction 2.

aggregating party. The construction of the aggregated client and server VAHSS construction for untrusted aggregating parties is presented in Construction 4.

For the VAHSS construction this can be implemented by letting the server, which computes the partial sums, aggregate different subset of the clients set membership proofs. For example in the case of 100 clients and 5 servers, each server aggregates 20 proofs. For example this could be implemented by $\mathcal{K} = \mathcal{M} = \{1, 2, 3, 4, 5\}$ and $\mathcal{S}_1 = \{1, ..., 20\}, \mathcal{S}_2 = \{21, ..., 40\}, ..., \mathcal{S}_5 = \{81, ..., 100\}$. In such a construction the number of aggregated set membership proofs for the verifier to verify depends on the number of servers in the construction. Using this principle the set $\mathcal{K}$ in the construction represents the set of parties performing the aggregation, would be equal to $\mathcal{M}$, the set or servers. And if the set set of aggregating parties $\mathcal{K}$ consist of one element and $\mathcal{S}_k = \mathcal{N}$ the partial aggregation would correspond to a full aggregation, and Construction 4 would describe the aggregated client and severs VAHSS construction for a single trusted aggregating party.

To conclude, if the aggregation is not assumed to be performed by a trusted party then the characteristics of the VAHSS construction all set membership proof cannot be aggregated by a single party. Thus then the aggregation must the split between atleast two parties.

6.3 Implementation

To practically investigate the proposed clients and server VAHSS and compare the runtime for different methods for verifying clients, an implementation of Constructions 3 and 4 is desired. To provide an implementation of Construction 3 and 4 the algorithms constructing a VAHSS will have to be combined with the algorithms to prove an verify clients. Different methods for verifying the clients will be considered. For the evaluation and comparison in this paper Bulletproofs, aggregated and not aggregated signature based set membership proofs will be considered.

Implementations of Bulletproofs, aggregated and not aggregated signature based set membership proofs written in Golang (Go) are publically available, the code can be found on Github at [10] respective [**?**]. Implementations of the VAHSS construction is available written in both python and C++, the python code is available at [14] and the C++ code at [13].

To implement Construction 3 and 4 the VAHSS algorithms need to be callable from the same scripts as the algorithms to verify clients, i.e Bulletproofs, signature based set membership proofs and aggregated signature based set membership proofs algorithms. To solve the problem of having the implementations written in different programming languages the VAHSS algorithms has been translated to Golang and the implementation is available at [**?**].

To provide an implementation of Construction 3 and 4 besides translating the VAHSS code to Golang the implementations has also been slightly modified in order to be merged. For example the VAHSS construction has as discussed above been adjusted such that it considers a Pedersen commitment π_i instead of the checksum τ_i to verify servers.

The implementations for Bulletproof, signature based set membership proofs and aggregated signature based set membership proof has also been adjusted to

be compatible with the VAHSS, the modified implementations is available at [**?**]. These adjustments are merely to merge the constructions and does not change the semantics of the constructions.

Implementation parameters

The finite field $\mathbb{F}$ is generated by a prime of size 256-bit as before.

The number of servers is set to 5, $|\mathcal{M}| = 5$, and the number of clients to 100, $|\mathcal{N}| = 100$. The set of aggregation parties $\mathcal{K}$ will be assumed to consist of a single party, $|\mathcal{K}| = 1$.

The range is set to $[18, 200]$ for the implementation of Bulletproofs, and the upper bound of the Bulletproof is then sufficient to be $2^8 = 256$.. This leads to that the parameter n in the Bulletproof construction determining the complexity is put to $n = 8$ To make the set membership implementation comparable the size of the set is equal to the length of the range , i.e $|\Phi| = 200 - 18 = 182$, for both the aggregated and not aggregated construction.

A final remark about the implementation is that its purpose is to test the concept on the above proposed construction and provide runtime evaluations, the code has not been tested enough to be used as secure implementation.

6.4 Prototype analysis

The implementation of Constructions 3 and 4 can be used to provide runtime results to compare different constructions for verifying clients. Such results will be presented in this section using different methods for verifying clients.

The construction of a client and server VHASS is a almost direct merge of a server VHASS and range proof or set membership proof. Consequently it can be expected that the runtime for the different algorithms in 3 and 4 will be almost the same as the corresponding algorithms for the separate constructions of VAHSS, range proofs and set membership proofs.

