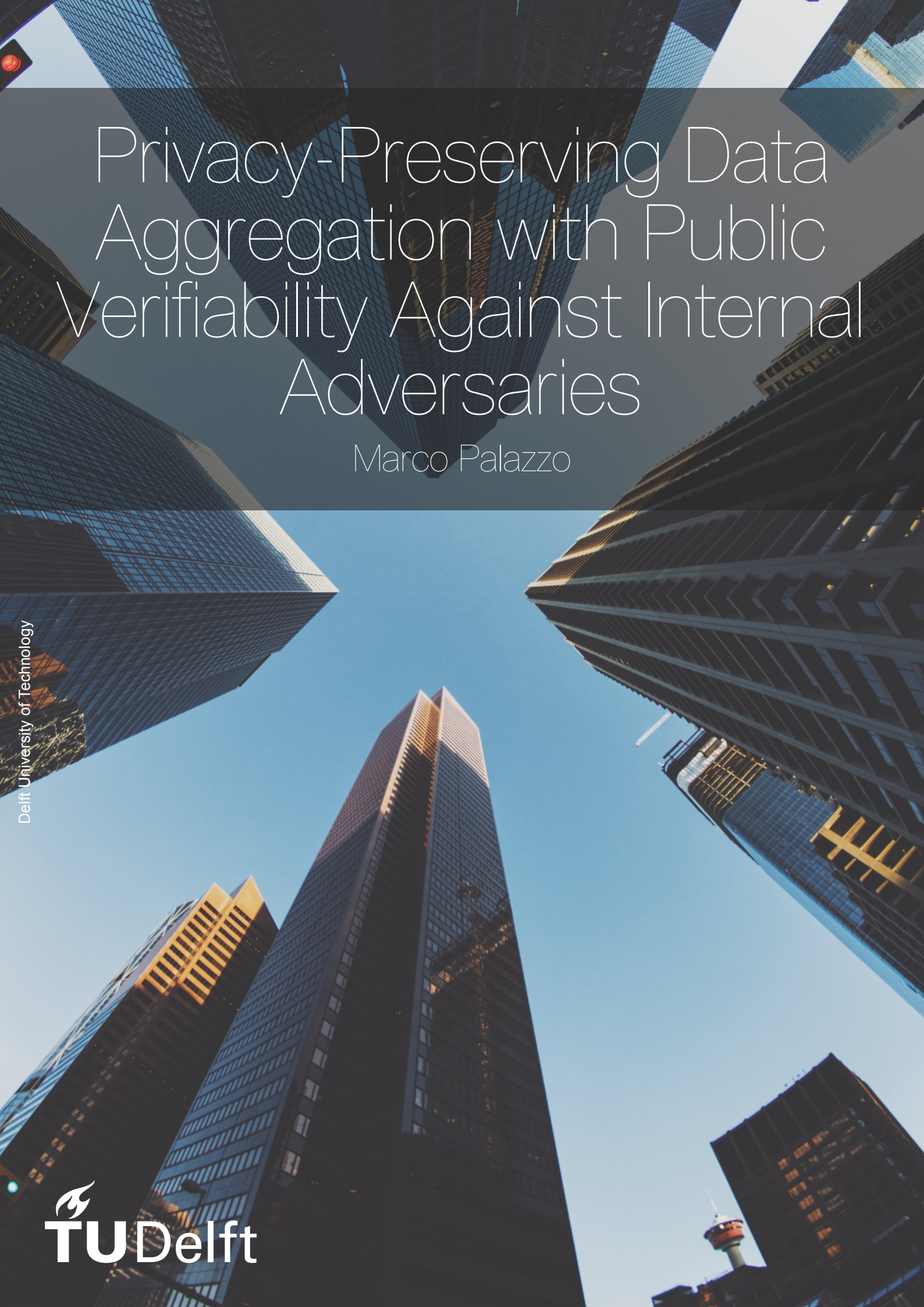


Privacy-Preserving Data Aggregation with Public Verifiability Against Internal Adversaries

Marco Palazzo

Delft University of Technology

TU Delft

Privacy-Preserving Data Aggregation with Public Verifiability Against Internal Adversaries

by

Marco Palazzo

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday, October 28, 2022 at 09:00 AM.

Student number:	5362202	
Project duration:	November 16, 2021 - October 28, 2022	
Thesis committee:	Dr. Z. Erkin	TU Delft, Supervisor
	Dr. J.E.A.P. Decouchant	TU Delft
	F. W. Dekker	TU Delft, Daily supervisor

An electronic version of this thesis is available at `https://repository.tudelft.nl`

TU Delft

Preface

I decided to work on the topic of privacy-preserving data aggregation after reading about some of the amazing applications it has in practice. I specifically decided to work on publicly-verifiable data aggregation after following the Secure Data Management course and learning about its properties. I spent around 3 months reading the related literature and trying to identify the limitations of the existing schemes. After settling on my research question, I started with the design process of my solution, which involved several iterations of what would become the final protocol presented in this work. I spent the last few months evaluating my solution and writing this thesis.

I would like to thank Dr. Zeki Erkin, for his continued support throughout the duration of this project and for always pushing me to do more and improve my work, and Florine Dekker, for her invaluable feedback during the designing and writing phases. I would also like to thank Dr. Jérémie Decouchant for being part of my thesis committee.

Special thanks to Armin, Aslı, Cassie, Charlie, Linus, Luke, and Sven, with whom I shared the last 2 to 5 years of this journey, for our discussions, game nights, and support during the pandemic.

Last but not least, I would like to thank my family for always being by my side.

Marco Palazzo
Den Haag, October 2022

Abstract

Large amounts of data are continuously generated by individuals, apps, or dedicated devices. These data can be aggregated to compute useful statistics from multiple sources using data aggregation protocols. However, oftentimes these data contain private information that must be protected from misuse. Privacy-preserving protocols can help to compute the same statistics without revealing the private input data to unauthorized parties. However, most privacy-preserving data aggregation schemes only work in the honest-but-curious model, where participants do not deviate from the protocol. As a result, the computed statistics cannot always be trusted. In particular, the aggregator, which is the party responsible for collecting the data is usually trusted with computing the correct result. However, a compromised aggregator may decide to output any value of its choosing without anyone noticing. Schemes with public verifiability help to counter malicious aggregators who try to falsify the final result by tampering with the inputs of honest users. However, they do not consider cases where both the aggregator and a subset of the users may be malicious, meaning they can deviate from the protocol and collude with each other in order to output results of their choosing. In this work, we develop a privacy-preserving data aggregation protocol to compute the sum of a set of private inputs such that a verifier can efficiently detect tampering even in the face of a malicious aggregator and a subset of malicious users. We also provide two extensions for better performance and for malicious user detection. We show that the scheme achieves the desired properties of confidentiality, integrity, and authenticity. Finally, theoretical and experimental evaluations show that its performance makes it feasible for real-world applications.

Contents

1 Introduction

We live in a data-driven society, where large amounts of data are being constantly generated, processed, and analyzed to better inform our societal and business decisions. However, in the age of mass surveillance by governments [38] and increasing interest in personal data for political [15] or economical [62] motives, it is of the utmost importance to protect sensitive data.

