CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Aggregated Set Membership Proofs

Aggregated Signature-Based Set Membership Proofs and implementation 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 and implementation 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 Set Membership Proofs
Aggregated Signature-Based Set Membership Proofs and implementation 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 Set Membership Proofs
Aggregated Signature-Based Set Membership Proofs and implementation 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

This thesis addresses the issue of inflated computational complexity for the verification of multiple zero-knowledge proofs. More precisely, verification of numerous zero-knowledge set membership proofs performed by a single verifier is considered. To reduce the computations required by such a verifier *Aggregated Set Membership Proofs* are introduced.
Aggregated set membership proofs unifies multiple set membership proofs into one aggregated proof, such that the validity of the aggregated proof implies the validity of all individual proofs. Completeness, soundness and zero-knowledge requirements are established for zero-knowledge aggregated set membership proofs.
A concrete construction of aggregated set membership proofs is presented and proved to satisfy the completeness, soundness and zero-knowledge requirements. The construction is a partial aggregation of signature-based set membership proofs, [5], and is referred to as *aggregated signature-based set membership proofs*.
A general technique to verify clients in verifiable additive homomorphic secret sharing is derived. The clients are verified by computing zero-knowledge proofs, derived from Pedersen commitments, of some given statement and then the proofs are validated by a verifier. If the proved statement is that the shared secrets belong to a discrete set, clients construct set membership proofs. Usually, several clients participate in verifiable additive homomorphic secret sharing protocols resulting in that the verification of clients is computationally expensive.
A prototype implementation considering 100 clients showed that the runtime for verification of clients was reduced by 13% when verifying an aggregated signature-based set membership proof compared to verifying the same proofs without performing the aggregation.

Keywords: Aggregated Set Membership Proofs, Zero-knowledge proofs, VAHSS, cryptography

Acknowledgements

I wish to express my gratitude towards my supervisors Georgia Tsaloli and Katerina Mitrokotsa for the support and help during the work. The thesis would not have been as extensive without your guidance, expertise and innovation.
I also want to thank Markus Pettersson for consultation regarding the implementation of the constructions.

Hanna Ek, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1

Introduction

The digitalisation of our society leads to a need for 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 providing your membership number to verify that you are a member of a site.

In many applications, however, it is sufficient to share more abstract information. Consider the example with the membership number. To prove that you are a member of a group it is sufficient to prove that you are one of the members, without specifying which one.

To illustrate the above idea, consider two parties Alice and Bob. Alice is a subscriber to Bob's paper and wishes to convince Bob of this, 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 and checks its validity. Bob is now convinced that Alice's name is on the list, but not which of the names it is, and allows Alice to get access to the paper. For the remainder of this thesis, Alice will be denoted 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 called a *set membership proof*. Constructions of set membership proofs are computationally expensive. Consequently, a lot of research has been done to reduce the computations 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. We are considering constructions consisting of multiple provers and one verifier. An example of such a construction is the verification of clients in a VAHSS protocol, [18]. In such a construction, using set membership proofs, the computational complexity for the verifier grows linearly with the number of provers. This since the verifier has to verify all received proofs individually.

A method for reducing the computations required of the verifier is aggregating the proofs beforehand. Then the verifier only needs to verify one proof, the aggregated proof. The aggregated proof should be such that its validity implies the validity of all individual proofs.

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

Purpose

This paper consists of two parts. The first is to explore the possibility to aggregate set membership proofs, in order to reduce the computational complexity for the verification of multiple provers. Initially, a general description of aggregate set membership proofs is sought. Then, given such a description we provide a concrete construction of an aggregated set membership proof and prove its security.

The second purpose is to obtain a method for extending a VAHSS construction, [18], to verify clients' inputs. Then we compare the runtime of using aggregated set membership proof contra Bulletproof for the verification of clients.

Limitations

The exploration of obtaining a construction of an aggregated set membership proofs is limited to investigate if one specific set membership proof can be aggregated.

Contribution

The main results obtained in this paper are:

- A general description of aggregated set membership proofs including the completeness, soundness and zero-knowledge requirements such proofs should fulfil.
- A construction and implementation of an aggregated signature-based set membership proof. The presented construction is proved to satisfy the completeness soundness and zero-knowledge properties for aggregated set membership proofs.
- The VAHSS construction using homomorphic hash functions to verify servers [18] is modified to additionally verifying clients' inputs. The clients' inputs are verified using either range proofs or set membership proofs. The construction is also modified to be compatible with aggregated set membership proofs for the verification of clients.
- Implementation of all proposed constructions in Golang and runtime comparison of the constructions.

Related work

Zero-Knowledge proofs

Set membership proofs

Set membership proofs are zero-knowledge proofs proving that a secret belongs to a discrete set. A construction of set membership proofs based on bilinear groups and public signatures of elements in the set is presented in [5]. Their construction has a computational complexity of $\mathcal{O}(1)$ for the construction and verification of the set membership proofs. However, when multiple proofs are verified at once the computational complexity grows linearly in the number of proofs.

Range Proofs

Range proofs are strongly connected to set membership proofs, but instead of proving that a secret belongs to a discrete set, they prove that a secret belongs to a numerical range.

Bulletproofs are state-of-the-art range proofs used in many real-world applications. Bulletproofs prove that a secret belongs to a given range, by making use of an inner-product argument [4]. Bulletproofs can be aggregated using a simple multi-party computation protocol. Consequently, aggregation of Bulletproofs requires interaction between the provers during the construction of the proof. Unlike the focus of this paper, where non-interactive aggregation of zero-knowledge proofs is considered.

A recently published paper presents range proofs that are faster than Bulletproofs for both the prover and the verifier, [8]. Their range proofs make use of square decomposition methods by converting commitment schemes over $\mathbb{Z}_p$ into proofs over $\mathbb{Z}$ between bounded-range integers. In applications where multiple provers participate each proof is verified individually, leading to that the computational complexity of the verification grows linearly in the number of provers.

Prio+

Prio+, [1], computes aggregated statistics on multiple clients data, without revealing the data of individual clients and is robust against malicious clients. Communication between the servers, of constant size per client, is required to compute the aggregated statistics of the clients' data. Prio+ is strongly connected to the Prio construction, [7], and can be seen as a development that reduces the computations required by the clients. In Prio+ clients use Boolean secret-sharing to convince servers of their honesty, instead of computationally expensive zero-knowledge proofs. Prio+ constitutes the same purpose as client and server VAHSS. However, in Prio+ the servers must communicate to verify the clients unlike client and server VAHSS where no communication is required between the servers.

Organisation

In chapter 2 the theoretical background is presented. First cryptographic principles are treated then a more detailed description of set membership proofs and range proofs is given. Chapter 3 presents a general definition of aggregated set membership proofs. Based on this definition chapter 4 presents a construction of an aggregation of signature-based set membership proofs. This construction is implemented and compared in terms of runtime with itself for different settings and additionally compared to the state-of-the-art Bulletproofs in chapter 5. Chapter 6 presents a client and server VAHSS. Clients are verified using Bulletproofs, aggregated and not aggregated signature-based set membership proofs. In chapter 7 a discussion about the results is given together with a conclusion.

2 Background

This chapter presents the theory used in this paper. In section 2.1 first notation, theorems, definitions and assumptions are stated, then cryptographic preliminaries are explained. In section 2.2 two different types of zero-knowledge proofs are presented, more precisely signature-based set membership proofs and Bulletproofs.

2.1 Preliminaries

Notations

To make the text more comprehensive the following notations and definitions are defined here:
$\mathbb{F} = \mathbb{Z}_p$ denotes a finite field, where p is a large prime and $\mathbb{G}$ denotes a unique subgroup of order q. $g \in \mathbb{G}$ is defined to be the group generator of $\mathbb{G}$ and $h \in \mathbb{G}$ a group element, such that $log_g\, h$ is unknown and h is a co-prime to p. The notation $y \in_R \mathbb{Y}$, means that an element y is chosen at random from the set $\mathbb{Y}$.

Definitions, Theorems and Assumptions

The assumptions given here are the assumptions that all cryptographic constructions in this paper rely on. The discrete logarithm assumption and the q-strong Diffie Hellman assumption defined below does not hold in the presence of quantum computers. Thus the cryptographic constructions presented in this paper are not guaranteed to be post quantum secure.

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 $x \in \mathbb{F}$, 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), [14], hides a secret x by splitting it into shares, such that any subset, $\mathcal{S}$, of shares smaller than a threshold τ, i.e $|\mathcal{S}| < \tau$, reveals no information about the value of x. If a secret x is split into m shares denoted x_i, such that $i \in \{1, ..., m\}$ and to reconstruct the value of x at least τ shares has to be combined, it is called a (τ, m)-threshold scheme. In this paper, the threshold is set equal to the number of shares, $\tau = m$.

Verifiable Additive Homomorphic secret sharing

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

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 m parties, referred to as servers, collaborates to compute the sum of multiple clients' secrets. Each client split their secret into m shares and sends one share to each server. The servers compute and output 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 the final sum is the correctly computed sum of the clients' secrets.

In section 6.1 a specific construction of a VAHSS is presented and in Appendix A the correctness, security and verifiability requirement of a VAHSS construction is stated.

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 $\mathcal{H}(x + x') = \mathcal{H}(x) * \mathcal{H}(x')$.

A homomorphic hash function that satisfies the thee properties is the function $\mathcal{H}_1(x) : \mathbb{F} \mapsto \mathbb{G}$ and $\mathcal{H}_1(x) = g^x$, [19].

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 $R \in_R \mathbb{F}$, [13]. 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 satisfies $\mathbb{E}(x, R) = \mathbb{E}(x', R')$ while $x \neq x'$ then it must hold that $R \neq R'$ $mod\, q$ and*

$$x - x' = (R - R')\ log_g\ h\ mod\ p \tag{2.1}$$

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 this person is able to provide two equal commits, $\mathbb{E}(x, R) = \mathbb{E}(x', R')$ such that $x \neq x'$. It is impossible to construct two equal commits hiding different secrets, since $log_g h$ is assumed to be unknown. The Pedersen commitment scheme is computationally binding under the discrete logarithm assumption and it is perfectly hiding of the secret, [13].

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 $C_1 \cdot C_2 = \mathbb{E}(x_1 + x_2, R_1 + R_2)$.

The Pedersen commitment is similar to the homomorphic hash function $\mathcal{H}_1$, discussed above, and the hash function can be seen as a special case of a 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 called bilinear pairing) maps two group elements from one group to an element in another group. This paper considers admissible bilinear mapping fulfilling Definition 3. Generally, the definition of an admissible bilinear mapping maps two elements from different groups to a third group, i.e the map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, in this paper it always holds that $\mathbb{G}_1 = \mathbb{G}_2$ and thereby 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 exists 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 $a, b \in \mathbb{F}$,*

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

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

Bohen-Boyen Signatures

Consider 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 [10]. The idea behind a ZKP is that after successfully performing a ZKP a certain statement about a secret, denoted x, has been verified to be true (or false) without having revealed any other information about the secret beyond the statement. In this thesis non-interactive ZKP that ensures proof of knowledge (PoK) are considered.

A ZKP consists of two parties a *prover* and a *verifier*, assume both parties have 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. PoK means that the verifier is not only convinced there exists a witness w of the secret x, but also that the prover knows such a witness.

A ZKP should fulfil the requirements stated in Definition 4. Informally the requirements are: 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 the output from the* ***`Simulator`*** *and* ***`Prove`*** *is indistinguishable, i.e have the same distribution given that $x \in \mathcal{L}$.*

Zero-knowledge range proof (ZKRP) and zero-knowledge set membership proofs (ZKSM) where the statement being proved is that the secret belongs to a predetermined range or set, will be considered in this paper.

Fiat-Shamir heuristic

The Fiat-Shamir heuristic [2] can be used to convert an interactive construction to a non-interactive construction. In this paper, it is 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 to convince the verifier of its validity. The Fiat-Shamir heuristic replaces the random challenge sent by the verifier with the output of a hash function computed from the partial-proof up to this point. The Fiat-Shamir heuristic converts an interactive ZKP to a non-interactive ZKP while preserving security and full zero-knowledge, relying on the random oracle model (ROM) [2] .

2.2 Set Membership Proofs and Range Proofs

Zero-knowledge set memberships proofs allow a prover to convince a verifier that the value of a secret is in an allowed set, without revealing any other information about the secret. Formally they are proofs of the 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 prove that a secret belongs to a range, instead of a set. Since ranges are continuous sets, zero-knowledge range proofs are less general than zero-knowledge set membership proofs. Zero-knowledge range proofs do not reveal any information about the secret beyond the fact that the secret belongs to the range. Formally they are proofs of the following statement:

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

Hereafter zero-knowledge set memberships proofs and zero-knowledge range proofs are denoted set memberships proofs and range proofs respectively.

Note that the above statements assume that x is a secret hidden in a Pedersen commitment. This is not a requirement for set membership proofs and range proofs, however only such proofs are studied in this thesis.