For example the algorithm **ShareSecret** almost the same in Construction 3 and 4 as in the VAHSS construction presented in [16]. Thus the runtime for the algorithm is likely to comparable with the results given in [15].

Smaller adjustment has been made to the algorithms due to combining and hence the runtime for all algorithms in the the client and server VAHSS in evaluated.

The runtime for the algorithms in Construction 3 and 4 using Bulletproofs, signature based set membership proofs respective aggregated signature-based set membership proofs to verify clients is seen in Table 6.1. Note that the runtime for the algorithm **VerifyProof** is given per client.

A remark is that the algorithms **PartialProof**, **FinalProof** and **VerifyServers** differs noteworthy in runtime for different construction to verify clients honesty. These algorithms as described in Construction 3 and 4 are seemingly independent of which method that is used. The reason for difference in runtime comes from when implementing different libraries and groups are used which affect the performance. Optimisation of the libraries has not been made and it the difference can be reduces due to optimisations has not been investigated.

In Construction 3 and 4 there is one verification algorithm called **Verify**. To separately measure the runtime for verifying the servers and clients the algorithm **Verify** is split into two steps. The first, **VerifyServers**, performs the verification of the servers computations and the second, **VerifyClients**, verifies all client by evaluating the range proof or set membership proof. This gives that the algorithm **Verify** has been rewritten as,

- **Verify** $(pp, \{\pi_i\}_{i\in\mathcal{N}}, y, \{\Sigma_i\}_{i\in\mathcal{N}}) \rightarrow \{0,1\}$
  Verify the clients and servers according to,
  - **VerifyServers** $(\{\pi_i\}_{i\in\mathcal{N}}, y) \rightarrow \{0,1\}$)
    Compute and output $\sigma = \prod_{i=1}^{n} \pi_i \wedge \prod_{i=1}^{n} \pi_i = H(y)$.
  - **VerifyClients** $(\{\pi_i\}_{i\in\mathcal{N}}\{\Sigma_i\}_{i\in\mathcal{N}}) \rightarrow \{0,1\}$
    For each proof Σ_i, verify that it is correct.This imples running **VerifyProof**(π_i, Σ_i) for all $i \in \mathcal{N}$. Where **VerifyProof** is the verification algorithm associated with the algorithm used to construct the proof, Σ_i. If the proofs has been aggregated then the verification if performed for each aggregated proof instead of al clients proofs. If all proofs are correct return 1, else 0.

  Return **VerifyServers** $\wedge$ **VerifyCleints**

Considering this paraphrase, it is clear what the runtime **VerifyServers** and **VerifyProof** in Table 6.1 measures. Note that **VerifyServers** is the runtime to verify all servers while **VerifyProof** is the runtime for verifying one client.

Uniformly for all construction for verifying the clients the runtime for **VerifyClients** is longer then **VerifyServers**. This is despite the fact that **VerifyClients** corresponds to the runtime of verifying one clients while **VerifyServers** is the runtime to verify all severs. This highlights how expensive the verification of clients are and motivates to aggregate the verification of clients.

In Table 6.1 the runtime is almost consequently fastest when using Bulletproofs to verify the clients. The considered allowed range is $[18, 200]$, hence it was sufficient to use $n = 8$ for the upper bond for the Bulletproofs implementation. In section 5.3 is has been seen that the runtime increases significantly if the upper bound of the range is increased. This implies that considering other ranges the aggregated signature based set membership would be faster for verification of all clients, as seen in Figure 5.1.

Above it was remarked the runtime difference for the algorithms **PartialProof**, **FinalProof** and **VerifyServers** between using Bulletproof and signature based set membership proofs, this runtime will not be affected by increasing the maximum upper bound. I.e Bulletproofs will consequently independent of implementation parameters be faster for these algorithms. Not considering possible optimisations of the used libraries.

The preference of which method to use for verification of clients is application specific. Determined by which party has the highest computational power and which praty the lowest.