Privacy-preserving data aggregation protocols help with the problem of securely processing private data, but they are not a one-size-fits-all solution. Every use case has different requirements, for example, in terms of security, reliability, performance, and usability, that must be addressed in order to make the adoption of these schemes more compelling and feasible in practice.

1.1. Data aggregation

Data aggregation is a process by which raw data is gathered from one or more sources and processed for statistical analysis. Some examples of summarization functions include the sum, average, minimum, maximum, standard deviation, and variance. Data aggregation enables us to make sense of the increasingly larger amounts of data that are generated every day. Nonetheless, data aggregation is not exactly a novel concept.

One of the earliest examples of data aggregation and, more generally, large-scale computing [70], is represented by censuses. A census is a systematic procedure to acquire and record information about the members of a population [4]. The first recorded census dates as far back as 3800 BC at the times of the Babylonians [25], where teams of men were sent door to door with clay tablets to tally up the number of men, women, children, livestock, and other goods. The main goals were to estimate how much food was needed to feed the population, to count the number of men fit to join the military service, and to establish a ceiling on how much the citizens could be taxed without starving them [52]. Nowadays, the data collected by censuses are processed using modern computers and used to calculate more fine-grained statistics about a demographic such as total population, geographic distribution, average age, education level by age and gender, and many more. In the US, national censuses are mandated by the constitution and their primary use is the apportionment of the seats in the House of Representatives based on state population [12]. Furthermore, census information is used for military and disaster planning in the event of disasters such as floods, hurricanes, and tornadoes. They are also used for product development and marketing, for example, to identify foreign language areas and place products and services tailored to people who speak a language other than English [82].

Perhaps the most popular and pervasive example of the importance of data aggregation during the past two years is given by the COVID-19 pandemic [17]. Since the beginning of the pandemic, data about individual infections, recoveries, and deaths were continuously collected both at small and large scales for epidemiological surveillance purposes. These data are aggregated to calculate several important metrics such as the incidence of infection in a specific population at a given period of time and have been used to better instruct government bodies on which policies to adopt to help combat the spread [21, 42].

Another common usage of data aggregation is in smart grids. A smart grid is a modern power grid

infrastructure that allows for bi-directional flows of electricity and information. It also features resilient, automated power delivery and balancing mechanisms [29]. Metering data may be collected and aggregated at regular intervals from smart meters, installed in each household, to compute statistics such as the total power usage for a single household during some pre-established intervals or from entire neighborhoods for billing or load-balancing purposes.

In the healthcare industry, smart devices have been increasingly used in the last few years to collect various medical data from patients and store it in dedicated cloud repositories. These data are used to continuously monitor the health status of a patient for better treatment [85]. Data aggregation is often used in such scenarios to summarize the health data in a more succinct way and to reduce redundancies in the data [85]. Another pivotal application of data aggregation in the healthcare domain is combining data from multiple, similar medical cases for research purposes. The general goal is that of building a more robust body of medical knowledge for faster diagnoses and more effective treatments [23].

Data aggregation is also an important primitive for web analytics. Content-driven websites are often interested in learning how visitors spend time on a website and what content was more popular. Additionally, aggregating visitors by location, time of visit, time spent, and browsing devices is often useful to construct user behavior and demographic profiles and tailor the website's content accordingly to maximize its effectiveness. Often, web analytics are collected using third-party services such as Google Analytics or Adobe Analytics which can use the aggregated data from multiple websites to create even larger and more accurate user profiles and sell these to advertisers or publishes to improve the reach of their products [2].

Last but not least, the unprecedented and widespread adoption of mobile devices such as smartphones, sometimes possessing enough computation power to surpass even personal computers [87]. These devices are often equipped with embedded sensors such as GPS, accelerators, gyroscopes, digital compasses, microphones, and cameras, from which heterogeneous data can be collected and aggregated, enabling a variety of applications [83]. Participatory sensing is a novel paradigm that aims at harnessing this distributed network of sensors for several purposes, including traffic monitoring and geo-imaging [43], analyzing the roughness of a road and its noise level in order to build accurate cycling maps for cyclists [80], or air pollution monitoring [60].

1.2. Protecting private information

While the benefits of data aggregation for our society and for businesses cannot be ignored, without proper countermeasures, they often come at the cost of sacrificing the privacy of individual users.

Censuses have historically been controversial due to the privacy violations that might come from abusing and misusing raw data. In the US, despite the countermeasures adopted in the past, such as removing information that can directly identify citizens from the aggregate data or making the raw data available only 72 years after the census, it is sometimes still possible to link anonymous records to individual identities [14]. Indeed, aggregate data can only hide information about individual citizens when it is computed from a large enough sample. Attributes such as gender, race, and salary may be enough to uniquely identify citizens of a small town when, for example, only two black males with different occupations live there. In order to overcome these issues, for the 2020 census, the Bureau decided to adopt differential privacy as a means to protect the census data [11]. Differential privacy [26] is a technique by which statistical noise is added to the raw data in such a way that privacy is maintained while keeping the data useful.

Smart grids are also prone to privacy issues. While it may not be immediately obvious, metering data contain privacy-sensitive information that, when collected over long periods of time, can be used to build user profiles and gain insights that extend far beyond just simple power consumption reports. Indeed, from metering data, it is possible to infer which types of devices are used, when they are used, and whether an appliance, like a fridge, starts to become old [40]. This information can later be used for targeted advertisements. It is also possible to guess when someone is not at home [34], which can be misused by burglars. Incredibly, it is also possible to infer whether someone belongs to a specific religious group. Indeed, a user that is also a devout Muslim may wake up at around 5 am for their morning prayer, and a spike in energy consumption at around the same time, coupled with other information such as their names, allows one to make a confident guess about their religion [34].

In the healthcare sector, patient data is particularly sensitive in light of the rise of Electronic Health

Records (EHR) and ever more pervasive data collection and processing technologies [59]. Patient data includes sensitive information about a person and their disease history and, as such, confidentiality of such data is recognized as a patient's right by most countries [46]. Indeed, health data is heavily regulated by compliance policies such as the Health Insurance Portability and Accountability Act (HIPAA) [71] in the US and the General Data Protection Regulation (GDPR) [35] in Europe. Disclosure of private medical information may cause healthcare institutions or individual medical professionals to incur high regulatory fines [45]. Additionally, it may erode patients' trust and lead them to harmful behaviors. Indeed, as of 2013 between 15% and 17% of US adults had changed their behavior by switching doctors, paying out-of-pocket despite being insured, giving incomplete information on medical history to avoid listing embarrassing conditions, or even self-treating or self-medicating [59]. Moreover, disclosed private medical data can be directly used against a patient, for example, by an insurance company to deny coverage or increase the premium rates [73], or by an employer to deny employment [46]. Consequently, it is of extreme importance to create medical data sharing and processing systems that can protect patient data, withstand the burdens of regulatory compliance, and enrich our overall body of medical knowledge for better treatments and faster diagnoses.

Despite data protection is so important and heavily regulated, breaches are still very common. In fact, between 2009 and 2021, the number of data breaches in the US healthcare sector steadily increased, resulting in the loss, theft, or disclosure of 314,063,186 healthcare records [41]. Hacking incidents and unauthorized access were the main causes of most of the breaches. In 2015 alone, Anthem, Inc., an American health insurance provider, revealed that hackers had broken into some of the company's servers and stolen up to 78.8 million records containing personally identifiable information, which are now expected to be sold on the black market [72]. The massive data breach also resulted in several lawsuits against the company, which were settled at the massive cost of $115 million [31].