All set membership proofs and range proofs presented in this paper fulfils Definition 4.

Signature-based Set membership proofs

This section presents a construction of set membership proofs referred to as *signature-based* set membership proofs, originally presented in [5].

Bohen-Boyen signatures, A_i for each element i in the set Φ, is published in the set up phase. The signature-based set membership proofs have a computational complexity of $\mathcal{O}(1)$ for the proof construction and verification, assuming that the signatures $\{A_i\}_{i\in\Phi}$ are known to both the prover and the verifier

To prove a secret is in the set Φ, the prover chooses the public signature representing the secret x, i.e A_x, and publishes the value $V = A_x^\tau$, where $\tau \in_R \mathbb{F}$. Then constructs a proof that convinces the verifier that: 1) the published value V is indeed equal to A_x^τ where A_x is a signature of $x \in \Phi$. 2) the secret in the pre-published Pedersen commitment C is a commitment to the same secrets as the signature V.

Construction 1 describes the PPT algorithms (**SetUp**, **Prove**, **Verify**) that builds a signature-based set membership proof. Construction 1 is modified compared to the original construction of signature-based set membership proof, presented in [5], according to the Fiat-Shamir heuristic. The notation $e(\cdot,\cdot)$ in the construction refers to an admissible bilinear mapping as defined previously in section 2.1.

Construction 1 : Non-interactive set membership proofs

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** $(1^\lambda, \Phi) \rightarrow (sk, pp)$
  Let g be a generator of the group $\mathbb{G}$ and h an 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** $(pp, C, x, \Phi) \rightarrow \Sigma = (V, a, D, z_x, z_\tau, z_R)$
  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 at 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 values to compute the challenge, $c = \text{Hash}(C, V, a, D)$. Given this challenge compute $z_x = s - xc$, $z_R = m - Rc$ and $z_\tau = t - \tau c$, finally construct and publish the proof, $\Sigma = (V, a, D, z_x, z_\tau, z_R)$.
- **Verify** $(pp, C, \Sigma) \rightarrow \{0, 1\}$
  Check if $D \overset{?}{=} C^c h^{z_R} g^{z_x} \wedge a \overset{?}{=} e(V, y)^c e(V, g)^{-z_x} e(g, g)^{z_\tau}$. If the equality holds return 1 otherwise return 0.

Signature-based range proofs

Signature-based set membership proofs can be used to construct efficient signature-based range proofs. Rewriting the secret x in base u such that,

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

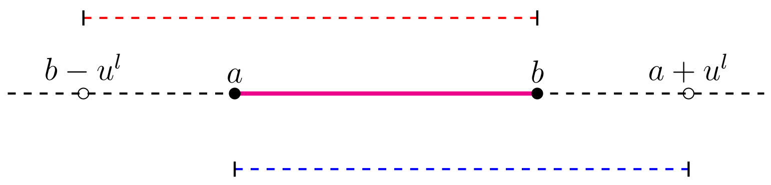

Figure 2.1: Extending range proofs to consider arbitrary ranges. It is illustrated that if $x \in [b - u^l, b) \wedge x \in [a, a + u^l)$ then $x \in [a, b]$

and then using set membership proofs to prove that $x_j \in [0, u) \; \forall j \in \mathbb{Z}_l$, constructs to a range proof that $x \in [0, u^l)$.

This range proof can be generalised to consider arbitrary ranges $[a, b]$, where $a > 0$ and $b > a$. This is obtained by realising that proving that $x \in [a, a + u^l)$ and $x \in [b - u^l, b)$, is equivalent to proving that the secret, x, belongs 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 be translated into proving $x - a \in [0, u^l)$ and $x - b + u^l \in [0, u^l)$, since both a, b are public.

In [6] an optimised implementation of the signature-based range proof, for arbitrary ranges, is presented, reducing the complexity with a factor of 2. This rather small reduction is important when a verifier is required to check the validity of multiple range proofs simultaneously.

Bulletproofs

Bulletproofs are state-of-the-art range proofs that are integrated into many real-world protocols. They are for example used in crypto-currencies, to prove that a value is in an allowed range or above some threshold. In this paper, Bulletproofs are used for runtime comparison with other constructions and to verify the clients' inputs in a VAHSS construction.

Bulletproofs are originally presented in [4] and prove that a secret in a Pedersen commitment belongs to the range $[0, 2^n)$, where n is a power of 2.

The construction of Bulletproofs builds 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. Given a Pedersen commitment $C = g^x h^R$, of the secret x, a prover wants to convince a verifier that the secret belongs to the interval $[0, 2^n)$. The binary representation of the secret x is $\boldsymbol{x} \in \{0, 1\}^n$ which can be equivalently written as $x = \langle \boldsymbol{x}, \mathbf{2}^n \rangle$. This inner product can the be used to construct an inner product argument of the vector $\mathbf{x}$, i.e the secret x.

Bulletproofs modified according to the Fiat-Shamir heuristic are considered. A construction of non-interactive Bulletproofs and a non-interactive inner product argument is given in [11].

3 Aggregated Set Membership Proofs

It is of high importance to keep the computational complexity for the verification of set membership proofs small, specially in applications where one verifier validates multiple provers. If set membership proofs, which are proofs of the statement:

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

are used to verify multiple provers then the verifier would have to verify each prover individually. This leads to that the computational complexity for verification grows linearly in the number of provers and the verification algorithm quickly becomes a bottleneck in application where many provers participates.

This motivates the question if set membership proofs can be aggregated, and thereby decreasing the computations required to verify multiple provers.

A definition of aggregated set membership proofs is presented in Definition 5. An aggregated set membership proof is defined to be a 5-tuple of PPT algorithms, (**SetUp**, **Prove**, **Aggregate**, **CalculateChallanges**, **Verify**).

Definition 5 (**Aggregated set membership proofs**)**.** *Aggregated set membership proofs are a zero-knowledge proof of the statement:*

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

Where C_i is a Pedersen commitment to the secret $x_i \in \mathbb{F}$ and $R_i \in_R \mathbb{F}$ are chosen at random, for all $i \in \mathcal{S}$.

The statement implies that after successfully having performed an aggregated set membership proof protocol, it is proved that the aggregated Pedersen commitment, C, is a product of the Pedersen commitment,$\{C_i\}_{i\in\mathcal{S}}$ and that for all $i \in \mathcal{S}$ C_i is a Pedersen commitment to a secret x_i belonging to the set Φ.

*A construction of aggregated set membership proofs is a 5-tuple of PPT-algorithms (**SetUp**, **Prove**, **Aggregate**, **CalculateChallanges**, **Verify**).*

- ***SetUp*** $(1^\lambda, \Phi) \to (pp, sk)$
  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*** $(pp, i, C_i, x_i, \Phi) \to \Sigma_i$
  On the input, the public parameters pp, $i \in \mathcal{S}$ denoting the index of the prover p_i, a secret x_i and a Pedersen commitment C_i of the secret. The algorithm outputs a polynomial time verifiable zero-knowledge proof of the statement in equation (2.2)*, denoted Σ_i.*
- ***Aggregate*** $(pp, \{\Sigma_i\}_{i\in\mathcal{S}} \to \Sigma_a$
  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*** $(\{C_i\}_{\in\mathcal{S}}, \{\sigma_i\}_{i\in\mathcal{S}}) \to \{c_i\}_{i\in\mathcal{S}}$
  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*** $(pp, \Sigma_a, \{C_i\}_{i\in\mathcal{S}}, \{c_i\}_{i\in\mathcal{S}}) \to \{0, 1\}$
  On input the aggregated set membership proof public parameters, pp, aggregated proof, Σ_a, the Pedersen commitments, $\{C_i\}_{i\in\mathcal{S}}$, and challenges ,$\{c_i\}_{i\in\mathcal{S}}$, the algorithm outputs either 1 *or* 0*.*

Aggregated set membership proofs consist of the following parties: provers, an aggregator and a verifier. A schematic figure of the interaction between the parties is seen in Figure 3.1. The provers, a group of many individual provers, individually run the algorithm **Prove** and publishes the computed proof. The aggregating party retrieves all proofs, runs the algorithm **Aggregate** and publishes the aggregated proof. Finally, the verifier validates the aggregated proof by running the algorithm **Verify**. The aggregation can be split between multiple parties implying there are many aggregators, this is discussed more in the next chapter. The algorithms **SetUp** and **CalculateChallenges** is either performed by the verifier or by an independent trusted party.

The algorithms of aggregated set membership proofs, presented in Definition 5, should fulfil the below completeness, soundness and zero-knowledge requirements:

- **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 $\texttt{Verify}(\texttt{Aggregate}(\{\texttt{Prove}(pp, i, C_i, \Phi)\}_{i\in\mathcal{S}})) = 1$.
- **Soundness** If for any $i \in \mathcal{S}$ the witness Σ_i does not satisfy the instance $x_i \notin \Phi$, then the probability $\text{Prob}[\texttt{Verify}(\texttt{Aggregate}(\{\texttt{Prove}(pp, i, C_i, \Phi)\}_{i\in\mathcal{S}})) = 1] < \varepsilon$, for a sufficiently small ε.
- **Zero-knowledge** A proof system is *honest verifier zero-knowledge* if there exists 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 $x \in \Phi$.

These requirements can be seen as a modification of the requirements given in Definition 4 to aggregated set membership proof. Note that the zero-knowledge property should hold for the provers, not the aggregation.

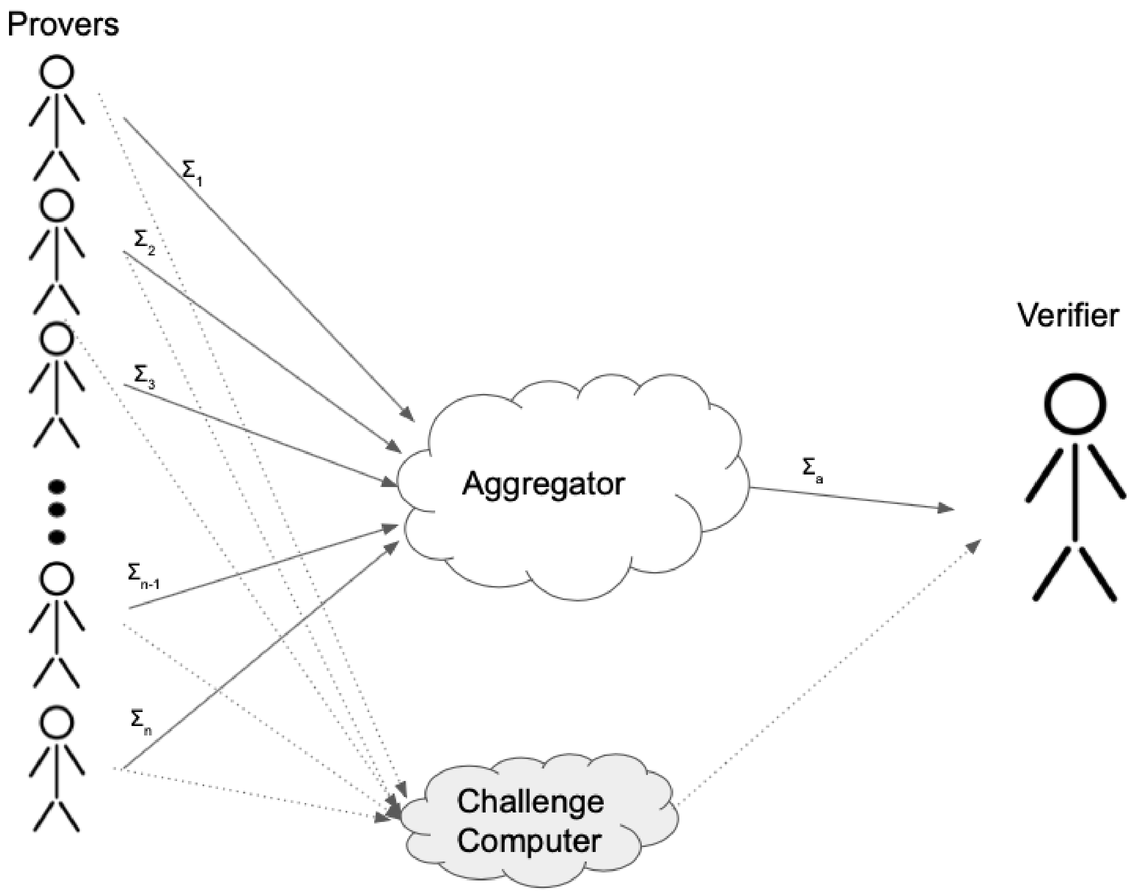

Figure 3.1: Schematic figure of the interaction between Provers, Aggregator and Verifier in aggregated set membership proofs.

4

Aggregated Signature-Based Set Membership Proofs

This chapter presents a construction of aggregated set membership proofs derived from the non-interactive signature-based set membership proofs, [5]. In section 4.2 it is proved that the aggregation is sound and in section 4.3 the completeness, soundness and zero-knowledge requirements, stated in the chapter 3, is proved to hold for the aggregated signature-based set membership proofs.