Table 6.1: Timing in seconds for server and client verifiable-AHSS. Verification of clients is done using three different constructions namely by implementing Bulletproofs, signature based range proofs and set membership proofs

	Time		
	Bulletproofs	Set membership	Aggregated Set membership
GenerateShares	95/x [μs]	98[μs]	98 [μs]
GenerateRangeProof	53/x [ms]	66 [ms]	66 [ms]
PartialEval	78/x [μs]	71[μs]	71 [μs]
PartialProof	273/x[μs]	5255 [μs]	5255 [μs]
Aggregate	-	-	7947 [μs]
FinalEval	689/x [ns]	699 [ns]	699 [ns]
FinalProof	50/x [μs]	115 [μs]	115 [μs]
VerifyProof (per client)	2979/x[ms]	9288 [ms]	9288 [ms]
VerifyServers	1672/x [μs]	7947 [ms]	7947 [μs]

7
Discussion

In this chapter the results obtained will be discussed as well as limitation and decision made during the work. Finally a short conclusion will be given.

In Table 5.1 it is noted that the aggregation itself is a very computationally demanding procedure. The runtime required to aggregate 100 proofs is almost one minute, while the reduction in runtime using the aggregated proof is just above one second.

Future work

A full aggregation, such that the verification is completely independent of the number of proofs has not been found for the signature based set membership proofs. The bilinear paring is the most time consuming operation in the verification of a set membership proof hence an aggregation of this part of the proofs would have a significant impact of the runtime.

Another question that arise during the work is if an non-interactive aggregation of Bulletproofs can be obtained. To the knowledge of the author no such construction exist. Considering the efficiency of Bulletproofs it would be interesting to compare an aggregated implementation of Bulletproofs to the aggregated signature based set membership proofs.

Conclusion

To conclude it has been seen how to construct aggregated set membership proofs and what requirements such an construction should satisfy. Further an partly aggregated construction of a signature based set membership proof was derived. If it is possible to construct an fully aggregated construction of signature based set membership proofs was unanswered and remains an open question.

It was also seen that aggregated set membership proof can be used to verify clients in a VAHSS construction.

Bibliography

[1] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, page 62–73, New York, NY, USA, 1993. Association for Computing Machinery.

[2] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 326–349, New York, NY, USA, 2012. Association for Computing Machinery.

[3] D. Boneh and X. Boyen. Short signatures without random oracles. In C. Cachin and J. L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 56–73, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[4] F. Boudot. Efficient proofs that a committed number lies in an interval. In B. Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, pages 431–444, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[5] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.

[6] J. Camenisch, R. Chaabouni, and a. shelat. Efficient protocols for set membership and range proofs. In J. Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, pages 234–252, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[7] R. Chaabouni, H. Lipmaa, and A. Shelat. Additive combinatorics and discrete logarithm based range protocols. In R. Steinfeld and P. Hawkes, editors, *Information Security and Privacy*, pages 336–351, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[8] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[9] E. Morais, T. Koens, C. van Wijk, and A. Koren. A survey on zero knowledge range proofs and applications. *SN Applied Sciences*, 1(946), 2019.

[10] E. Morais, T. Koens, C. van Wijk, A. Koren, P. Rudgers, and C. Ramaekers. Zero knowledge range proof implementation. `https://github.com/ing-bank/zkrp`, 2020.

[11] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[12] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.

[13] G. Tsaloli and G. Banegas. Additative vhsss implemented in c++. `https://github.com/tsaloligeorgia/AddVHSS`, 2020.

[14] G. Tsaloli and G. Banegas. Additative vhsss implemented in python. `https://github.com/gbanegas/VHSSS_temp`, 2020.

[15] G. Tsaloli, G. Banegas, and A. Mitrokotsa. Practical and provably secure distributed aggregation: Verifiable additive homomorphic secret sharing. *Cryptography*, 4(3), 2020.

[16] G. Tsaloli and A. Mitrokotsa. Sum it up: Verifiable additive homomorphic secret sharing. In J. H. Seo, editor, *Information Security and Cryptology – ICISC 2019*, pages 115–132, Cham, 2020. Springer International Publishing.

[17] H. Yao, C. Wang, B. Hai, and S. Zhu. Homomorphic hash and blockchain based authentication key exchange protocol for strangers. In *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, pages 243–248, 2018.

A

Non-interactive Bulletproofs

Construction 5 : Bulletproof

Goal: Given a Pedersen commitment $C = g^x h^R$ and a number n, prove that the secret x in the commitment belongs to the range $[0, 2^n)$ without revealing anything else about x.