There are, however, countermeasures that can limit the impact of these breaches or even prevent them. Privacy-enhancing technologies (PET) are a collection of techniques and guidelines which aim at increasing the control over personal data and protecting it from misuse [77]. Examples of PETs include access control, tunnel encryption, and onion routing. Since we deal with data aggregation, a class of PETs that is of particular interest to us is privacy-preserving data aggregation (PPDA) schemes. This is a class of schemes that allows for the distributed computation of various data summarization functions over a set of private data points held by multiple participants. The main goal of these schemes is that they do not reveal the individual data points submitted during execution to anyone but the data owner itself and other authorized parties. We discuss several examples of PPDA schemes in Chapter 3.

1.3. Current solutions

Many private data aggregation protocols, e.g. [28, 79, 51], assume an honest-but-curious model where all participants follow the protocol but may try to infer private information they are not entitled to know. Nonetheless, in many of these schemes, the aggregator is usually in charge of aggregating the submitted values as well as decrypting and publishing them. As a result, the aggregator has the power to modify the aggregate or even output any arbitrary values of its choosing. This is a fair assumption in many realistic scenarios. For example, in smart grids, if the aggregator is found to be tampering with the submitted value it may cause huge reputational damages to the utility company and it may infringe privacy regulations, such as GDPR. However, these consequences are contingent on tampering attempts being detected. Many data aggregation schemes, like the ones we summarize in Section 3.1, do not offer efficient methods to detect such tampering attempts unless they come from external actors and, instead, they just assume that they will be eventually detected or that the consequences will deter participants from acting maliciously. Incorrect statistics can, however, be dangerous.

In smart grids, if incorrect statistics about the power usage of a neighborhood are sent to the responsible control center, then power outages may occur. The Northeast blackout of 2003, was a power outage that affected the Northeast and Midwest of the United States, as well as parts of Ontario in Canada [18]. The outage lasted between 2 hours, in some places, to almost 2 days in New York City and affected millions of people [22]. There are multiple causes that led to the blackout, ranging from violations of established safety standards, human error, and software bugs. A common aspect in the chain of events was the lack of situational awareness about the need to re-balance the power distribution due to computer failures and incorrect telemetry readings [64].

When dealing with medical data, correctness is also of the utmost importance. Indeed, incorrect

data can lead to incorrect diagnoses and may have catastrophic consequences for patients.

During the past decade, some authors have attempted to tackle this problem from various angles. On the one hand, a possible cause of incorrect statistics is that of data poisoning injection attacks at the hand of malicious users, where out-of-range values are submitted to skew the result in a specific direction or to render it completely useless. These attacks are generally mitigated by having the aggregator check whether each submitted value is in a valid range, for example, using zero-knowledge proofs or other techniques as in [48, 44]. On the other hand, another cause of incorrect statistics is a compromised aggregator that outputs an aggregation result that is different from the real one. A simple technique to counter this problem is to have each user sign their submitted value. However, this incurs high verification costs as possibly thousands of signatures need to be verified in applications with many users. A more common technique adopted in the literature is to make only the result of the aggregation verifiable. However, many of the proposed solutions, like [5, 54], only assume a malicious aggregator but they do not take into account the possibility that users may also be compromised. Two recent papers [53, 63] tried to tackle the problem of performing data aggregation with public verifiability in the presence of aggregators and users that are malicious both with respect to the confidentiality of the inputs and integrity of the result. However, the solution proposed in [63] does not take collusions between the aggregator and the users into account. Moreover, it works in a weaker model that requires additional semi-trusted parties to help with the computation. The scheme proposed in [53] also suffers from several drawbacks. Like [63], it relies on a semi-trusted third party which is a single point of failure that, if compromised, undermines the whole premise of the scheme. Furthermore, it does not provide any mechanism to detect which users act maliciously and the verification of the result can only be carried on by a trusted party. Finally, it is still possible for the aggregator to collude with a user and forge valid aggregation results by exploiting a vulnerability in the protocol.

1.4. Research question

The main goal of this thesis is to design a protocol to compute the sum of a set of private values without revealing anything other than the sum itself. The parties involved in the protocol include a trusted authority, a powerful aggregator, and a set of users that hold the data to be aggregated. We work in the malicious model where a bounded subset of users may behave maliciously and collude with a malicious aggregator in order to learn the private values of other users as well as to affect the integrity and authenticity of the result. In other words, no party should be able to learn the private data of an other participant nor tamper with it once it is submitted, and the aggregator should not be able to publish a result that is not equal to the sum of the submitted data. The result of the aggregation can be publicly verified by any party that holds that verification key. In this thesis, we focus on the sum because it is the most common statistic for data aggregation and can be used as a primitive for more complex techniques, for example, federated learning [7]. Our research question is the following.

> **How can a privacy-preserving data aggregation scheme with public verifiability achieve confidentiality of the private inputs, integrity and authenticity of the aggregate statistic, in the face of a malicious aggregator, multiple malicious users, and without relying on additional semi-trusted parties during aggregation?**

1.5. Contributions

In this thesis, we propose a data aggregation protocol to compute the sum of a set of privately-held values in a privacy-preserving and publicly-verifiable manner. More specifically, given a set of registered users, each holding a private integer value, the protocol computes the sum of all values in such a way that no party can learn the private value of another user. Only the final sum is revealed in the end. Additionally, the protocol allows any party that holds the verification key to check, in constant time, whether the result was computed from the values submitted by the registered users only, and that the aggregate does not consist of any other value. To the best of our knowledge, our protocol is the first to ensure confidentiality of the private values as well as integrity and authenticity of the aggregate in the presence of a malicious aggregator and a bounded number of malicious and colluding users, without relying on other semi-trusted parties. Note that data poisoning attacks from the users are outside of the scope of this thesis. Additionally, we augment our main protocol with two extensions. The first extension drastically improves the communication overhead at the cost of a weaker security model with non-

adaptive user corruptions. The second extension allows the aggregator to identify users that behave maliciously and that attempt to disrupt the correct execution of the protocol by causing the verification to constantly fail. Finally, we provide theoretical evaluations of the security and performance of our protocols as well as a practical analysis of their performance with a proof-of-concept implementation.

1.6. Outline

In Chapter 2, we discuss the essential mathematical and cryptographic concepts required to understand our protocols. In Chapter 3, we lay out an overview of some related works. In Chapter 4, we describe our main privacy-preserving and publicly-verifiable data aggregation protocol. In Chapter 5, we proceed to present two extensions to the main protocol that provide better communication complexity and detection of malicious users, respectively. In Chapter 6, we provide security analyses for the main protocol and for the two extensions. In Chapter 7 we analyze the theoretical computational and communicational complexity of the protocols as well as their experimental running time for each participant. Finally, in Chapter 8, we provide a final discussion of our results.

2

Preliminaries

In this chapter, we provide an overview of the cryptographic techniques required to understand the protocols presented in Chapter 4 and Chapter 5 as well as to understand the related literature. We assume the reader is already familiar with some fundamental concepts of cryptography such as groups and finite fields, number theory, and modular arithmetic, which can also be referenced in [81, 49].