4.1 Construction

Construction 2 presents an aggregated signature-based set membership proof, derived from the non-interactive signature-based set membership proofs [5].

The algorithm **Aggregate**, in Construction 2, partly aggregates a group of set membership proofs by constructing the aggregated proof, Σ_a. The aggregated values D_a, z_{x_a}, z_{R_a} in Σ_a are computed according to Equation (4.1).

$$\begin{aligned}
D_a &= \prod_{i \in \mathcal{S}} D_i^{\prod_{\substack{j \in \mathcal{S} \\ j \neq i}} c_j} = g^{\sum_{i \in \mathcal{S}} \left(\prod_{\substack{j \in \mathcal{S} \\ j \neq i}} c_j\right) s_i} h^{\sum_{i \in \mathcal{S}} \left(\prod_{\substack{j \in \mathcal{S} \\ j \neq i}} c_j\right) m_i} \\
z_{x_a} &= \sum_{i \in \mathcal{S}} \Big(\prod_{\substack{j \in \mathcal{S} \\ j \neq i}} c_j\Big) z_{x_i} = \sum_{i \in \mathcal{S}} \Big(\prod_{\substack{j \in \mathcal{S} \\ j \neq i}} c_j\Big) s_i - \Big(\prod_{j \in \mathcal{S}} c_j\Big) \sum_{i \in \mathcal{S}} x_i \\
z_{R_a} &= \sum_{i \in \mathcal{S}} \Big(\prod_{\substack{j \in \mathcal{S} \\ j \neq i}} c_j\Big) z_{R_i} = \sum_{i \in \mathcal{S}} \Big(\prod_{\substack{j \in \mathcal{S} \\ j \neq i}} c_j\Big) m_i - \Big(\prod_{j \in \mathcal{S}} c_j\Big) \sum_{i \in \mathcal{S}} R_i
\end{aligned} \tag{4.1}$$

The set $\mathcal{S}$ denotes the index set of the provers and each prover p_i $i \in \mathcal{S}$ publishes a set membership proof $\Sigma_i = (V_i, a_i, D_i, z_{x_i}, z_{\tau_i}, z_{R_i})$ constructed according to the algorithm **Prove**. The challenges, denoted c_i for $i \in \mathcal{S}$, are computed according to the algorithm **CalculateChallenges**.

The above aggregation has the computational complexity $\mathcal{O}(|\mathcal{S}|^2)$. This is reduced to be linear in $|S|$, by considering the following optimization. Instead of computing the product $\prod_{j \in \mathcal{S},\, j \neq i} c_j$ for each i, it is sufficient to compute the product $c = \prod_{j \in \mathcal{S}} c_j$ once and then the truncated product can be obtained by noting that $\prod_{j \in \mathcal{S}, j \neq i} c_j = c/c_i$. Thereby, for each $i \in \mathcal{S}$, it is sufficient to perform one division instead of computing the product $\prod_{j \in \mathcal{S},\, j \neq i} c_j$.

Due to the aggregation, the equality $D_a = C^c h^{z_{R_a}} g^{z_{x_a}}$ is checked once instead of once for each proving party in the protocol. This can bee seen by comparing the algorithm **Verify** in Construction 2 with running the algorithm **Verify** in Construction 1 once for each prover.

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 construction proves that all secrets x_i belong to the set Φ, without revealing anything else about the secrets.

- **SetUp** $(1^\lambda, \Phi) \to (sk, pp)$
  Let g be a generator of the group $\mathbb{G}$ and h an 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** $(pp, i, C_i, x_i, \Phi) \to \Sigma_i$
  Pick uniformly at random $\tau_i \in_R \mathbb{F}$, choose from the set $\{A_i\}$ the element A_{x_i} and calculate $V_i = A_{x_i}^{\tau_i}$. Pick uniformly at random three values $s_i, t_i, m_i \in_R \mathbb{F}$. Put $a_i = e(V_i, g)^{-s_i}e(g, g)^{t_i}$, $D = g^{s_i}h^{m_i}$, $c = \text{Hash}(C_i, V_i, a_i, D_i)$ and compute $z_{x_i} = s_i - x_ic_i$, $z_{R_i} = m_i - R_ic_i$ and $z_{\tau_i} = t_i - \tau_ic_i$. Finally construct and publish the proof $\Sigma_i = (V_i, a_i, D_i, z_{x_i}, z_{\tau_i}, z_{R_i})$.
- **Aggregate** $(pp, \{\Sigma_i\}_{i\in\mathcal{S}}) \to \Sigma_a$
  Given a set of 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 (4.1). 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** $(\{C_i\}_{i\in\mathcal{S}}, \{\Sigma_i\}_{i\in\mathcal{S}}) \to \{c_i\}_{i\in\mathcal{S}}$
  For all $i \in \mathcal{S}$ parse the proof Σ_i, then compute the challenge $c_i = Hash(C_i, V_i, a_i, D_i)$. Finally output the set of all challenges $\{c_i\}_{i\in\mathcal{S}}$.
- **Verify** $(pp, \Sigma_a, \{C_i\}_{i\in\mathcal{S}}, \{c_i\}_{i\in\mathcal{S}}) \to \{0, 1\}$
  Compute the product of the challenges $c = \prod_{i\in\mathcal{S}} c_i$. Check if $D_a \stackrel{?}{=} \left(\prod_{i\in\mathcal{S}} C_i\right)^c h^{z_{R_a}}g^{z_{x_a}} \wedge a_i \stackrel{?}{=} e(V_i, y)_i^c e(V_i, g)^{-z_{x_i}}e(g, g)^{z_{\tau_i}}$ for all $i \in \mathcal{S}$. If the equalities hold return 1 otherwise return 0.

In Construction 2 the algorithm **Prove** is run by all provers, the algorithm **Aggregate** is run by the aggregating party and the algorithm **Verify** is performed by the verifier. The algorithm **SetUp** is assumed to be performed in advance by a trusted party.

In the aggregated signature-based set membership proof the challenges are given as input to the verification algorithm, unlike Construction 1 where they are computed in the verification algorithm. This raises the question, which party should be responsible for computing the challenges? It is not desired that the party performing the aggregation computes the challenges since it creates opportunities for cheating. For the same reason, the proving parties should not compute the challenges. The challenges are constructed according to the Fiat-Shamir heuristic. Two important characteristics of the Fiat-Shamir heuristic are that: the prover does not know the challenge when constructing the proof and that the verifier re-computes the challenge, to check that it is correctly computed. If the challenges are computed by the aggregating party they are known when constructing the aggregated proof and if they are computed by the provers they are never checked. Thereby, the algorithm **CalculateChallenges** in Construction 2 is assumed to be performed by an honest party independent of both the proving parties and the aggregating party.

Another alternative is to provide the entire proofs $\{\Sigma_i\}_{i\in\mathcal{S}}$ as input to the verification algorithm, enabling the verifier to compute the challenges. This would consequently increase the computations required by the verifier. For the rest of this paper, if nothing else is mentioned, it is assumed that the algorithm **Calculate-Challenges** is performed according to Construction 2 by a trusted party.

Multiple aggregating parties

To reduce the computational load of the aggregating party it is possible to split the aggregation between multiple parties.

Let the set $\mathcal{K}$ represent the set of parties performing the aggregation, and the set $\mathcal{S}_k$ represent the assigned set of set membership proofs to the k^{th} aggregating party. The sets $\mathcal{S}_k$ for $k \in \mathcal{K}$ 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 $i \neq j$.

An aggregated signature-based set membership proof, where the aggregation is split between several parties, is presented in appendix B in Construction 5. The adjustments made compared to Construction 2 are: the algorithm **Aggregate** is performed by each party in the set $\mathcal{K}$ on the input $(pp, \{\Sigma_i\}_{i\in\mathcal{S}_k})$, outputting the aggregated proof Σ_{a_k}. The algorithm **Verify** is performed once on the input $(pp, \{\Sigma_{a_k}\}_{k\in\mathcal{K}}, \{C_i\}_{i\in\mathcal{S}}, \{c_i\}_{i\in\mathcal{S}})$ and is modified, compared to Construction 2, such that it verifies multiple aggregated proofs. By letting the set of aggregating parties $\mathcal{K}$ consist of one element k, and $\mathcal{S}_k = \mathcal{S}$ constructions 2 and 5 are equivalent.

An illustrative figure of how the aggregation is split between the aggregating parties is seen in Figure 4.1. Each prover generates a set membership proof, and then sends it to its assigned aggregating party. Then each aggregating party aggregates the received set membership proofs, according to the algorithm **Aggregate** in Construction 5, and sends the aggregated proof to the verifier. Finally the verifier validates all aggregated proofs.

In Figure 4.1 the computation of the challenges is done by an independent party, which introduces a new party to the scheme. To compute the challenges without introducing a new party the aggregating parties can compute the challenges. However, as discussed previously the same party that aggregates a set of proofs should not compute the challenges for these proofs. To use the aggregating parties for the computations, assume that they are not collaborating and linked in a closed chain. Then each aggregating party computes the challenges for the set of proofs aggregated by the consecutive party in the chain.

4.2 Soundness of Aggregation

In this section it is proved that the aggregation must be performed according to the algorithm **Aggregate** if the aggregated proof Σ_a validates true in Construction 2. This is proved under the assumption that: proving parties are not communicating or collaborating between themselves, proving parties are not communicating or

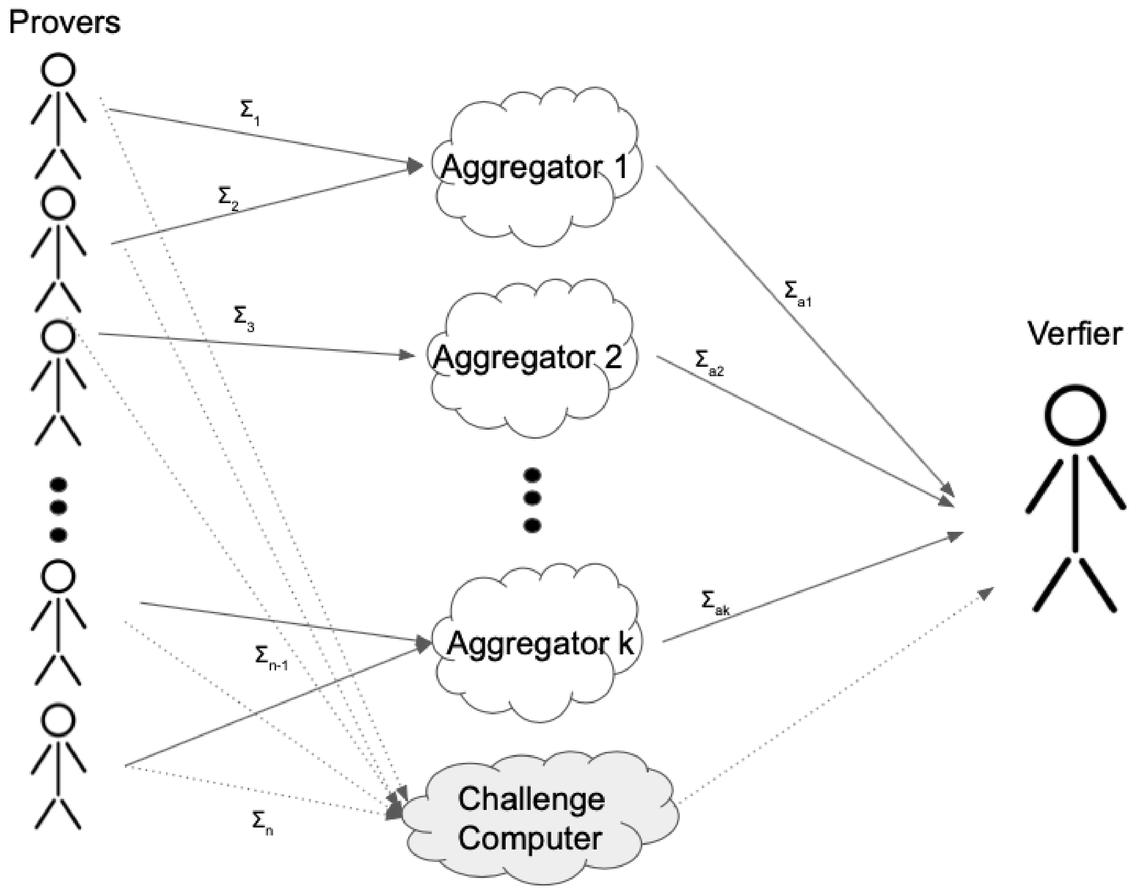

Figure 4.1: Provers outsourcing their set membership proofs to their assigned aggregator that in turn publishes an aggregated set membership proof.

collaborating with the aggregating party (or parties), the set membership proof in Construction 1 is sound and $D_a \neq C^c$. In this section the product of the Pedersen commitments, $\prod_{i\in\mathcal{S}} C_i$, is denoted C and the product of the challenges, $\prod_{i\in\mathcal{S}} c_i$, is denoted c.