- **Prove** $(g, h, \mathbf{g}, \mathbf{h}, P, n, x, R, u) \rightarrow \ proof_{RP}$
  Let $\boldsymbol{x}$ denote the binary representation of the secret x in the commitment C and $\bar{\boldsymbol{x}}$ the component-wise complement such that $\boldsymbol{x} \circ \bar{\boldsymbol{x}} = 0$. Construct the commitment $A = h^{\alpha}\boldsymbol{g}^{\boldsymbol{x}}\boldsymbol{h}^{\bar{\boldsymbol{x}}}$, where $\alpha \in_R \mathbb{F}$. Then chose the two blinding vectors $\boldsymbol{s_R}, \boldsymbol{s_L} \in_R \mathbb{F}^n$ and the value $\rho \in_R \mathbb{F}$ and compute the commitment $S = h^{\rho}\boldsymbol{g}^{\boldsymbol{s_L}}\boldsymbol{h}^{\boldsymbol{s_R}}$. Let $y = \text{Hash}(A, S)$, $z = \text{Hash}(A, S, y)$ and $\tau_1, \tau_2 \in_R \mathbb{F}$. Now the it is possible to construct t_1, t_2 defined above. Given this let $T_1 = g^{t_1}h^{\tau_1}$ and $T_2 = g^{t_2}h^{\tau_2}$, next let $X = \text{Hash}(T_1, T_2)$. Now construct the two vectors for the inner product argument: $\boldsymbol{l} = \boldsymbol{x} - z \cdot \mathbf{1}^n - \boldsymbol{s_L} \cdot X$, $\boldsymbol{r} = \boldsymbol{y}^n \circ (\bar{\boldsymbol{x}} + z \cdot \mathbf{1}^n + \boldsymbol{s_R} \cdot X) + z^2\, X$ and calculate the inner product $\hat{t} = \langle \boldsymbol{l}, \boldsymbol{r} \rangle$. Finally compute $\tau_X = \tau_2 x^2 + \tau_1 X + z^2 R$ and $\mu = \alpha + R\, X$. Now use the inner product argument to prove that $\hat{t}$ is indeed the inner product of the two vectors $\mathbf{l}, \mathbf{r}$ using the commitment $P_v = \boldsymbol{g}^{\boldsymbol{l}}\boldsymbol{h}^{\boldsymbol{r}}$, run the algorithm **Prove** defined Construction 6 with the input $(\boldsymbol{g}, \boldsymbol{h}, u, P_v, \hat{t}, \boldsymbol{l}, \boldsymbol{r})$ to construct such a proof denoted $proof_{IP}$. Combine and publish the proof: $proof_{RP} = (\tau_X, \mu, \hat{t}, P, A, S, T_1, T_2, P_v, proof_{IP})$.
- **Verify** $(g, h, C, proof_{RP}) \rightarrow \{0, 1\}$
  Compute the three hash functions $y = \text{Hash}(A, S)$, $z = \text{Hash}(A, S, y)$ and $X = \text{Hash}(T_1, T_2)$. Then given y, z, X compute $h'_i = h_i^{y^{-i+1}}$ for all $i \in \{1, ..., n\}$, $P_l = P \cdot h\mu$ and $P_r = A \cdot S^x \boldsymbol{g}^{-z}\boldsymbol{h'}^{z\boldsymbol{y}^n + z^2 \cdot \mathbf{2}^n}$. Then check if the following equalities hold: $P_l \stackrel{?}{=} P_r \wedge g^{\hat{t}}h^{\tau_X} \stackrel{?}{=} P^{z^2}\, g^{\delta(y,z)}\, T_1^x\, T_2^{x^2}$ and if the output of **Verify** in Construction 6 on the input (proof_{IP}) is 1. If all three criterion is fulfilled then the secret x in the commitment P is in the range $[0, 2^n]$.

Construction 6 : Inner-product argument

Goal: Given a Pedersen vector commitment $P_v = \boldsymbol{g}^{\boldsymbol{s}}\boldsymbol{h}^{\boldsymbol{q}}$ and a value c prove that the two vectors $\boldsymbol{s}, \boldsymbol{q}$ satisfies $\langle \boldsymbol{s}, \boldsymbol{q} \rangle = c$.