2.1. Secret sharing

While encryption is useful to protect data from unauthorized access, it is only one piece of the puzzle when it comes to data protection. One of the golden rules of cryptography is known as Kerckhoff's principle, which states that a cryptosystem should be secure even if everything about the system, with the exception of the secret key, is public knowledge [32]. As such, keeping the secret keys safe seems like another important step to effectively protect data. However, this raises the question: how can we keep the secret keys safe? A common approach would be to simply store a secret key in a well-guarded and inaccessible place. Unfortunately, this becomes a single point of failure because if the key were to be lost or damaged, then decryption would be impossible afterward. Another solution would be to split the key into several pieces, but this is non-trivial since naive solutions may not be secure and leak information. This is where secret-sharing schemes come into play. They provide a way to split a secret S into several shares, say n, such that no individual share reveals any information about the secret, but when the shares are combined the original secret can be reconstructed. These are particularly suited for applications in which a group of untrusting parties must cooperate to perform some action, for example, opening a safe, but no individual party should have the authority to do so without the collaboration of the other parties. There are two common methods to share a secret: additive secret sharing and threshold secret sharing.

2.1.1. Additive secret sharing

A rather simple yet effective solution is additive secret sharing. Assume we want to split the secret S into n shares s_i such that S can be recovered when all shares are combined together, that is:

$$S = \sum_{i=1}^{n} s_i \mod p. \tag{2.1}$$

In order to achieve this, we sample $n-1$ random numbers $s_1, s_2, \ldots, s_{n-1}$ and then let

$$s_n = S - \sum_{i=1}^{n-1} s_i \mod p. \tag{2.2}$$

As a result, all shares must be combined to reconstruct S. All operations are performed over a finite field of integers modulo a large prime p. It is common among many privacy-preserving data aggregation schemes to choose $S = 0$, because the secret shares can be used as additive masks to conceal a private value so that when n shares are added together, the masks cancel out revealing only the final aggregate value.

2.1.2. Threshold secret sharing

The other common method is to use a (k, n) threshold scheme. This is a more general method by which the secret can be split into n shares such that when any k shares are combined, with $k \leq n$, then the secret can be reconstructed. One of the most popular (k, n) threshold schemes is Shamir Secret Sharing (SSS) [78], which is based on polynomial interpolation. If we have k points $\{(x_1, y_1), \ldots, (x_k, y_k)\}$, with $x_i \neq x_j$, then there exists a polynomial $f(x)$ of degree $k-1$ such that $f(x_i) = y_i$, for $1 \leq i \leq k$. Thus, in order to split a secret S into n shares such that k shares can reconstruct S, we pick a polynomial of degree $k-1$:

$$f(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{k-1} x^{k-1} \mod p, \tag{2.3}$$

where p is a large prime, $c_0 = S$, and each $c_i \leftarrow\$ \mathbb{Z}_p$ is a random integer modulo p. Then, we can create n shares $s_1 = (x_1, f(x_1)), ss_2 = (x_2, f(x_2), \ldots, ss_n = (x_n, f(x_n))$, and distribute them among the participants in the protocol. When any subset of k share-holders want to find S, they can use their shares to reconstruct the polynomial $f(x)$ via polynomial interpolation. A common technique to achieve this is by using Lagrange interpolation. First, we compute the Lagrange basis polynomials:

$$l_j(x) = \prod_{0 \leq j \leq k-1, m \neq j} \frac{x - x_m}{x_j - x_m} \mod p, \tag{2.4}$$

then the polynomial can be recomputed with:

$$f(x) = \sum_{j=0}^{k-1} y_j l_j(x) \mod p. \tag{2.5}$$

Since we are usually interested in $f(0) = S$, a more efficient formula to reconstruct the secret is given by:

$$f(0) = \sum_{j=0}^{k-1} y_j \prod_{m=0, m \neq j}^{k-1} \frac{x_m}{x_m - x_j} \mod p. \tag{2.6}$$

To conclude, let us see why an adversary that gets hold of $k-1$ secret shares cannot recover S. For each possible secret $S' \in [0, p-1]$, there exists exactly one polynomial $f'(x)$ of degree $k-1$ with $f'(0) = S'$ and $f'(i) = y_i$, for $1 \leq i \leq k-1$. Each of these p polynomials is equally likely to be $f(x)$, which renders the scheme information theoretically secure.

2.2. Bilinear maps

A bilinear map, or pairing, is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that, for all $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, it holds:

$$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}. \tag{2.7}$$

A bilinear map associates pairs of elements from $\mathbb{G}_1$ and $\mathbb{G}_2$ with elements in $\mathbb{G}_T$. In order to actually make pairings useful, we further require that they are efficiently computable and that any $e(g_1, g_2)$ is a generator of $\mathbb{G}_T$, which excludes degenerate bilinear maps in which $e(g_1, g_2) = 1$, for every $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$.

Bilinear maps are instantiated using elliptic curves over finite fields. The cyclic groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are assumed to be of the same prime order p, i.e. they have the same number of elements p. There are three types of bilinear maps [33]:

- Type 1: when $\mathbb{G}_1 = \mathbb{G}_2$;
- Type 2: when $\mathbb{G}_1 \neq \mathbb{G}_2$ but there exists an efficiently computable homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$;
- Type 3: when $\mathbb{G}_1 \neq \mathbb{G}_2$ and there are no known efficiently computable homomorphisms $\phi : \mathbb{G}_2 \to \mathbb{G}_1$.

One of the reasons why bilinear maps are useful for cryptographic protocols is because they permit "cheating" at computing the Diffie-Hellman function in $\mathbb{G}_T$. In other words, if we have $g^a, g^b, g^c \in \mathbb{G}$, with $b, c \in \mathbb{Z}_p$ secret, then we can efficiently compute $\hat{g}^{abc} = e(g^b, g^c)^a \in \mathbb{G}_T$ without first having to compute $\hat{g}^{bc}$. Bilinear maps have applications in many fields of cryptography, like identity-based encryption, homomorphic signatures, and zero-knowledge proofs [86].

2.3. Cryptographic hash functions

Hash functions are functions that map inputs of some arbitrary length to fixed-length outputs [49]. More formally, they are functions $H\colon \{0,1\}^* \to \{0,1\}^n$. They are useful building blocks for digital signature schemes. In cryptography, there are three increasingly stronger notions of security for hash functions [81].

The first one is *preimage resistance*. A hash function is said to be preimage resistant if, given y from the codomain of H, it is computationally infeasible to find any x in the domain of H such that $H(x) = y$. A hash function for which this property holds is said to be *one-way*. More specifically, a preimage-resistant hash function that produces l-bit outputs requires $\mathcal{O}(2^l)$ time to find a preimage.

The second, stronger, security property is *second preimage resistance*. A hash function H is second preimage resistant when, given x in the domain of H and $H(x)$, it is computationally infeasible to find another y in the domain of H such that $H(y) = H(x)$.

The third, and strongest, security property required by cryptographic hash functions is *collision resistance*. A hash function is said to be collision resistant if it is infeasible to find two distinct values x and y such that $H(x) = H(y)$. Note that, from the definition of a hash function is clear that its domain is usually much larger than its codomain and, as such, by the pigeonhole principle, collisions are bound to exist. Collision resistance is the strongest notion of security for hash functions because it also implies the other two.