If $D_a = C^c$, it would be possible for an aggregating party to choose the values D_a, z_{x_a}, z_{R_a} according to: $D_a = C^c, z_{x_a} = \phi(p), z_{R_a} = \phi(p)$. For the above choice of D_a, z_{x_a}, z_{R_a} the equation, $D_a \overset{?}{=} C^c h^{z_{R_a}} g^{z_{x_a}}$ holds trivially true, independent of whether the commitment C_i and the values z_{x_i}, thereby z_{x_a}, hides the same secret for all $i \in \mathcal{S}$. Therefore it is required that $D_a \neq C^c$.

Under the assumption discussed above, Theorem 3 states that the aggregation must be performed according to the algorithm **Aggregate** for the algorithm **Verify** to validate true, in Construction 2.

Theorem 3 (**Soundness of aggregation**). *Let $\mathcal{A}$ be a PPT adversary. Assume $\mathcal{A}$ has access to the input to the algorithm **Aggregate** in Construction 2 and the Pedersen commitment, $\{C_i\}_{i\in\mathcal{S}}$, but no other information. Assume that the adversary $\mathcal{A}$ cannot collaborate with the provers, that the provers cannot collaborate with each other and that the set membership proof in Construction 1 is sound.*

*Under these assumptions the adversary $\mathcal{A}$ has a negligible probability of construction Σ_a such that: Prob(**Verify**$(pp, \Sigma_a, \{C_i\}_{i\in\mathcal{S}}, \{c_i\}_{i\in\mathcal{S}})) = 1$, where $D_a \neq C^c$ and $z_{x_a} \neq \sum_{i\in\mathcal{S}} \left(\prod_{j\in\mathcal{S}, j\neq i} c_j\right) z_{x_i}$.*

Proof. The soundness of signature-based set membership proofs and that the algorithm **Verify** checks that $a_i \overset{?}{=} e(V_i, y)^{c_i} e(V_i, g)^{-z_{x_i}} e(g, g)^{z_{\tau_i}}$, for all $i \in \mathcal{S}$, implies that the values $V_i, a_i, z_{x_i}, z_{\tau_i}$ of Σ_i must be computed according to the algorithm **Prove** in Construction 2. The values z_{R_i} and D_i are not used directly in the verification, thus the only requirements are that $z_{R_i} \in \mathbb{F}$ and $D_i \in \mathbb{G}$, for all $i \in \mathcal{S}$. Assume, without loss of generality that $D_a = g^\alpha h^\beta$, where $\alpha, \beta \in \mathbb{F}$.

Assume that the adversary $\mathcal{A}$, can construct a valid proof Σ_a, such that: $D_a \in \mathbb{G}$, $z_{x_a}, z_{R_a} \in \mathbb{F}$ and $z_{x_a} \neq \sum_{i\in\mathcal{S}} \left(\prod_{j\in\mathcal{S}, j\neq i} c_j\right) z_{x_i}$.

For the aggregated proof to be valid it must hold that $D_a = C^c h^{z_{R_a}} g^{z_{x_a}}$ which implies that the values α, β, z_x, z_R must satisfy:

$$g^\alpha h^\beta = C^c g^{z_{x_a}} h^{z_{R_a}}.$$

The values of C^c cannot be modified by the adversary since it is sent directly from the provers to the verifier. The Pedersen commitments are assumed to be on the form $C_i = g^{x_i} h^{R_i}$ for all $i \in \mathcal{S}$ and the challenges are correctly computed and checked by a trusted party.

Consequently, by expanding the right hand side of the equality it follows that,

$$g^\alpha h^\beta = g^{c\sum_{i\in\mathcal{S}} x_i + z_{x_a}} h^{c\sum_{i\in\mathcal{S}} R_i + z_{R_a}}.$$

If $\alpha \neq c\sum_{i\in\mathcal{S}} x_i + z_{x_a}$ and $\beta \neq c\sum_{i\in\mathcal{S}} R_i + z_{R_a}$, the above equality contradicts to the binding property of the Pedersen commitments. Thereby, the values α, β, z_{x_a} and z_{R_a} must satisfy: $\alpha = c\sum_{i\in\mathcal{S}} x_i + z_a$ and $\beta = c\sum_{i\in\mathcal{S}} R_i + z_{R_a}$.

This can be rewritten as $\alpha - z_{x_a} = c\sum_{i\in\mathcal{S}} x_i \bmod \Phi(p)$ and $\beta - z_{R_a} = c\sum_{i\in\mathcal{S}} R_i \bmod \Phi(p)$. The sums $\sum_{i\in\mathcal{S}} x_i$ and $\sum_{i\in\mathcal{S}} R_i$ are assumed to be unknown.

Thereby, under the assumption that $z_{x_a} \neq \sum_{i\in\mathcal{S}} \Big(\prod_{j\in\mathcal{S}, j\neq i} c_j \Big) z_{x_i}$ the adversary has a negligible probability of choosing $\alpha, \beta, z_{x_a}, z_{R_a}$ such that $D_a = C^c h^{z_{R_a}} g^{z_{x_a}}$. It follows that $z_{x_a} = \sum_{i\in\mathcal{S}} \Big(\prod_{j\in\mathcal{S}, j\neq i} c_j \Big) z_{x_i}$, which contradicts the assumption and proves the theorem.

□

Note that the theorem would still hold even if several untrusted parties aggregated subsets of the proofs and a verifier validated all the aggregated proofs.

4.3 Completeness, Soundness and Zero-Knowledge

In this section, it is proved that Construction 2 fulfils the completeness, soundness and zero-knowledge requirements stated in chapter 3. This is proved under the assumption that the aggregation is performed according to the algorithm **Aggregate**, provers cannot communicate or collaborate with each other and that Construction 1 satisfies the requirement stated in Definition 4. The requirements also hold for Construction 5, considering $|\mathcal{K}| > 1$.

Completeness

To prove the completeness of Construction 2, it has to be proved that
Verify(**Aggregate**($\{$**Prove**$(pp, i, C_i, x_i, \Phi)\}_{i\in\mathcal{S}}$)) $= 1$. To prove this, let Σ_i for $i \in \mathcal{S}$ denote proofs constructed by the algorithm **Prove** and Σ_a denote the aggregation of these proofs obtained according to algorithm **Aggregate**. For all $i \in \mathcal{S}$, let C_i denote the Pedersen commitment of the secrets x_i and c_i denote the challenge used in the proof.

Then by the completeness property of the signature-based set membership proof [5], it holds that:

$$a_i = e(V_i, y)^{c_i} e(V, g)^{z_{x_i}} e(g, g)^{z_{\tau_i}} \;\; \forall\, i \in \mathcal{S}.$$

It remains to argue that $D_a = C^c h^{z_{R_a}} g^{z_{x_a}}$, where $C = \prod_{i\in\mathcal{S}} C_i$ and $c = \prod_{i\in\mathcal{S}} c_i$.

By construction it holds that:

$$
\begin{aligned}
D_a =& g^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i} h^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i},\\
C^c h^{z_{R_a}} g^{z_{x_a}} =& \left(\prod_{i\in\mathcal{S}} C_i\right)^{\prod_{i\in\mathcal{S}} c_i} h^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i} g^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i-\left(\prod_{j\in\mathcal{S}} c_j\right)\sum_{i\in\mathcal{S}} x_i}\\
=&\left(g^{\sum_{i\in\mathcal{S}} x_i} h^{\sum_{i\in\mathcal{S}} R_i}\right)^{\prod_{i\in\mathcal{S}} c_i} h^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i-\left(\prod_{j\in\mathcal{S}} c_j\right)\sum_{i\in\mathcal{S}} R_i}\\
& g^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i-\left(\prod_{j\in\mathcal{S}} c_j\right)\sum_{i\in\mathcal{S}} x_i}\\
=& g^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)s_i} h^{\sum_{i\in\mathcal{S}}\left(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\right)m_i},\\
\implies D_a =& C^c h^{z_{R_a}} g^{z_{x_a}}
\end{aligned}
$$

Combining the above results it has been shown that $D_a = C^c h^{z_{R_a}} g^{z_{x_a}} \wedge a_i = e(V_i, y)_i^c e(V_i, g)^{-z_{x_i}} e(g, g)^{z_{\tau_i}}$ for all $i \in \mathcal{S}$. Implying that **Verify**(**Aggregate**($\{$**Prove**($pp, i, C_i, x_i, \Phi)\}_{i\in\mathcal{S}})) = 1$.

Zero-Knowledge

The zero-knowledge property of Construction 2 follows from the zero-knowledge property of Construction 1 and by realising that the algorithm **Aggregate** does not reveal any information about the secrets x_i for any $i \in \mathcal{S}$. The latter follows from multiplying and adding elements perfectly hiding the secret x_i, with other elements independent of the secret x_i does not reveal any information about the secret.

Soundness

The aggregated set membership proof in Construction 2 satisfies the soundness property stated in chapter 3 under the assumptions that the aggregation is performed according to the algorithm **Aggregate** in Construction 2, that provers cannot communicate or collaborate and that the set membership proof in Construction 1 satisfies the soundness in Definition 4. To prove this it has to be shown that if for any $i \in \mathcal{S}$ the secret $x_i \notin \Phi$ then it holds that
Prob[**Verify**(**Aggregate**($\{$**Prove**($pp, i, C_i, x_i, \Phi)\}_{i\in\mathcal{S}})) = 1] < \varepsilon$, for some negligible ε.

Let T denote the index-set of the malicious provers. Assume that all honest provers p_i, such that $i \in \mathcal{S}\backslash T$, computes their set membership proofs, Σ_i, according to **Prove**.

If the algorithm **Verify** validates true then $a_i = e(V_i, y)^{c_i} e(V, g)^{z_{x_i}} e(g, g)^{z_{\tau_i}}$ for all $i \in \mathcal{S}$. Thereby the soundness assumption of the set membership proofs in Construction 1 implies that the values $a_i, V_i, z_{x_i}, z_{\tau_i}$ must be computed according to the algorithm **Prove** in Construction 2. Implying that the malicious provers, p_i for

$i \in T$, must construct their set membership proofs, Σ_i, and Pedersen commitments, C_i, such that the following is fulfilled:

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

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

For the algorithm **Verify** to validate true it has to hold that $D_a = C^c h^{z_{R_a}} g^{z_{x_a}}$, where D_a, z_{R_a}, z_{x_a} is the aggregation of Σ_i for all $i \in \mathcal{S}$ according to equation (4.1). By expanding the equality it follows that:

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

Both above equations can be interpreted as 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 the exponents of g and h are equal for the two above equations. Consider the exponent of g this leads to:

$$\sum_{i\in T}\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\Big)\tilde{s}_j = \prod_{j\in\mathcal{S}} c_j\Big(\sum_{i\in T}\tilde{x}_i + \sum_{i\in T} x_i\Big) + \sum_{i\in T}\Big(\prod_{\substack{j\in\mathcal{S}\\ j\neq i}} c_j\Big)s_i. \tag{4.2}$$

It remains to argue that the equality in equation (4.2) cannot hold unless $\tilde{x}_i = x_i$ for all $i \in \mathcal{S}$.

First consider the case when the set T only consists of one element, implying that there is only one malicious prover. Without loss of generality assume that this is the kth prover, p_k. Under this assumption equation (4.2) can be rewritten as,

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

If it would be possible to choose $\tilde{s}_k = c_k\Big(\tilde{x}_k + x_k\Big) + s_k$, it would contradict the soundness assumption of Construction 1. Thereby if $|T| = 1$, it must hold that $\tilde{x}_k = x_k$.

Assume $|T| > 1$ and $k \in T$ such that p_k is a malicious prover. Under the assumption that the provers cannot communicate or collaborate, the proofs $\{\Sigma_i\}_{i\in\mathcal{S}, i\neq k}$, can be considered random values for the prover p_k. Equation (4.2) can be rewritten as:

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

If $\tilde{x}_k \neq x_k$, this would imply that it is possible to cheat in Construction 1 by adding random values to D_i and z_{x_i}. This is a contradiction to the soundness assumption of the set membership proof in Construction 1. This implies that $\tilde{x}_k = x_k$.

Thereby, it is proved that if for any $i \in \mathcal{S}$ the secret $x_i \notin \Phi$, it holds that $\text{Prob}[\textbf{Verify}(\textbf{Aggregate}(\{\textbf{Prove}(pp, i, C_i, x_i, \Phi)\}_{i\in\mathcal{S}})) = 1] < \varepsilon$, for a negligible ε.

Theorem 4 (Completeness, Zero-Knowledge and Soundness). *Assume that the aggregated proof Σ_a is computed according to algorithm* ***Aggregate*** *in Construction 2, that the parties constructing the proofs, $\{\Sigma_i\}_{i\in\mathcal{S}}$, cannot communicate and that the set membership proof in Construction 1 satisfies the requirements stated in Definition 4. Then the aggregated signature-based set membership proof in Construction 2 satisfies the completeness, zero-knowledge and soundness requirements for aggregated set membership proofs, stated in chapter 3.*

Proof. The proof follows from the arguments given above. □

5 Implementation and Evaluation

The construction of aggregated signature-based set membership proofs derived in chapter 4 is in this chapter implemented and evaluated in terms of runtime. The construction is compared with itself for different settings in section 5.2 and in section 5.3 it is compared to the state-of-the-art Bulletproofs.