- **Prove** $(\mathbf{g}, \mathbf{h}, u, P_v, c, \mathbf{s}, \mathbf{q}) \rightarrow proof_{IP}$
  Let $y = \text{Hash}_{IP}(\mathbf{g}, \mathbf{h}, P_v, c) \in \mathbb{F}^*$ and compute $P'_v = u^{y \cdot c}P$. Let $\mathbf{l}, \mathbf{r}$ be two empty vectors. Run the recursive algorithm **GenerateProof**$(\mathbf{g}, \mathbf{h}, u^{x \cdot c}, P_v, c, \mathbf{s}, \mathbf{q}, \mathbf{l}, \mathbf{r})$ use the output $(g', h', u', P'_v, s', q', \mathbf{l}, \mathbf{r})$ to construct the inner product proof $\text{proof}_{IP} = (\mathbf{g}, \mathbf{h}, u', P_v, s', q', \mathbf{l}, \mathbf{r})$ and output $proof_{IP}$.
- **GenerateProof**$(\mathbf{g}, \mathbf{h}, u, P_v, \mathbf{s}, \mathbf{q}, \mathbf{l}, \mathbf{r}) \rightarrow (g, h, u, P_v, s, q, \mathbf{l}, \mathbf{r})$
  - If the dimension of the vectors $\mathbf{g}, \mathbf{h}, \mathbf{s}, \mathbf{q}$ drop the bold font and publish the proof $proof_{IP} = (g, h, P_v, u, s, q, \mathbf{l}, \mathbf{r})$.
  - Otherwise: Let $n' = n/2$ and define $c_L = \langle \boldsymbol{s}_{[:,n']}, \boldsymbol{q}_{[n',:]} \rangle$ and $c_R = \langle \mathbf{s}_{[n',:]}, \mathbf{q}_{[:,n']} \rangle$. Then use these variables to calculate $L = \mathbf{g}_{[n':]}^{\mathbf{s}_{[:n']}} \mathbf{h}_{[:n']}^{\mathbf{q}_{[n':]}} u^{c_L}$ and $R = \mathbf{g}_{[:n']}^{\mathbf{s}_{[n':]}} \mathbf{h}_{[n':]}^{\mathbf{q}_{[:n']}} u^{c_R}$. Append $L, R \in \mathbb{G}$ to the vectors $\mathbf{l}$ resp $\mathbf{r}$. Now update $y = \text{Hash}_{BP}(L, R)$, and update $\mathbf{g}' = \mathbf{g}_{[:n']}^{y^{-1}} \mathbf{g}_{[n':]}^{y}$, $\mathbf{h}' = \mathbf{h}_{[:n']}^{y} \mathbf{h}_{[n':]}^{y^{-1}}$ and the commitment $P'_v = L^{y^2} P R^{y^{-2}}$. Finally update the vectors $\mathbf{s}, \mathbf{q}$ to $\mathbf{s}' = \mathbf{s}_{[:n']} y + \mathbf{s}_{[n':]} y^{-1}$ and $\mathbf{q}' = \mathbf{q}_{[:n']} y^{-1} + \mathbf{q}_{[n':]} y$. Run the algorithm recursively, **GenerateProof**$(\mathbf{g}', \mathbf{h}', u, P'_v, \mathbf{s}', \mathbf{q}', \mathbf{l}, \mathbf{r})$ with the updated variables. Note that the vectors $\mathbf{g}, \mathbf{h}, \mathbf{s}, \mathbf{q}$ now have the dimension $n' = n/2$, hence performing the recursion until one-dimensional vectors will require $log\, n$ iterations.
- **Verify** $(proof_{IP} = (\mathbf{g}, \mathbf{h}, u, P_v, s, q, \mathbf{l}, \mathbf{r})) \rightarrow \{0, 1\}$
  For $i \in \{0, log(n)\}$ put $n = n/2$ and $y = \text{Hash}(\boldsymbol{l}[i], \boldsymbol{r}[i])$, then update the vectors $\boldsymbol{g}$ and $\boldsymbol{h}$ as well as the variable P'_v according to, $\boldsymbol{g}' = \boldsymbol{g}_{[:,n]}^{y^{-1}} \boldsymbol{g}'^{y}_{[n,:]}$, $\boldsymbol{h} = \boldsymbol{h}_l[:, n]^x \boldsymbol{h}_{[n,:]}^{y^{-1}}$ and $P'_v = L^{y^2} P R^{y^{-2}}$. After iterating over all i the dimension of the vectors $\boldsymbol{g}, \boldsymbol{h}$ is one and the bold font can be dropped. Compute $c = \langle s, q \rangle$ and accept if $P'_v = g^s h^r u^c$.