2.4. Homomorphic encryption

Homomorphic encryption is a cryptographic technique that enables computations directly on encrypted data without having to first decrypt it. Homomorphic cryptosystems are heavily relied upon by algorithms that perform computations on privacy-sensitive data. Given the encryptions $\mathcal{E}(m_1)$, $\mathcal{E}(m_2)$ of two messages m_1, m_2, a cryptosystem is said to be homomorphic if the following property holds:

$$\mathcal{E}(m_1) \otimes \mathcal{E}(m_2) = \mathcal{E}(m_1 \oplus m_2). \tag{2.8}$$

In this formula, the symbols $\otimes$ and $\oplus$ refer to two arbitrary, possibly different arithmetic operators. In other words, applying the $\otimes$ operator to the two ciphertexts results in a new ciphertext over the message resulting from applying the $\oplus$ operator to the two original messages.

There are two main classes of homomorphic cryptosystems: those that provide *partial homomorphic encryption* and those that provide *fully homomorphic encryption*. Partially homomorphic (PHE) schemes only allow for a single operation over the encrypted day, whereas fully homomorphic (FHE) schemes allow for multiple operations. Homomorphic schemes can be further split based on the number of operations allowed. For example, the Boneh-Goh-Nissim (BGN) scheme [8] is both multiplicatively and additively homomorphic. This, intuitively, means that we can perform both addition and multiplication over encrypted data. However, while the scheme allows for an unlimited number of additions, only one multiplication is permitted. These types of schemes are often referred to as *somewhat homomorphic*. The first FHE scheme to allow for an unbounded number of additions and multiplications and, thus, for the evaluation of arbitrary arithmetic circuits, is due to [36]. While FHE is often considered to be the Holy Grail of cryptography, most practical FHE schemes suffer from significant performance limitations.

2.4.1. The Paillier cryptosystem

A popular example of a public-key cryptosystem that supports additively-homomorphic operations is the Paillier cryptosystem [67]. While Paillier encryption is not used in our work, it is often used by other related works and provides a simple, practical introduction to homomorphic encryption. A Paillier key pair can be generated by sampling two random prime numbers p, q. The approximate size of each prime number depends on the chosen security parameter κ. If $\kappa = 1024$, then p and q can be chosen to have a bitlength of 512 bits each. Next, we compute $n = pq$ and $\lambda = \mathsf{lcm}(p-1, q-1)$. A base $g \leftarrow\!\!\$\, \mathbb{Z}^*_{n^2}$ must be randomly chosen such that $\gcd(L(g^\lambda \mod n^2), n) = 1$, where L is a function defined as:

$$L(x) = \frac{x-1}{n}. \tag{2.9}$$

The public key is $pk = (n, g)$, while the secret key is $sk = \lambda$. To encrypt a message $m < n$, we choose a random integer $r \leftarrow\!\!\$\, \mathbb{Z}_n^*$ and then compute:

$$\mathcal{E}_{pk}(m, r) = c = g^m \cdot r^n \mod n^2. \tag{2.10}$$

On the other hand, to decrypt a ciphertext c, we compute:

$$\mathcal{D}_{sk}(c) = m = \frac{L(c^\lambda \mod n^2)}{L(g^\lambda \mod n^2)} \mod n. \tag{2.11}$$

The Paillier cryptosystem is additively homomorphic because it holds

$$\mathcal{E}_{pk}(m_1, r_1) \cdot \mathcal{E}_{pk}(m_2, r_2) = g^{m_1} r_1^n \cdot g^{m_2} r_2^n = g^{m_1+m_2} \cdot (r_1 r_2)^n = \mathcal{E}_{pk}(m_1 + m_2, r_1 r_2). \tag{2.12}$$

The security of the Paillier cryptosystem relies on the hardness of the decisional composite residuosity problem: given a composite integer n and an integer z, decide whether there exists an integer y such that $z = y^n \mod n^2$. There is no known polynomial-time algorithm to solve this problem. Furthermore, due to the randomness included in each ciphertext, two encryptions of the same plaintext message will produce two different ciphertexts, making the Paillier cryptosystem semantically secure [81].

2.5. Commitments

A commitment scheme is a cryptographic primitive that allows a party to commit to some message m such that the commitment $\mathcal{C}(m)$ does not reveal the message itself. This is known as the *commit phase*. The commitment can then be sent to other parties. Afterward, during the *reveal phase*, the commitment can be opened by revealing the message m, along with some other information, such that any party can verify whether the commitment actually matches the revealed message. A cryptographic commitment should have two important properties:

- **Binding:** given a commitment $\mathcal{C}(m)$ on a message m, it is hard to find another message $m' \neq m$ such that $\mathcal{C}(m') = \mathcal{C}(m)$.
- **Hiding:** given a commitment $\mathcal{C}(m)$ on a message m, it is hard to find m.

Additionally, we can define two notions of security for these properties.

- **Computational security:** the property is maintained against polynomially-bounded adversaries.
- **Information-theoretical security:** the property is maintained even against adversaries that are computationally unbounded.

While the second security notion is stronger, a commitment scheme cannot be both information-theoretically binding and hiding at the same time.

2.5.1. Pedersen commitments

A popular example of a computationally binding and information-theoretically hiding commitment scheme is given by Pedersen commitments [68], which work as follows. We assume a cyclic group $\mathbb{G}$ of large prime order p with random generators $g, h \in \mathbb{G}$ are publicly known. The committer then commits to a message $m \in \mathbb{Z}_p$ by computing:

$$\mathcal{C}(m, r) = h^r g^m, \tag{2.13}$$

where $r \leftarrow\!\!\$\, \mathbb{Z}_p$ is a random value chosen by the committer. The committer then publishes $\mathcal{C}(m, r)$. During the reveal phase, the committer publishes (m', r') and anyone who possesses the commitment can verify its validity by checking:

$$\mathcal{C}(m, r) \stackrel{?}{=} h^{r'} g^{m'}. \tag{2.14}$$

The scheme is computationally binding because the committer needs to solve a discrete logarithm in order to find another pair $(m', r') \neq (m, r)$ such that $\mathcal{C}(m, r) = \mathcal{C}(m', r')$. There are currently no known polynomial-time algorithms to solve a discrete logarithm on classical computers.

The scheme is information-theoretically hiding because given a commitment $\mathcal{C}(m, r)$, there are multiple pairs $(m', r') \in \mathbb{Z}_p^2$ that generate the same commitment and all are equally likely to be the correct

candidates. As such, even a computationally-unbounded adversary can never know which pair is the correct one.

Pedersen commitments are relevant for us because their structure and properties are employed in many data aggregation schemes with public verifiability, including the schemes presented in this thesis. In these works, the generator h is often replaced with a random oracle and the properties of bilinear maps are exploited during the reveal phase for reduced interactivity.

We conclude by noting that Pedersen commitments are additively homomorphic. In fact,

$$\mathcal{C}(m_1, r_1) \cdot \mathcal{C}(m_2, r_2) = h^{r_1} g^{m_1} \cdot h^{r_2} g^{m_2} = h^{r_1+r_2} g^{m_1+m_2} = \mathcal{C}(m_1 + m_2, r_1 + r_2). \tag{2.15}$$

This property is particularly useful in order to create signatures that can be aggregated together.

2.6. Diffie-Hellman key agreement

The Diffie-Hellman (DH) key agreement protocol [24] allows two parties A and B to generate public-private key pairs (pk_A, sk_A) and (pk_B, sk_B) and use them to agree on a shared private key k_{AB}.