5.1 Implementation

A prototype of the aggregated signature-based set membership proofs is implemented in Golang. The implementation is based on the code for the signature-based set membership [12], and is available on GitHub [9]. Both the construction considering one aggregating party and the construction considering multiple aggregating parties are implemented in Golang.

The purpose of the implementation is to test the above-proposed constructions and provide runtime evaluations, consequently the code should not be considered as a secure implementation.

Implementation parameters

The parameter settings used for the implementation are defined below.

The number of provers is fixed to 100, unless otherwise stated, and the verifier is assumed to be a single party. The number of aggregating parties, $|\mathcal{K}|$, varies between $|\mathcal{K}| = 1, 5, 10$ or 20. The set of proofs are split evenly between all aggregating parties implying that $|\mathcal{S}_k|$ is equal for all $k \in \mathcal{K}$.

The set Φ consists of 182 elements belonging to the interval $[0, 1000]$.

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

The hardware used for benchmarking, throughout the entire paper, has 1.6 GHz Dual-Core Intel Core i5 − 5250U CPU, 8GB 1600 MHz DDR3 RAM and runs macOS 10.15.

Table 5.1: Timing in seconds for algorithms **Aggregate** and **Verify** in Construction 5 considering different numbers of aggregating parties, $|\mathcal{K}|$.

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

5.2 Trade-off between Aggregation and Verification

The aggregation of set membership proofs is computationally heavy. To reduce the computation required by a single aggregating party the aggregation can be split between several aggregating parties as in Construction 5 in Appendix B. If the set $\mathcal{K}$ in Construction 5 consists of one single element and $\mathcal{S}_k = \mathcal{S}$, then constructions 2 and 5 are equivalent. Hence this section will consider Construction 5 for various number of aggregating parties.

Table 5.1 presents the trade-off between increased verification time due to validating multiple aggregated proofs and reduced runtime of an individual aggregating party obtained by splitting the aggregation between several parties. The number of aggregating parties varies between $1, 2, 5$ and 10, while the number of proofs is fixed to 100. An individual aggregating party thus has to aggregate $100, 50, 20$ or 10 proofs respectively and the verifier has to verify $1, 2, 5$ or 10 aggregated proofs respectively. In Table 5.1 the runtime for the algorithm **Aggregate** is given per aggregating party and the runtime for **Verify** is for verification of all aggregated proofs.

It is noted that the aggregation is a computationally demanding procedure. Considering one aggregating party, the runtime required to aggregate 100 proofs is almost one minute. While the reduction in runtime for verifying the aggregated proof instead of verifying all 100 proofs separately is just above one second. This illustrates that aggregation does not reduce the total runtime for the entire construction, but rather move computations from one party to another and thereby allows offloading computations from the verifier.

Figure 5.1, which is a visualisation of the values in Table 5.1, shows that if the aggregation is shared between a set of aggregating parties the runtime per aggregating party is reduced almost exponentially. At the same time the increased runtime for verifying multiple aggregated proofs is linear. Resulting in that splitting the aggregation between a set of aggregating parties reduced the total runtime for the construction and per aggregating party.

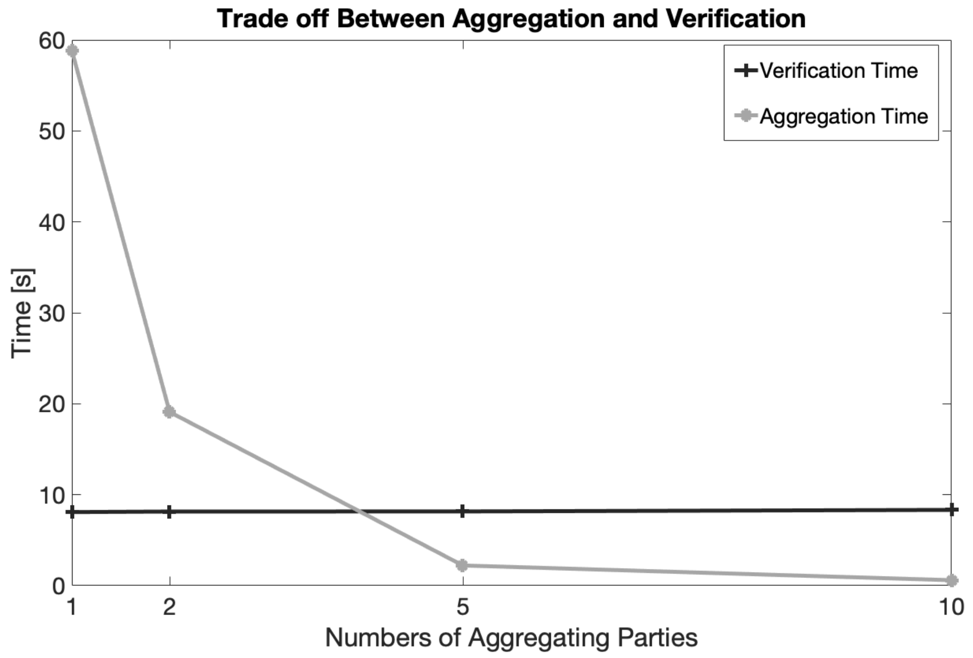

Figure 5.1: Timing in seconds for running the algorithms **Aggregate** and **Verify** in Construction 5 considering different numbers of aggregating parties. The number of aggregating parties varies between $1, 2, 5$ and 10. The figure is a visualisation of the results presented in Table 5.1.

5.3 Comparison to Bulletproofs

In this section, the construction of aggregated signature-based set membership proofs are compared to the state-of-the-art Bulletproofs. The focus of comparison is the runtime of a party verifying of multiple proving parties. In this section the aggregation is considered to be performed by a single party, i.e $|\mathcal{K}| = 1$. Before presenting the results, Bulletproofs are described in more detail.

Bulleproofs settings

The original paper about Bulletproofs [4] presents a method for aggregating Bulletproofs such that n parties, each having committed to a Pedersen commitment, 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, since it is required that all provers construct their proofs using for the same challenges. If the provers were to use different challenges the verification would fail.

To Conclude, Bulletproofs can be aggregated with the cost of an interactive construction. Since this paper aims to investigate non-interactive constructions, non aggregated Bulletproofs are considered. Therefore the verifier has to verify all Bulletproofs separately, i.e once for each prover.

The computational complexity for verification of a Bulletproof depends on the maximum upper bound of the range, i.e it depends on n determining the range $[0, 2^n)$. This motivates to see how the runtime is affected by considering different

Table 5.2: Runtime for verification of 100 proofs constructed using three different ZKP. The three considered proof constructions are: Bulletproofs, with a maximal upper bound equal to 2^8 and 2^{32}, and aggregated signature-based set membership proofs.

	Verify [s]
Bulletproofs, $n = 8$	2.98
Bulletproofs, $n = 32$	10.22
Aggregated Set Membership Proofs	8.09

maximal 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 are considered for comparison $n = 8$ and $n = 32$.

Bulletproofs can be modified to allow arbitrary ranges $[a, b]$, such that $b < 2^n$ and $a < b$, with the same approach as presented for the signature-based range proofs and illustrated in Figure 2.1. This is exploited and the range is set to $[18, 200]$.

Runtime Comparison

This section will compare runtime results considering the verification of 100 proofs using Bulletproofs and aggregated signature-based set membership proofs. Bulletproofs with an upper bound equal to 2^8 and 2^{32} are considered meaning that in total three constructions are compared.

A remark is that the computational complexity of Bulletproofs depend on the variable n, where n determines the maximal upper bound of the range. The signature-based set membership proofs have a computational complexity of $\mathcal{O}(1)$ for the proof construction and verification, meaning that the complexity is independent of the size of the set Φ. Therefore, to provide a comparison between Bulletproofs and aggregated signature-based set membership proofs considering different applications scenarios, the runtime for verification of Bulletproofs is presented considering two different values of n.

Table 5.2 shows the runtime for the verification of 100 proofs using the three considered constructions.

Bulletproofs with an upper bound equal to 2^8 is considerably faster than the other constructions, however, it is highly limited in its applications since the upper bound is fairly low. The runtime for Bulletproof with an upper bound equal to 2^{32} is longer than for aggregated set membership proofs. Thus, aggregated signature-based set membership proofs is a relatively fast and yet flexible implementation.

In Figure 5.2 the runtime for verification is given as a function of the number of provers. In addition to the above constructions, signature-based set membership proofs are considered. The runtime has been measured considering $1, 25, 50, 75, 100, 125$ and 150 provers respectively. From the figure, it is seen that there is a linear relationship between the number of provers and the runtime for verification. It is also seen that the runtime relationship between the different constructions seen in Table 5.2, appears to remain independent of the number of provers.

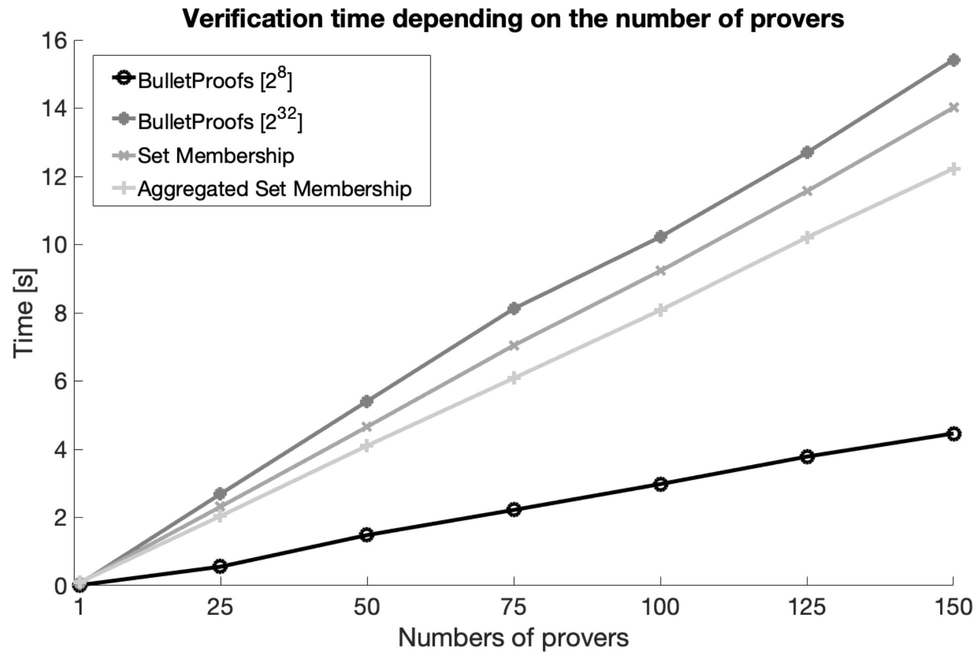

Figure 5.2: Runtime for verification of multiple provers as a function of the number of provers. Verification of four different constructions of proofs are considered: aggregated signature-based set membership proofs, signature-based set membership proofs, Bulletproofs with a maximal upper bound equal to 2^8 and Bulletproofs with a maximal upper bound equal to 2^{32}.

In Figure 5.2 it is noted that the difference between the runtime for the aggregated signature-based set membership proof and the ordinary signature-based set membership proofs, increases with the number of provers. This result is expected since the difference in the number of computation performed by the verifier increases linearly with the number of provers.

To conclude, whether Bulletproofs or aggregated signature-based set membership proofs are faster in terms of runtime for the verifier depends on the application, since the runtime for verifying Bulletproofs depends on the maximal upper bound of the range. The rule of thumb is that aggregated signature-based set membership proofs are faster when considering a large range and Bulletproofs are faster when considering a small range. It is however always true that set membership proofs are more flexible than range proofs since they allow non-continuous discrete sets. An important thing to mention, although not the focus of this comparison, is that the set-up phase of aggregated signature-based set membership proofs is linear in the number of elements in the set, resulting in that it is computationally demanding for large sets.

6
Application in VAHSS

In this chapter, an application of the aggregated set membership proof is presented and evaluated. As discussed in chapter 1, VAHSS is an example where multiple data providers, henceforth denoted clients, participate. In this chapter first a construction of VAHSS that verifies the servers computation is presented, then it is derived how to extend the verification to also include the clients. Then different methods for verifying clients are compared. The comparison mainly focuses on comparing aggregated signature-based set membership proof presented in Construction 2 with the state-of-the-art Bulletproofs for verification of clients to see how the runtime of the construction is affected.

6.1 VAHSS

The considered construction of VAHSS presented in [18], implemented and benchmarked in [17], makes use of homomorphic hash functions to verify the computations performed by servers.