B

Aggregation

Consider two set membership proofs Σ_1 and Σ_2. Remember that for $i = 1, 2$ it hold that $\Sigma_i = (V_i, a_i, D_i, z_{x_i}, z_{\tau_i}, z_{R_i})$ and $V_i = g^{\frac{\tau_i}{q+x_i}}, a_i = e(V_i, g)^{s_i} e(g, g)^{t_i}$, $D_i = g^{s_i} h^{m_i}$, $z_{x_i} = s_i - cx_i$, $z_{\tau_i} = t_i - c\tau_i$ and $z_{R_i} = m_i + cR_i$, where $s_i, t_i, m_i \in_R \mathbb{F}$, g, h are group elements of the group $\mathbb{G}$, q is a secret key not known by any party, $y = g^q$ is the public key known to any party. The, here assumed same for both parties, challenge is denoted$c$ and is publicly known, and finally x_i, R_i denoted the secret and randomness of prover $i = 1, 2$

The aim is to investigate is an aggregated proof, called Σ without subscript, of Σ_1 and Σ_2 such that $\Sigma = (V, a, D, z_x, z_\tau, z_R)$,$a \stackrel{?}{=} e(V, y)^c e(V, g)^{-z_x} e(g, g)^{z_\tau}$ and if equality it implies that $x_i \in \Phi$ for $i = 1, 2$. The aggregation is done according equation (4.1), it follows that,

$$
\begin{aligned}
LHS =& a_1 a_2 = \Big(e(V_1, g)^{s_1} e(g, g)^{t_1}\Big)\Big(e(V_2, g)^{s_2} e(g, g)^{t_2}\Big) \\
=& \Big(e(g, g)^{\frac{\tau_1 s_1}{q+x_1}} e(g, g)^{t_1}\Big)\Big(e(g, g)^{\frac{\tau_2 s_2}{q+x_2}} e(g, g)^{t_2}\Big) = e(g, g)^{\frac{\tau_1 s_1}{q+x_1} \frac{\tau_2 s_2}{q+x_2} + t_1 + t_2} \\
RHS =& e(V, y)^c e(V, g)^{-z_x} e(g, g)^{z_\tau} = e(V_1 V_2, y)^c e(V_1 V_2, g)^{-z_{x_1} - z_{x_2}} e(g, g)^{z_{\tau 1} + z_{\tau 2}} \\
=& e(g, g)^{cq\left(\frac{\tau_1}{q+x_1} + \frac{\tau_2}{q+x_2}\right)} e(g, g)^{\Big(-(s_1 - cx_1) - (s_2 - cx_2)\Big)\left(\frac{\tau_1}{q+x_1} + \frac{\tau_2}{q+x_2}\right)} e(g, g)^{(t_1 - c\tau_1) + (t_2 - c\tau_2)} \\
=& e(g, g)^{\frac{\tau_1}{q+x_1}\Big(cq - (s_1 - cx_1) - (s_2 - cx_2) - c(q+x_1)\Big) + t_1} e(g, g)^{\frac{\tau_2}{q+x_2}\Big(cq - (s_1 - cx_1) - (s_2 - cx_2) + c)(q+x_2)\Big) + t_2} \\
=& e(g, g)^{\frac{s_1 \tau_1}{q+x_1} + t_1} e(g, g)^{\frac{\tau_1}{q+x_1}\Big(cq + cx_1 - c(q+x_1) - (s_2 - cx_2)\Big)} \\
& e(g, g)^{\frac{s_2 \tau_2}{q+x_2} + t_2} e(g, g)^{\frac{\tau_2}{q+x_2}\Big(cq + cx_2 - c(q+x_2) - (s_1 - cx_1)\Big)} \\
=& e(g, g)^{\frac{s_1 \tau_1}{q+x_1} + t_1} e(g, g)^{\frac{s_2 \tau_2}{q+x_2} + t_2} e(g, g)^{\frac{\tau_1}{q+x_1}\Big(-(s_2 - cx_2)\Big)} e(g, g)^{\frac{\tau_2}{q+x_2}\Big(-(s_2 - cx_2)\Big)} \\
=& e(g, g)^{\frac{s_1 \tau_1}{q+x_1} + t_1} e(g, g)^{\frac{s_2 \tau_2}{q+x_2} + t_2} e(V_1, g)^{-z_{x_2}} e(V_2, g)^{-z_{x_1}} \\
& \implies LHS \neq RHS
\end{aligned}
$$

It is seen that the terms $e(V_1, g)^{-z_{x_2}} e(V_2, g)^{-z_{x_1}}$ are not cancelled, which results in that the left hand side does not equal the right hand side.