The protocol works as follows. First, we assume the existence of a public cyclic group $\mathbb{G}$ of prime order p along with a generator g. Each party then generates its asymmetric key pair. The secret key is a random element $x \leftarrow\!\!\$\, \mathbb{Z}_p$ and the public key is g^x. Thus, A obtains the key pair $k_A = (x, g^x)$ and B obtains $k_B = (y, g^y)$. In order to agree on a new shared secret key k_{AB}, A and B share their public keys with each other, and then they compute:

$$k_{AB} = (g^x)^y = (g^y)^x = g^{xy}. \tag{2.16}$$

Following the approach presented in [7], we can further extend the protocol by introducing a secure hash function H, for example, SHA-256 [37]. Then, the modified shared key becomes:

$$k'_{AB} = H(k_{AB}) = H(g^{xy}). \tag{2.17}$$

The reason for this modification is to ensure the shared key is a random string that can be used a seed of a secure pseudo-random number generator.

2.7. Zero-knowledge proofs

Sometimes we are interested in proving that some statement is true without revealing any other information other than the truth of the statement itself. For example, in an identification protocol, it might be required for a party to prove that they know the secret key for a given public key, without revealing what the secret key itself is. Zero-knowledge proofs (ZKP) solve this problem. They are two-party protocols between a prover and a verifier in which the prover convinces the verifier of the truth of some statement.

A simple yet very popular example of a zero-knowledge proof is the Schnoor protocol, shown in Figure 2.1. In parentheses, next to the name of the participant, we list the information known by the corresponding participant. The protocol works as follows: given a cyclic group $\mathbb{G}$ of order p with generator g and a group element $h = g^x$, the Schnorr protocol can convince a verifier that the prover knows the discrete logarithm of h, i.e. x, without revealing x to the verifier.

In this work, to simplify reasoning about ZKPs, we adopt the notation of Camenisch and Stadler [13] to succinctly express the objective of a ZKP. The general notation is as follows:

$$PK\{(w) : \mathcal{L}(x, w)\}. \tag{2.18}$$

In this notation, the witness w represents secret information, known to the prover but hidden from the verifier, while x represents public information. $\mathcal{L}(x, w)$ is the predicate over x and w that the prover intends to convince the verifier of without revealing any information about w. Following this notation, the Schnorr protocol can be summarized as $PK\{(x) : h = g^x\}$.

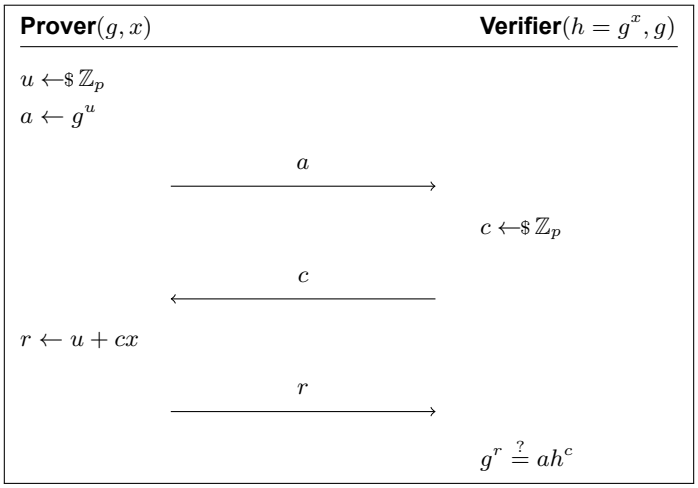


Figure 2.1: Interactive zero-knowledge protocol to prove knowledge of a discrete logarithm.

A ZKP should have three properties:

- **Completeness:** if the statement can be proven to be true and the prover and verifier follow the protocol, the verifier always accepts.
- **Soundness:** a cheating prover has negligible probability of convincing a verifier that a false statement is true.
- **Zero-knowledge:** the proof does not reveal any secret information to the verifier.

There are several classes of zero-knowledge proofs, including zk-SNARKs and zk-STARKs, which are nowadays featured heavily, especially in blockchain technology. However, in this work, we focus on two other types of protocols: sigma protocols and bulletproofs.

2.7.1. Sigma protocols

Many ZKPs follow the same underlying structure of the Schnorr protocol. The prover starts with an announcement, the verifier replies with a challenge, and, finally, the prover concludes the protocol with a response. Protocols that have this structure are called Sigma protocols and they can be used to prove various statements pertaining to discrete logarithms, including compound statements. In fact, more complex statements can be proven by combining multiple Sigma protocols together via composition. For example, via AND-composition it is possible to prove two statements at the same time by running two Sigma protocols, one for each statement, in parallel with a common challenge. With EQ-composition, which is a special case of AND-composition, it is possible to prove two statements that share one or more witnesses, by running two Sigma protocols for each statement using the same challenge, witness and random tape [76].

Sigma protocols can only be proven to be honest-verifier zero-knowledge, meaning that the zero-knowledge property may only hold in the presence of a non-cheating verifier. In fact, a cheating verifier may carefully choose the challenge in such a way that information about the secret witness is revealed in the response. While this may seem like an important limitation, in practice, it can be overcome using additional techniques like random oracles.

Zero-knowledge proof of equality between commitments

In Figure 2.2, we show a Sigma protocol that is used to prove that two different commitments share the same committed value without revealing what the value is, i.e. we have commitments $\mathcal{C}(x, r_1)$ and $\mathcal{C}(x, r_2)$ and want to prove that they are indeed committed to the same value x. Formally, it proves the relation:

$$PK\{(x, y, z) : S = g_1^x h_1^y \wedge T = g_2^x h_2^z\}. \tag{2.19}$$

The protocol consists of two Okamoto protocols [66] that run in parallel and are combined using EQ-composition because of the common witness x. Throughout this work, we refer to this protocol as ZKPEq.