Assume that n clients and m servers participate in the VAHSS construction. The sets $\mathcal{N} = \{1, ..., n\}$ and $\mathcal{M} = \{1, ..., m\}$ are introduced to simplify notation. The clients are denoted c_i for $i \in \mathcal{N}$, and their respective data x_i. The servers in the construction are refereed to s_j for $j \in \mathcal{M}$.

Each client c_i splits the secret x_i into m shares, denoted $\{x_{ij}\}_{j \in \mathcal{M}}$, such that $x_i = \sum_{j \in \mathcal{M}} x_{ij}$ and sends one share to each server. Each server s_j, $j \in \mathcal{M}$, receives shares from all n clients and computes and publishes the partial sum $y_j = \sum_{i \in \mathcal{N}} x_{ij}$. Then the final sum can be computed by any party by summing the public partial sums, which gives $y = \sum_{j \in \mathcal{M}} y_j = \sum_{j \in \mathcal{M}} \sum_{i \in \mathcal{N}} x_{ij} = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} x_{ij} = \sum_{i \in \mathcal{N}} x_i$.

A 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 it holds the servers computations are proved to be correct. For a precise implementation and proof of correctness, security and verification the reader is refereed to the original paper about VAHSS [18].

6.2 Client and Server VAHSS

The VAHSS construction discussed in section 6.1 assumes honest clients and verifies the servers. This section extends the VAHSS construction to verify both the clients' input and the computations performed by the servers. First it is stated how to extend the VAHSS construction to additionally verify clients. Then a construction, using range proofs or set membership proofs, to verify clients is derived. It is then discussed how such a construction can be modified to use aggregated set membership proofs for verification of client' inputs.

Extending VAHSS to verify clients

To ensure honest clients it is not sufficient to construct and perform a set membership proof and a 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 principle holds considering range proofs. Therefore, a connection between the shares generated in the algorithm **ShareSecret** in the VAHSS construction and the secret hidden in a Pedersen commitment is desired. Proving that the sum of the shares is equal to the secret in a Pedersen commitment and then proving that the secret in the Pedersen commitment is in an allowed range or set convinces the verifier that the shares represent a secret that is in the allowed range or set.

In the VAHSS construction the clients publishes, in addition to the shares, the checksums τ_i for the secrets x_i. Recall that the definition of the checksum is $\tau_i = g^{x_i+R_i}$. The checksum τ_i can be interpreted as a Pedersen commitment, where $g = h$.

Hereafter, the clients compute and outputs the Pedersen commitments $\pi_i = g^{x_i}h^{R_i}$, instead of the previously computed checksums τ_i. Given a commitment π_i, the clients can construct a range proof or set membership proof of the committed secret. It remains to argue that this would ensure the verifier that the secret hidden by the shares is the same secret proved to be in the allowed range or set as for all $i \in \mathcal{N}$.

Assume that client c_k commits to the value $\hat{x}_k$ in the Pedersen commitment π_k, constructs the shares $\{x_{kj}\}_{j\in\mathcal{M}}$ such that $x_k = \sum_{j\in\mathcal{M}} x_{kj} \neq \hat{x}_k$ and generates a ZKP that $\hat{x}_k$ belongs to the set Φ or range $[a, b]$. Since $x_k \neq \hat{x}_k$ this does not necessarily imply that the secret hidden by the shares belongs to the range or set. Then $\prod_{i=1}^{m} \pi_i \neq g^y$ and thereby the verification of servers does not succeed. Thus it has to hold that $x_k = \hat{x}_k$ for the protocol to succeed and any cheating client is detected. It is not possible to determine which party cheated and more precisely not even if the cheating party was a client or a server.

Remark 1. *The correctness, security and verifiability requirements of a VAHSS construction holds after replacing the checksums $\tau_i = g^{x_i+R_i}$ with a Pedersen commitment $\pi_i = g^{x_i}h^{R_i}$ in the VAHSS construction, [18].*

Additionally if a range proof or set membership proof, denoted Σ_i, that satisfies the soundness, completeness and zero-knowledge requirements of zero-knowledge proofs is constructed of the secret x_i in the Pedersen commitment π_i, for all clients

c_i in the VAHSS construction. Then any PPT adversary $\mathcal{A}$ who can modify the Pedersen commitments π_i to any $\pi_i^{'}$ $\forall i \in T$, where T is the set of corrupted clients, has a negligible probability of choosing a commitment $\pi_i^{'}$ such that verification of all the proofs Σ_i and the VAHSS validates true.

Proof of remark: The remark follows from that replacing τ_i with π_i for $i = 1, ..., n$, it still holds that:

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

Thereby it follows that: $\prod_{i \in \mathcal{N}}^{n} \tau_i = \prod_{i \in \mathcal{N}} \pi_i.$

Further the Pedersen commitment is perfectly hiding as well as computationally binding and thus it follows that the requirements are still fulfilled.

From the soundness of range proofs and set membership proofs, and the above argument that the secret hidden in the commitment π_i must be the same as the secret obtained by combining the the shares $\{x_{ij}\}_{j \in \mathcal{M}}$ for all $i = \mathcal{N}$, it follows that any adversary who can modify the commitments π_i has a negligible probability of doing so such that the proofs Σ_i and the VAHSS construction validates true. $\square$

Verifying clients using set membership proofs or range proofs

A VAHSS where clients are verified by publishing range proofs or set membership proofs is presented in Construction 3. In order to clarify the changes made to extend the construction to verify clients, the differences to the VAHSS construction presented in [18] are pointed out.

The algorithms **ShareSecret** and **Verify** has been modified, and the algorithms **ProveSecret** and **GenerateCommitment** have been added. More precisely, the algorithm **ShareSecret** does not output the checksum τ_i, instead the Pedersen commitment π_i is computed in the algorithm **GenerateCommitment**. The algorithm **ProveSecret** constructs a set membership proof or range proof denoted Σ_i given the commitment π_i. In addition to the steps of the algorithm **Verify** in the VAHSS construction presented in [18], the algorithm **Verify** also validates the proofs Σ_i for all $i \in \mathcal{N}$.

The algorithms **GenerateCommitment** and **ProveSecret** are executed by the clients and the other algorithms are executed by the same party as in the VAHSS construction in [18].

Remark 1 implies that Construction 3 satisfies the correctness, security and verification requirements for a VAHSS construction and that the verification of clients' input satisfies the completeness, soundness and zero-knowledge requirements of the considered range proofs or set membership proofs.

Verifying clients using aggregated set membership proofs

Construction 4 describes a client and server VAHSS compatible with aggregated set membership proofs for verification of clients. The algorithms in Construction 4

Construction 3 : Client and Server Verifiable additive homomorphic secret sharing

Goal: Compute the sum $y = \sum_{i=1}^{n} x_i$. The values x_i are kept secret. The servers computations and the clients shared values are verified.

- **ShareSecret** $(1^\lambda, i, x_i) \mapsto \{x_{ij}\}_{j\in\mathcal{M}}$
  Pick uniformly at random the coefficients, $\{a_i\}_{i\in\{1,..,t\}} \in_R \mathbb{F}$ and define a t-degree polynomial p_i to be on the form $p_i(X) = x_i + a_1X + ... + a_tX^t$. Put the shares $x_{ij} = \lambda_{ij}p_i(\theta_{ij})$ for $j \in \mathcal{M}$. The parameters θ_{ij} and Lagrange coefficients λ_{ij} are chosen such that $p_i(0) = \sum_{j=1}^{m} \lambda_{ij}p_i(\theta_{ij})$. Output $\{x_{ij}\}_{j\in\mathcal{M}}$.
- **GenereteCommitment**$(1^\lambda, i, x_i) \mapsto \pi_i$
  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 $\pi_i = \mathbb{E}(x_i, R_i) = g^{x_i}h^{R_i}$.
- **ProveSecret** $(pp, x_i, \pi_i) \mapsto \Sigma_i$
  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 $[a, b]$ or a set Φ. All required public parameters, pp, needed to construct the proof Σ_i is assumed to be pre-shared and known by all parties.
- **PartialEval** $(j, \{x_{ij}\}_{i\in\mathcal{N}}) \rightarrow y_j$
  Compute and output $y_j = \sum_{i=1}^{n} x_{ij}$.
- **PartialProof** $(j, \{x_{ij}\}_{i\in\mathcal{N}}) \rightarrow \sigma_j$
  Compute and output $\sigma_j = \prod_{i=1}^{n} g^{x_{ij}} = g^{\sum_{i=1}^{n} x_{ij}} = g^{y_j} = \mathcal{H}_1(y_j)$.
- **FinalEval** $(\{y_j\}_{j\in\mathcal{M}}) \rightarrow y$
  Compute and output $y = \sum_{j=1}^{m} y_j$.
- **FinalProof** $(\{\sigma_j\}_{j\in\mathcal{M}}) \rightarrow \sigma$
  Compute and output $\sigma = \prod_{j=1}^{m} \sigma_j = \prod_{j=1}^{m} g^{y_j} = g^{\sum_{j=1}^{m} y_j} = g^y = \mathcal{H}_1(y)$.
- **Verify** $(\{\pi_i\}_{i\in\mathcal{N}}, x, y, \{\Sigma_i\}_{i\in\mathcal{N}}) \rightarrow \{0, 1\}$
  Compute and output $\sigma = \prod_{i=1}^{n} \pi_i \wedge \prod_{i=1}^{n} \pi_i = \mathcal{H}_1(y) \wedge \{\textbf{VerifyProof}(\Sigma_i)\}_{i\in\mathcal{N}}$. **VerifyProof** is the verification algorithm of the proofs $\{\Sigma_i\}_{i\in\mathcal{N}}$.

are the same as in Construction 3 except that the algorithm **PartialAggregate** is introduced and the algorithm **Verify** is modified.

The algorithm **PartialAggregate** is run by all aggregating parties in the set $\mathcal{K}$ and the algorithm **Verify** is modified such that it verifies the aggregated proofs instead of the individual clients' proofs.

If the server computing the partial sums y_j are responsible for aggregation of the clients set membership proofs, then no new parties are introduced to the VAHSS construction to aggregate the proofs. Then the set $\mathcal{K}$ is equal to $\mathcal{M}$ in construction 4.

For aggregation in Construction 4 to be sound, the aggregation must either be performed by a trusted party or split between at least two aggregating parties.

If the aggregation is performed by an untrusted party, this party can cheat in the aggregation of the proofs, by exploiting that the sum y can be computed once the servers have performed the algorithm **PartialEval** and that $h^{\sum_{i\in\mathcal{N}} R_i} = 1 \bmod \phi(p)$. Note that this induces that the assumptions of Theorem 3 are not fulfilled, since the aggregation party has additional knowledge beyond the input to the algorithm **Aggregate** and the commitments $\{C_i\}_{i\in\mathcal{S}}$. Since y is known to the aggregating party, the aggregated proof Σ_a can be constructed as, $D_a = C^{k+c}$, $z_{R_a} = k\phi(N)$ and $z_{x_a} = ky$, where $y = \sum_{i\in\mathcal{N}} x_i$ and $k \in_R \mathbb{F}$. Then it follows that:

$$\begin{aligned}
D_a = C^{k+c} &= \Big(g^{k\sum_{i=1}^n x_i} h^{k\sum_{i=1}^n R_i}\Big)\Big(g^{(\prod_{i=1}^n c_i)\sum_{i=1}^n x_i} h^{(\prod_{i=1}^n c_i)\sum_{i=1}^n R_i}\Big) \\
C^c h^{z_{R_a}} g^{z_{x_a}} = C^c h^{k\phi(N)} g^{ky} &= \Big(g^{(\prod_{i=1}^n c_i)\sum_{i=1}^n x_i} h^{(\prod_{i=1}^n c_i)\sum_{i=1}^n R_i}\Big) h^{k\phi(N)} g^{ky} \\
&= \Big(g^{(\prod_{i=1}^n c_i)\sum_{i=1}^n x_i} h^{(\prod_{i=1}^n c_i)\sum_{i=1}^n R_i}\Big) g^{ky} h^{k\phi(N)\lceil \frac{\sum_{i=1}^{n-1} R_i}{\phi(N)} \rceil)} \\
&\implies D_a = C^c h^{z_{R_a}} g^{z_{x_a}}.
\end{aligned}$$

It has been shown that if one party aggregating all clients' set membership proofs the aggregation is not sound, since the aggregated proof can be constructed such that it validates true without proving the statement in equation 3.1.

If multiple parties aggregates subsets of the proofs, the assumptions in Theorem 3 holds, this follows from that the sum of the secrets and random values are unknown considering any true subset of the clients.

Under the assumption that the aggregation is sound, Remark 1 applies to Construction 4. Thereby, if the aggregation is performed by a trusted party or is split between at least two independent aggregating parties, Construction 4 satisfies the correctness, security and verification requirements for a VAHSS construction and the verification of clients' input satisfies the completeness, soundness and zero-knowledge requirements of the considered range proofs or set membership proofs.