C

Multiple aggregating parties construction

Construction 7 : Aggregation of non interactive set membership proof

Goal: Given the Pedersen commitments $C_i = g^{x_i}h^{R_i}$ for $i \in \mathcal{S}$. The goal is to prove that all secrets x_i belongs to the set Φ. Without revealing anything else about the secrets x_i.

- **SetUp** $(1^\lambda, \Phi) \rightarrow (sk, pp)$
  Let g be a generator of the group $\mathbb{G}$ and h and element in the group such that $log_g(h)$ is unknown. Pick uniformly at random $\chi \in_R \mathbb{F}$ and put $sk = \chi$. Define $y = g^\chi$ and $A_i = g^{\frac{1}{\chi+i}}$ $\forall i \in \Phi$, output $pp = (g, h, y, \{A_i\}_{i\in\Phi})$.
- **Prove** $(pp, i, C_i, \Phi) \rightarrow \Sigma_i$
  Pick uniformly at random $\tau_i \in_R \mathbb{F}$, choose from the set $\{A_i\}$ the element A_{x_i} and calculate $V = A_{x_i}^{\tau_i}$. Pick uniformly random three values $s_i, t_i, m_i \in_R \mathbb{F}$. Put $a_i = e(V_i, g)^{-s_i}e(g, g)^{t_i}$ ($e(\cdot,\cdot)$ is a bilinear mapping as described above), $D = g^{s_i}h^{m_i}$, and $c = \text{Hash}(C_i, V_i, a_i, D_i)$. Finally compute $z_{x_i} = s_i - x_i c_i$, $z_{R_i} = m_i - R_i c_i$ and $z_{\tau_i} = t_i - \tau_i c_i$ then construct and publish $\Sigma_i = (V_i, a_i, D_i, z_{x_i}, z_{\tau_i}, z_{R_i})$.
- **Aggregate** $(pp, k, \{\Sigma_i\}_{i\in\mathcal{S}_k}) \rightarrow \Sigma_{a_k}$
  Given a set of set membership proofs $\{\Sigma_i\}_{i\in\mathcal{S}_k}$. Aggregate the values $(\{D_i\}_{i\in\mathcal{S}_k}\{z_{x_i}\}_{i\in\mathcal{S}_k}\{z_{r_i}\}_{i\in\mathcal{S}_k}) \mapsto D_{a_k}, z_{x_{a_k}}, z_{R_{a_k}}$ according to equation (4.3). Then construct and publish the aggregated proof $\Sigma_{a_k} = (\{V_i\}_{i\in\mathcal{S}_k}, \{a_i\}_{i\in\mathcal{S}_k}, D_{a_k}, \{z_{x_i}\}_{i\in\mathcal{S}_k}, z_{x_{a_k}}, \{z_{\tau_i}\}_{i\in\mathcal{S}_k}, z_{R_{a_k}})$.
- **CalculateChallenges** $(\{C_i\}_{i\in\mathcal{S}}\{\Sigma_i\}_{i\in\mathcal{S}}) \rightarrow \{c_i\}_{i\in\mathcal{S}}$
  For all $i \in \mathcal{S}$ parse the proof Σ_i, then compute the challange $c_i = Hash(C_i, V_i, a_i, D_i)$. Finally output the set of all challenges $\{c_i\}_{i\in\mathcal{S}}$.
- **Verify** $(pp, \{\Sigma_{a_k}\}_{k\in\mathcal{K}}, \{C_i\}_{i\in\mathcal{S}}, \{c_i\}_{i\in\mathcal{S}}) \rightarrow \{0, 1\}$
  For all $k \in \mathcal{K}$ compute the product of the challenges $c_k = \prod_{i\in\mathcal{S}_k} c_i$. Check if $D_{a_k} \stackrel{?}{=} \left(\prod_{i\in\mathcal{S}_k} C_i\right)^{c_k} h^{z_{Ra_k}} g^{z_{xa_k}}$ Then for check if $a_i \stackrel{?}{=} e(V_i, y)^c_i e(V_i, g)^{-z_{x_i}} e(g, g)^{z_{\tau_i}}$ for all $i \in \mathcal{S}$. If the equalities holds the provers has convinced the verifier that $x_i \in \Phi$ for all x_i such that $i \in \mathcal{S}$ return 1 otherwise return 0.