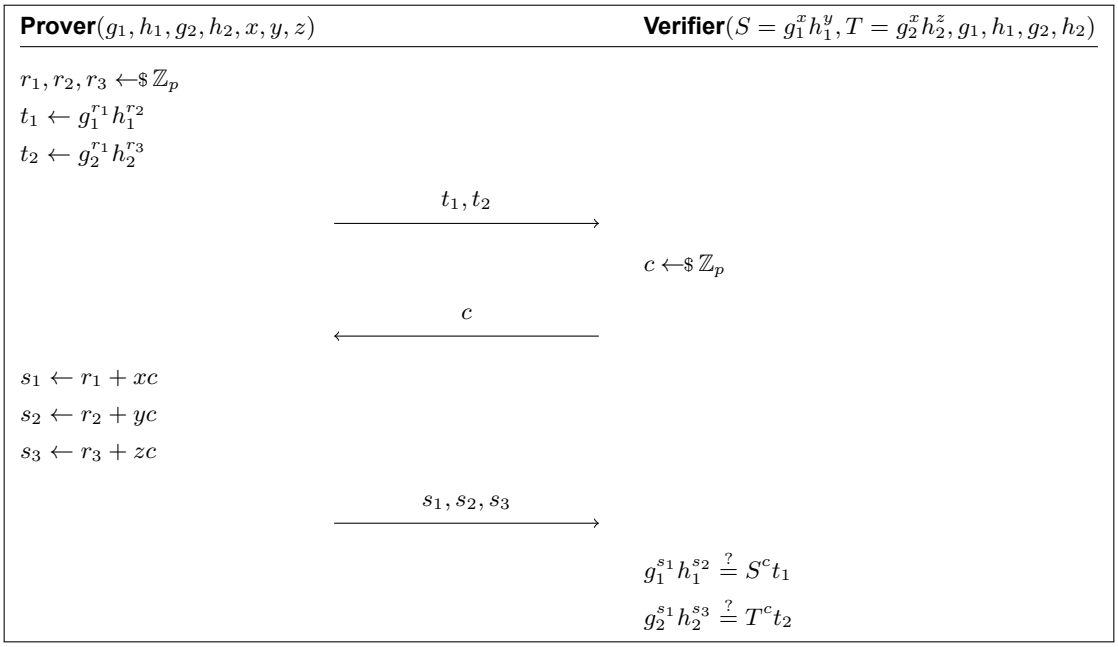


Figure 2.2: Sigma protocol to prove equality between commitments (ZKPEq), i.e. $PK\{(x, y, z) : S = g_1^x h_1^y \wedge T = g_2^x h_2^z\}$.

Zero-knowledge proof of inequality to zero

In this section, we present a zero-knowledge proof of inequality to zero. The proof, shown in Figure 2.3, can convince a verifier that neither of the exponents of a Pedersen-like commitment is 0. More formally, it proves the relation:

$$PK\{(x, y) : S = h^x g^y \wedge x \neq 0 \wedge y \neq 0\}. \tag{2.20}$$

The ZKP for the case in which only one exponent is non-zero can be found in [76], along with proofs of completeness, soundness, and zero knowledge. Here, instead, we present a ZKP for the case in which both exponents are non-zero at the same time. We combine two separate Sigma protocols for the single-exponent case, one for each exponent, via AND-composition by using a common challenge. The main idea behind the proof is that it forces the prover to come up with the multiplicative inverses of both exponents. However, this is only possible if they are non-zero since 0 has no multiplicative inverse. Throughout this work, we refer to the following ZKP as ZKPNeqZero. We use the notation $\zeta_{i,t} = (a, b, r_1, r_2, l_1, l_2)$ to refer to the tuple of messages that user u_i sends out in round t to perform an iteration of ZKPNeqZero.

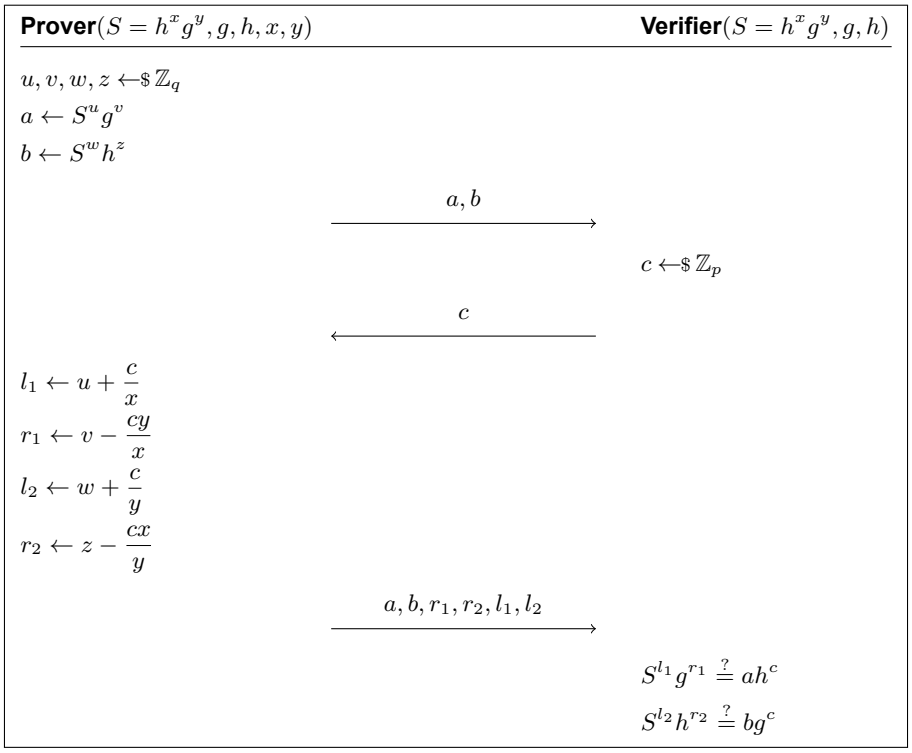


Figure 2.3: ZKP of inequality to zero of the exponents of a Pedersen commitment (ZKPNeqZero).

2.7.2. Fiat-Shamir heuristic

One of the main downsides of Sigma protocols is that they require interactivity between the prover and the verifier. The Fiat-Shamir heuristic [30] is a solution that can turn any interactive Sigma protocol into a non-interactive one by relying on random oracles to generate the challenges. Thus, both the prover and the verifier can generate the same challenge locally without interacting with each other. In practice, secure hash functions are used as random oracles. However, one must still pay particular attention to the information that is hashed in order to avoid cheating. In general, all public information as well as the statement itself should be included in the hash.

Another interesting property of the Fiat-Shamir heuristic is that it can also turn honest-verifier zero-knowledge Sigma protocols into zero-knowledge protocols, since using random oracles automatically forces any verifier to behave honestly [20].

2.7.3. Bulletproofs

Bulletproofs [10] are another class of ZKPs that has emerged recently. They are short and non-interactive ZKPs that do not require a trusted setup which can be used to prove a variety of statements about the structure of an encrypted plaintext. They are particularly useful to efficiently prove that some value is within a predefined range, e.g. $v \in [0, 2^{32})$, and are commonly used as a building block for confidential transaction systems such as Bitcoin, where they can be used to prove some secret asset values are non-negative. They can also be used to replace complex Sigma proofs with shorter and more efficiently computable proofs.

3

Related work

In this chapter, we present an overview of some notable privacy-preserving data aggregation schemes from the related literature. In Section 3.1, we start by describing some protocols that work in the honest-but-curious model where all parties behave as instructed by the protocol but may still try to infer information they are not entitled to know. In Section 3.2, we proceed to review some protocols in which the users may act maliciously by sending invalid values to the aggregator, which is still assumed to be honest but curious. Next, in Section 3.3, we describe some protocols in which the adversarial model is flipped by having the aggregator act maliciously and attempt to publish incorrect results, while the users remain honest but curious. In Section 3.4, we conclude the chapter by mentioning two schemes that attempt to tackle scenarios in which both the aggregator and the users may be dishonest, not only with respect to privacy but also to the integrity of the aggregation result.

3.1. Privacy-preserving data aggregation

In this section, we present several privacy-preserving protocols that work in the honest-but-curious model, where parties are assumed to follow the protocols honestly but may try to infer private information about other users in the system.