6.3 Implementation

An implementation of Constructions 3 and 4 is obtained to investigate the proposed clients and server VAHSS in a practical setting and comparing the runtime for different methods for verifying clients.

Bulletproofs, aggregated and non-aggregated signature-based set membership proofs implemented in Golang are publically available on Github, [12] [9]. The

Construction 4 : Client and Server Verifiable additive homomorphic secret sharing

Goal: Compute the sum $y = \sum_{i=1}^{n} x_i$. The values x_i are kept secret. Servers computations and clients shared values are verified.

- **ShareSecret** $(1^\lambda, i, x_i) \mapsto \{x_{ij}\}_{j \in \mathcal{M}}$
  Pick uniformly at random the coefficients, $\{a_i\}_{i\in\{1,..,t\}} \in_R \mathbb{F}$ and define a t-degree polynomial p_i to be on the form $p_i(X) = x_i + a_1 X + ... + a_t X^t$. Put the shares $x_{ij} = \lambda_{ij} p_i(\theta_{ij})$ for $j \in \mathcal{M}$. The parameters θ_{ij} and Lagrange coefficients λ_{ij} are chosen such that $p_i(0) = \sum_{j=1}^{m} \lambda_{ij} p_i(\theta_{ij})$. Output $\{x_{ij}\}_{j\in\mathcal{M}}$.
- **GenereteCommitment**$(1^\lambda, i, x_i) \mapsto \pi_i$
  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 $\pi_i = \mathbb{E}(x_i, R_i) = g^{x_i} h^{R_i}$.
- **ProveSecret** $(pp, x_i, \pi_i) \mapsto \Sigma_i$
  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 $[a, b]$ or a set Φ. All required public parameters, pp, needed to construct 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 $\sigma_j = \prod_{i=1}^{n} g^{x_{ij}} = g^{\sum_{i=1}^{n} x_{ij}} = g^{y_j} = \mathcal{H}_1(y_j)$.
- **PartialAggregate** $(pp, k, \mathcal{S}_k, \{\Sigma_i\}_{i\in\mathcal{S}_k}) \to \Sigma_{a_k}$
  On the input $\{\Sigma_i\}_{i\in\mathcal{S}_k}$ where $\mathcal{S}_k \subseteq \{1, ..., n\}$, the set of proofs is aggregated according to the algorithm **Aggregate** in Construction 2 and aggregated proof Σ_{a_k} is published.
- **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** $(\{\pi_i\}_{i\in\mathcal{N}}, x, y, \{\Sigma_{a_k}\}_{k\in\mathcal{K}}) \to \{0, 1\}$
  Compute and output $\sigma = \prod_{i=1}^{n} \pi_i \wedge \prod_{i=1}^{n} \pi_i = H(y) \wedge \{\textbf{VerifyProof}(\Sigma_{a_k})\}_{k\in\mathcal{K}}$. **VerifyProof** is the verification algorithm in Construction 2.

VAHSS construction implemented in both python and C++, is available at [16] and [15] respectively.

To implement Construction 3 and 4, the VAHSS algorithms need to be callable from the same programs as the algorithms for Bulletproofs, signature-based set membership proofs and aggregated signature-based set membership proofs. The VAHSS algorithms have been translated to Golang, to solve the problem of having the implementations written in different programming languages. The implementation is available at [9].

To provide an implementation of Construction 3 and 4 besides translating the VAHSS code to Golang the implementations have also been slightly modified. The VAHSS construction has as discussed above been adjusted such that it considers a Pedersen commitment π_i instead of the checksum τ_i. The implementations of Bulletproofs, signature-based set membership proofs and aggregated signature-based set membership proofs have also been adjusted to be compatible with the VAHSS algorithms. These adjustments are merely to merge the constructions and does not change the semantics of the constructions. The modified implementations are available at [9].

Implementation parameters

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

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 $\mathcal{K}$ is assumed to consist of a single party, meaning that $|\mathcal{K}| = 1$.

Remark that the trade-off between runtime for aggregation and verification depending on the number of aggregation parties is presented in section 5.2. The result presented there translates directly to the runtime of the aggregation and the verification of clients in a server and client VAHSS. Thereby, although $|\mathcal{K}| = 1$ in this section the runtime considering multiple aggregating parties can be obtained by studying the results presented in chapter 5.

The range is set to $[18, 200]$ for the implementation of Bulletproofs, and thus the upper bound of the Bulletproof is set to 2^n, where $n = 8$. The size of the set Φ is put to the length of the range, $|\Phi| = 200 - 18 = 182$, for both the aggregated and not aggregated signature-based set membership proofs.

A final remark about the implementation is that its purpose is to test the above-proposed constructions and provide runtime evaluations, the code has not been tested enough to be considered as a secure implementation.

6.4 Prototype analysis

The runtime for the algorithms in Construction 3 and 4 is presented in Table 6.1. Construction 3 is benchmarked considering two different constructions for verifying clients: Bulletproofs and signature-based set membership proofs. Construction 4 is benchmarked considering aggregated signature-based set membership proofs, with a single trusted aggregating party.

In Construction 3 and 4 there is one algorithm called **Verify**, verifying both clients and servers. To separately measure the runtime for verifying the servers and clients the algorithm **Verify** is split into two procedures, **VerifyServers** and **VerifyClients**. The first procedure, **VerifyServers**, performs the verification of the servers computations. The second procedure **VerifyClients**, verifies the clients by evaluating their range proofs or set membership proofs. To clearly state what the algorithms **VerifyServers** and **VerifyClients** correspond to, consider the following reformulation of the algorithm **Verify**:

- **Verify** $(pp, \{\pi_i\}_{i\in\mathcal{N}}, y, \{\Sigma_i\}_{i\in\mathcal{N}}) \to \{0,1\}$
  Verify the clients and servers according to,
  - **VerifyServers** $(\{\pi_i\}_{i\in\mathcal{N}}, y) \to \{0,1\})$
    Compute and output $\sigma = \prod_{i=1}^{n} \pi_i \wedge \prod_{i=1}^{n} \pi_i = H(y)$.
  - **VerifyClients** $(\{\pi_i\}_{i\in\mathcal{N}}\{\Sigma_i\}_{i\in\mathcal{N}}) \to \{0,1\}$
    For each proof Σ_i, verify that it is correct. This implies 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 have been aggregated then the verification is performed for each aggregated proof instead of for each clients proofs. If all proofs are correct return 1, else 0.

  Return **VerifyServers** $\wedge$ **VerifyClients**

Table 6.1: Runtime for the algorithms in Construction 3 and 4. The runtime of Construction 3 is presented using Bulletproofs and signature-based set membership proofs to verify clients. Aggregated signature-based set membership proofs are used to verify clients in Construction 4

	Construction 3		**Construction 4**
	Bulletproofs	Set membership	Aggregated Set membership
GenerateShares	95 [μs]	98 [μs]	98 [μs]
ProveSecret	53 [ms]	66 [ms]	66 [ms]
PartialEval	78 [μs]	71 [μs]	71 [μs]
PartialProof	273 [μs]	5255 [μs]	5255 [μs]
Aggregate			59 [s]
FinalEval	689 [ns]	699 [ns]	699 [ns]
FinalProof	50 [μs]	115 [μs]	115 [μs]
VerifyClients	2979 [ms]	9288 [ms]	8120 [ms]
VerifyServers	1672 [μs]	7947 [μs]	7947 [μs]

In Table 6.1 the runtime for all algorithms are consistently faster when using Bulletproofs to verify the clients. Note that the considered range is $[18, 200]$, hence it is sufficient to use $n = 8$ for the upper bound for the Bulletproofs. In section 5.3 it is noted that the runtime for verification of multiple clients increased significantly if the upper bound of the range is increased. This implies that if a larger range is considered the aggregated signature-based set membership would be faster than Bulletproofs for verification of all clients. This is seen in Chapter 5 in Figure 5.2.

Uniformly, independent of which construction used to verify clients, the runtime for **VerifyServers** is approximately 10^3 times faster than for **VerifyClients**. This highlights how expensive the verification of clients is and motivates the attempt to reduce the computations required to verify multiple clients by aggregating set membership proofs.

Considering the runtime of **VerifyClients**, it is noted that aggregation of signature-based set membership proofs reduce the runtime by 13%. The set membership proofs are only partly aggregated and thereby the runtime depends on the number of provers. Consequently, when a large number of clients participate the verification of clients still becomes a bottleneck of the construction.

It is noted that the algorithms **PartialProof**, **FinalProof** and **VerifyServers** differs noteworthy in runtime for different constructions for verifying clients. These algorithms, as described in Construction 3 and 4, are seemingly independent of the construction used to verify clients. The difference in runtime comes from that different libraries for the elliptic curve groups are used for Bulletproofs and the signature-based set membership proofs. The libraries have not been benchmarked against each other, and whether the difference in runtime can be reduced via optimisations has not been investigated.

7 Discussion and Conclusion

Discussion

The simplest aggregation of set membership proofs is to construct the aggregated proof as the element-wise product of all individual proofs. Appendix C states how such an aggregation can be implemented. It also shows that it results in a construction that does not satisfy the completeness requirement of aggregated set membership proofs, implying that cleverer aggregation is required.

Moreover, it is noted that if the challenges are equal for all proofs, then the completeness requirement is satisfied for the first part of the validation. The challenges depend on randomness in the proofs, thereby it cannot be guaranteed that they are equal. In the aggregated signature-based set membership proof presented in chapter 4 the challenges appear as a product, which resolves the problem of unequal challenges.

The complexity of the presented aggregated signature-based set membership proof depends on the number of provers, since $a_i = e(V_i, y)^{c_i} e(V_i, g)^{-z_{x_i}} e(g, g)^{z_{\tau_i}}$ is checked separately for each proof Σ_i, $i \in \mathcal{S}$, in the verification. Therefore it can be interpreted as a partial aggregation of the proofs. A complete aggregation of the signature-based set membership proofs fulfilling the completeness requirement has not been found nor proved impossible to construct. The reasoning of why a complete aggregation of the signature-based set membership proofs does not fulfil the completeness requirement is given in appendix D. In the verification of the signature-based range proofs derived from the signature-based set membership proofs [5], the equality $a_i = e(V_i, y)^{c} e(V_i, g)^{-z_{x_i}} e(g, g)^{z_{\tau_i}}$ is checked separately for each $j \in \mathbb{Z}_l$. This can be seen as an indicator that it is not possible to efficiently aggregate the entire signature-based set membership proofs.

The presented construction of aggregated signature-based set membership proof can be translated to a construction of aggregated signature-based range proofs. The signature-based range proofs are very similar in construction to the signature-based set membership proofs and thereby the aggregation can easily be adjusted to aggregate the range proofs. Details on how to generalise the aggregation are not given, but it follows directly from inspection.

Using set membership proofs or range proofs to verify clients in a VAHSS construction prevents clients from cheating, but no requirement ensures that clients do not lie. Cheating means that a client shares a value not in the allowed set or range and lying means that the client shares a value in the allowed set or range, but not the truthful value.

Conclusion

This paper has presented a definition of aggregated set membership proofs, including the completeness, soundness, and zero-knowledge requirements that it should fulfil.

According to the definition of aggregated set membership proofs, signature-based set membership proofs have been partly aggregated. Since a part of the proofs is verified separately for each prover, the complexity for verification depends on the number of provers. However, the computational complexity required per prover is decreased, due to that, a part of the proofs is aggregated, such that it is checked once to validate all provers.

It has been proved that an untrusted party must perform the aggregation according to the protocol, for the verification to validate true. The assumptions made to prove this are that: the provers do not collaborate with each other or the aggregating party and that the aggregated proof Σ_a is such that $D_a \neq C^c$. C is the product of all provers Pedersen commitments and c is the product of all challenges.

The completeness, soundness and zero-knowledge requirements for aggregated set membership proofs are proved to hold for the presented construction of aggregated signature-based set membership proofs. This was proved under the assumptions that the aggregation was performed according to the algorithm **Aggregate**, proves does not collaborate and that the signature-based set membership, [5], satisfies the requirements in Definition 4.

The aggregated signature-based set membership proof has been implemented in Golang. Considering 100 provers, each having constructed a signature-based set membership proof, the verification was found to be approximately 13% faster for verifying an aggregated proof, computed according to algorithm **Agrgegate**, compared to verifying the proofs individually. The prototype analysis also showed that splitting the aggregation between several parties decreased the runtime per aggregating party almost exponentially. The verification time, in tandem, does not increase exponentially.

The second part of this paper focused on the verification of clients in VAHSS constructions and comparing different methods for verifying clients. The VAHSS construction, presented in [18] has been modified to additionally verify the clients. Clients are verified by publishing set membership proofs or range proofs. Then the verifier, in addition to validating the servers computations, validates all clients set membership proofs or range proofs. To reduce the computations required by the verifier aggregated set membership proofs are considered for verification of clients.

Implementations in Golang have been provided for the client and server VAHSS, considering Bulletproofs, aggregated and non-aggregated signature-based set membership proofs for verification of clients.

Future work

In this section some question that has raised during the work is mentioned and discussed as possible future work.

- A complete aggregation of a set membership proofs. The presented aggregated signature-based set membership proof is partly aggregated and therefore the computational complexity of verification dependent on the number of provers. Providing a full aggregation of the signature-based set membership proof or alternatively construct a complete aggregation of some other set membership proofs would be two interesting results.

- Constructing a non-interactive aggregation of Bulletproofs. Bulletproofs are state of the art range proofs aggregating them using a non-interactive construction would be useful in many applications. To the knowledge of the author no such construction exists.
- Prio+, [1], constitutes the same purpose as client and server VAHSS. The computations required by the clients in Prio+ is smaller compare to client and server VAHSS, due to that they are not required to construct a ZKP. However, the servers must communicate to verify the clients. An interesting topic would be to investigate if Prio+ can be modified such that the verification of clients can be obtained without communication between the servers and with a computational complexity independent of the number of clients. This would result in a construction highly efficient for both clients and servers.

Bibliography

[1] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. Cryptology ePrint Archive, Report 2021/576, 2021. `https://eprint.iacr.org/2021/576`.

[2] 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.

[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] 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.

[5] 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.

[6] 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.

[7] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 259–282, Boston, MA, Mar. 2017. USENIX Association.

[8] G. Couteau, M. Klooß, H. Lin, and M. Reichle. Efficient range proofs with transparent setup from bounded integer commitments. Cryptology ePrint Archive, Report 2021/540, 2021. `https://eprint.iacr.org/2021/540`.

[9] H. Ek. Aggrgated signature based set membership proofs. `https://github.com/ek-k-hanna/Aggregated-SMP`, 2021.

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

[11] 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.

[12] 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.

[13] 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.

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

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

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

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

[18] 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.

[19] 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
Correctness, Security and Verifiability of VAHSS

Assume that a VAHSS construction consists of n clients and m servers, and that the index set of clients and servers are $\mathcal{N}$ and $\mathcal{M}$ respectively. A construction of VAHSS is a 7-tuple of PPT-algorithms (**Setup**, **ShareSecret**, **PartialEval**, **PartialProof**, **FinalEval**, **FinalProof**, **Verify**). Assuming that the algorithms are as defined in [18] they should satisfy the following correctness, verifiability and security requirements [18]:

- **Correctness** It should always hold that $\Pr[\textbf{Verify}(pp, \sigma, y) = 1] = 1$. pp are the public parameters constructed by the algorithm **Setup**, σ is the output of the algorithm **FinalProof** computed by the partial proofs σ_j, which are outputs of the algorithm **PartialProof** for $\forall j \in \mathcal{M}$. The value y is the output of the algorithm **FinalEval**.
- **Verifiability** For any set T of corrupted servers and any PPT adversary $\mathcal{A}$ it should hold that $\Pr[\textbf{Exp}^{Verify}_{VHSS}(x_1, ..., x_n, T, \mathcal{A}) = 1] \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$. The values $x_1, ..., x_n \in \mathbb{F}$ are secret values and the experiment $\textbf{Exp}^{Verify}_{VHSS}(x_1, ..., x_n, T, \mathcal{A})$ is defined as:
  1. For all $i \in \mathcal{N}$ run the algorithm **ShareSecret** in the input $(1^\lambda, i, \mathbf{x}_i)$ and publish the output τ_i.
  2. All servers $s_j \in T$ gives the shares $(\text{share}_{1j}, ..., \text{share}_{mj})$ to the adversary.
  3. For all corrupted server s_j the adversary outputs the modified values y'_j and σ'_j. While for all servers $s_j \notin T$ the values y'_j and σ'_j are computed according to the algorithms **PartialEval** and **PartialProof** respectively.
  4. The final value y' is computed according to the algorithm **FinalEval** and the final proof σ is computed according to the algorithm **FinalProof**.
  5. If $y \neq f(x_1, ..., x_n)$ and $\textbf{Verify}(pp, \sigma', y') = 1$ output 1 else 0. In this thesis the function f is assumed to be the sum of the arguments.

  The above can be interpreted as: any PPT adversary who controls the shares of the secret inputs for the corrupted servers has a negligible probability of having a wrong value of y being accepted.
- **Security** Denote the set of corrupted servers by T and assume that at least one servers is honest, resulting in that $|T| < m$. A VAHSS scheme is t-secure if $\text{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$, for a negligible $\varepsilon(\lambda)$ and all $T \subset \{s_i, ..., s_m\}$ such that $|T| < t$. $\text{Adv}(1^\lambda, \mathcal{A}, T) := \Pr[b = b'] - 1/2$ is the advantage of the adversary $\mathcal{A} = \{\mathcal{A}_1, \mathcal{D}\}$ in guessing b in the following experiment:
  1. The adversary $\mathcal{A}_1$ gives (i, x_i, x'_i) to the challenger, where $i \in \mathcal{N}$, $x_i \neq x'_i$ and $|x_i| = |x'_i|$.

2. The challenger picks a bit $b \in_R \{0,1\}$, if $b = 0$ then the challenger puts $\hat{\mathbf{x}}_i = x_i$ and if $b = 1$ the challenger puts $\hat{\mathbf{x}}_i = x'_i$. Then the challenger runs the algorithm **ShareSecret** on the input $(1^\lambda, i, \hat{\mathbf{x}}_i)$ and algorithm outputs $(\widehat{\text{share}}_{i1}, ..., \widehat{\text{share}}_{im}, \hat{\tau}_i)$.
3. Given the shares sent to the corrupted servers T and the hash $\hat{\tau}_i$ from the challenger. The adversary distinguisher outputs a guess $b' = \mathcal{D}\Big((\text{share}_{ij})_{j|s_j \in T}, \hat{\tau}_i\Big)$.

B

Multiple Aggregating Parties

Construction 5 : 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 construction proves that all secrets x_i belongs to the set Φ, without revealing anything else about the secrets.

- **SetUp** $(1^\lambda, \Phi) \rightarrow (sk, pp)$
  Let g be a generator of the group $\mathbb{G}$ and h an 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** $(pp, i, C_i, x_i, \Phi) \rightarrow \Sigma_i$
  Pick uniformly at random $\tau_i \in_R \mathbb{F}$, choose from the set $\{A_i\}$ the element A_{x_i} and calculate $V_i = A_{x_i}^{\tau_i}$. Pick uniformly at random three values $s_i, t_i, m_i \in_R \mathbb{F}$. Put $a_i = e(V_i, g)^{-s_i} e(g, g)^{t_i}$, $D = g^{s_i} h^{m_i}$,$c = \text{Hash}(C_i, V_i, a_i, D_i)$ and compute $z_{x_i} = s_i - x_i c_i$, $z_{R_i} = m_i - R_i c_i$ and $z_{\tau_i} = t_i - \tau_i c_i$. Finally construct and publish the proof $\Sigma_i = (V_i, a_i, D_i, z_{x_i}, z_{\tau_i}, z_{R_i})$.
- **Aggregate** $(pp, k, \{\Sigma_i\}_{i \in \mathcal{S}_k}) \rightarrow \Sigma_{a_k}$
  Given a set of 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 (4.1). 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 challenge $c_i = Hash(C_i, V_i, a_i, D_i)$. 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 $D_{a_k} \overset{?}{=} \left(\prod_{i \in \mathcal{S}_k} C_i\right)^{c_k} h^{z_{Ra_k}} g^{z_{xa_k}}$ Then for check if $a_i \overset{?}{=} e(V_i, y)^c_i e(V_i, g)^{-z_{x_i}} e(g, g)^{z_{\tau_i}}$ for all $i \in \mathcal{S}$. If the equalities holds return 1 otherwise return 0.

C
Naive Aggregation

Consider two set membership proofs Σ_1 and Σ_2, computed according to the algorithm **Prove** in Construction 2. The proofs are on the form $\Sigma_i = (V_i, a_i, D_i, z_{x_i}, z_{\tau_i}, z_{R_i})$ for $i = 1, 2$. Denote the challenges used to construct the proofs c_i for $i = 1, 2$ and the related Pedersen commitments are denoted C_i for $i = 1, 2$.

The naive construction of the algorithm **Aggregate** is to define the aggregated proof, Σ_a, as the element-wise product or addition of the two proof Σ_1 and Σ_2. This yields that $\Sigma_a = (V_a, a_a, D_a, z_{x_a}, z_{\tau_a}, z_{R_a})$, where $V_a, a_a, D_a, z_{x_a}, z_{\tau_a}, z_{R_a}$ are computed as,

$$\begin{aligned}
V_a =& V_1 V_2 = g^{\frac{\tau_1}{\chi + x_1}} g^{\frac{\tau_2}{\chi + x_2}} \\
a_a =& a_1 a_2 = \left(e(g,g)^{\frac{-s_1 \tau_i}{\chi + x_1}} e(g,g)^{t_1} \right) \left(e(g,g)^{\frac{-s_2 \tau_2}{\chi + x_2}} e(g,g)^{t_2} \right) \\
D_a =& D_1 D_2 = (g^{s_1} h^{m_1})(g^{s_2} h^{m_2}) = g^{s_1 + s_2} h^{m_1 + m_2} \\
z_{x_a} =& z_{x_1} + z_{x_2} = (s_1 - c_1 x_1) + (s_2 - c_2 x_2) \\
z_{R_a} =& z_{R_1} + z_{R_2} = (m_1 - c_1 R_1) + (m_2 - c_2 R_2) \\
z_{\tau_a} =& z_{x_1} + z_{x_2} = (t_1 - c_1 \tau_1) + (t_2 - c_2 \tau_2)
\end{aligned} \tag{C.1}$$

$C = C_1 C_2 = g^{x_1 + x_2} h^{R_1 + R_2}$ is the product of the two Pedersen commitments and $c = c_1 c_2$ denotes the product of the challenges .

Investigated if the aggregated proof Σ_a satisfies the equations: $D_a = C^c h^{z_{R_a}} g^{z_{x_a}}$ and $a_a \overset{?}{=} e(V_a, y)^c e(V_a, g)^{-z_{x_a}} e(g,g)^{z_{\tau_a}}$.

Considering the first equality it follows that.

$$\begin{aligned}
& D_a = g^{s_1 + s_2} h^{m_1 + m_2} \\
& C^c h^{z_{R_a}} g^{z_{x_a}} = \ ... \ = g^{s_1 + s_2} h^{m_1 + m_2} g^{c(x_1 + x_2) - c_1 x_1 - c_2 x_2} h^{c(R_1 + R_2) - R_1 c_1 - R_2 c_2} \\
\implies & D_a \neq C^c h^{z_{R_a}} g^{z_{x_a}}
\end{aligned}$$

Thereby the completeness property is not satisfied for the aggregated proof Σ_a. The equality does not hold due to the $c(x_1 + x_2) \neq x_1 c_1 + x_2 c_2$. Note that if $c_1 = c_2 = c$ then the above is an equality.

D
Complete Aggregation

To fully aggregate the signature-based set membership proofs the verification would need to be independent of the number of provers.

Consider two set membership proofs Σ_1 and Σ_2 computed according to the algorithm **Prove** in Construction 2. The proof $\Sigma_i = (V_i, a_i, D_i, z_{x_i}, z_{\tau_i}, z_{R_i})$ for $i = 1, 2$. Denote the Pedersen commitments by C_i and the challenges c_i, for $i = 1, 2$. It will be assumed, although unrealistic for non interactive constructions of set membership proof, that $c = c_1 = c_2$.

The aim is to investigate if the two proofs can be aggregated into the aggregated proof, $\Sigma_a = (V_a, a_a, D_a, z_{x_a}, z_{\tau_a}, z_{R_a})$, such that $a_a \overset{?}{=} e(V_a, y)^c e(V_a, g)^{-z_{x_a}} e(g, g)^{z_{\tau_a}}$ holds and that equality implies that $x_i \in \Phi$ for $i = 1, 2$.

The aggregation is performed according equation (C.1), using c as the challenge. This given that,

$$a_a = e(g,g)^{\frac{\tau_1 s_1}{q+x_1}\frac{\tau_2 s_2}{q+x_2}+t_1+t_2}$$

$$e(V_a, y)^c e(V_a, g)^{-z_{x_a}} e(g, g)^{z_{\tau_a}} = \ldots = e(g,g)^{\frac{s_1\tau_1}{q+x_1}+t_1} e(g,g)^{\frac{s_2\tau_2}{q+x_2}+t_2} e(V_1, g)^{-z_{x_2}} e(V_2, g)^{-z_{x_1}}$$

$$\implies a_a \neq e(V_a, y)^c e(V_a, g)^{-z_{x_a}} e(g, g)^{z_{\tau_a}}$$

Even under the assumption that $c = c_1 = c_2$, the terms $e(V_1, g)^{-z_{x_2}} e(V_2, g)^{-z_{x_1}}$ are not cancelled. Thereby, although the aggregation is performed such that the challenges only appears as a product the completeness property does not hold.