Kursawe et al. In [51], the authors present four protocols for private data aggregation in smart grids. These protocols are classified into aggregation protocols and comparison protocols. The aggregation protocols work by blinding the private measurements using zero-sum random masks in such a way that when all the blinded values are added together, the masks cancel out leaving out only the aggregated result. The comparison protocols work by assuming that the aggregator already knows an approximate value of the aggregation. The reason for this is that, in these protocols, the input values and the final aggregation itself are exponents of generators of some cyclic group. As such, the aggregator is required to either solve a discrete logarithm or, using its approximate value, test whether the result is in the close interval around this value. In one of the methods introduced in the paper, each user possesses a Diffie-Hellman key pair, with the public key being also known by all the other users in the system. p leaders are elected among the users. Each user, excluding the leaders, selects a random number r_j, with $1 \leq j \leq p$ for each leader, and sends it to the corresponding leader l_j, encrypted with the public key of l_j. Each leader l_j, then, generates a new random number r_j such that $r_j = -\sum_{i=1}^{n-1} r_i$, where n is the number of users in the system. When a non-leader user decides to submit a value x_i, it creates a blinded ciphertext c_i as follows:

$$c_i = x_i + \sum_{i=j}^{p} r_j \mod 2^{32}. \tag{3.1}$$

The leaders, instead, only use their blinding value r_j. Thus, when all values are submitted and aggregated by the aggregator, only the sum is revealed when all ciphertexts are combined, since the random masks cancel out. Some of the drawbacks of the scheme include the interactivity between the users and the lack of support for user dropouts.

Li et al. In [55], the authors present an in-network data aggregation protocol tailored for smart grids. The system model consists of a network of connected smart meters and a collector device, which is responsible for collecting the measurements and reporting the results to the central management. The scheme works in the honest-but-curious model. The Paillier cryptosystem is used to encrypt the individual smart meter measurements and its homomorphic properties are exploited to additively aggregate the ciphertexts and obtain the sum of all measurements. The collector holds the Paillier decryption key, while all the smart meters hold the public key. First, since we deal with in-network aggregation where all smart meters collaborate in the process, the collector and the smart meters collaborate to construct a spanning tree rooted at the collector. In this way, each smart meter can encrypt its private measurement using the Paillier public key and send it upward to its parent. The parent node, then, adds its own encrypted measurement to the received ones and sends the partially-aggregated ciphertext upward. In the end, the collector device homomorphically combines the received ciphertexts, decrypts the result, and sends it to the central management. The scheme has some downsides. First, the performance of the protocol heavily depends on the network topology. The shorter and wider the spanning tree is, the faster the protocol will perform. Moreover, since the Paillier cryptosystem is malleable, the scheme cannot guarantee the integrity of the result since a compromised smart meter or an external adversary might tamper with the individual ciphertexts, thus invalidating the result. Finally, the scheme is not collusion-resistant because if an internal node shares the ciphertext of another user with the collector, the collector may directly decrypt it.

Shi et al. In [79], the authors introduce a concept they call *aggregator obliviousness*. This concept captures three main security notions. The first one is that the aggregator can only learn the aggregated value at the end of a time period. Moreover, only the aggregator can compute the final aggregation. Finally, in the case of collusion between the aggregator and a subset of the users, then the aggregator can learn the sum of the remaining participants but nothing more about the participants' data.

In the same paper, the authors introduce a privacy-preserving data aggregation protocol that is aggregator oblivious. The protocol is non-interactive except for a trusted setup phase, performed only once, where the public parameters and secret keys are generated and distributed among the participants. The scheme works in a cyclic group $\mathbb{G}$ of prime order p and with random generator g. The secret keys of each user $sk_i \leftarrow\!\!\$\, \mathbb{Z}_p$, with $1 \leq i \leq n$, are randomly sampled during the setup. The secret key of the aggregator sk_0 is chosen in such a way that:

$$sk_0 = -\sum_{i=1}^{n} sk_i. \tag{3.2}$$

The protocol works on time-series data, so each input value is timestamped with an integer t to indicate the time period to which it belongs. To do so, the protocol assumes the existence of a secure hash function $H\colon \{0,1\}^* \to \mathbb{G}$. Each individual ciphertext $c_{i,t}$ for a private measurement x_i is computed as:

$$c = g^{x_i} \cdot H(t)^{sk_i}. \tag{3.3}$$

The aggregator can aggregate all ciphertexts for a time period t with

$$V = H(t)^{sk_0} \prod_{i=1}^{n} c_i = g^{\sum_{i=1}^{n} x_i}, \tag{3.4}$$

The actual aggregate value $\sum_{i=1}^{n} x_i$ must be then found by finding the discrete logarithm of V, either by using brute force or other suitable methods, like Pollard's rho algorithm.

The scheme also supports differential privacy. Each user adds some noise $r_{i,t}$ to their private value to obtain $\hat{x_{i,t}} = x_{i,t} + r_{i,t}$, where $r_{i,t}$ is sampled from a symmetric geometric distribution. This results in a differentially-private scheme where the accumulated noise in the final statistic is minimal.

A clear drawback of this scheme is that the input space must be small enough, for the decryption of the aggregate to run in polynomial time. Additionally, it is not fault-tolerant since the aggregation can only be performed correctly when every user participates in the protocol.

Lu et al. The authors of [58] present a privacy-preserving data aggregation protocol with support for multidimensional data whose communication complexity is linear in the number of users and, thus, it does not depend on the number of dimensions. The scheme is targeted at smart grids. Similar to many smart grid aggregation schemes, the model includes a set of n honest-but-curious users, an honest-but-curious aggregator, and a fully-trusted operation center. The main novelty of the scheme is the adoption of a superincreasing sequence of large prime numbers:

$$\boldsymbol{a} = (a_1 = 1, a_2, \ldots, a_l), \tag{3.5}$$

with $\sum_{j=1}^{i-1} a_j \cdot w \cdot d < a_i$, for $1 \leq i \leq l$, where w is the maximum number of users in a residential area, l is the number of dimensions supported by the protocol, and d is the maximum size of a single datum, for example, a 32-bit number. The Paillier cryptosystem is used to encrypt the individual reports from the smart meters and its homomorphic properties are used to aggregate the ciphertexts. A ciphertext looks as follows:

$$c_i = g^{a_1 d_{i1}} \cdot \cdots \cdot g^{a_l d_{il}} \cdot r_i^n = g^{a_1 d_{i1} + \cdots + a_l d_{il}} \cdot r_1^n \mod n^2, \tag{3.6}$$

where $r_i \leftarrow\!\!\$\, \mathbb{Z}_n^*$ is a random number chosen by user i, and d_{ij} indicates the j^{th} dimensional value of user i. As a result, c_i is a normal Paillier ciphertext that, however, hides l values each split by a prime number a_i. Once all ciphertexts are aggregated by the aggregator and decrypted by the operation center, the resulting plaintext value is:

$$M = a_1 \sum_{i=1}^{w} d_{i1} + \cdots + a_l \sum_{i=1}^{w} d_{il} \mod n. \tag{3.7}$$

In order to extract the individual aggregates for each dimension, notice how $M_{l-1} = M \mod a_l$ and the l-dimensional value can be extracted using the following equation:

$$\sum_{i=1}^{w} d_{il} = \frac{M - M_{l-1}}{a_l} = \frac{a_l \sum_{i=1}^{w} d_{il}}{a_l}. \tag{3.8}$$

The same procedure can be used to extract the remaining aggregates. While this scheme is very efficient communication-wise, since l different values can be packed into a single one before submission, it has, nonetheless, several drawbacks. The authors only assume external adversaries that eavesdrop on the communication channels, while all internal parties are assumed to be honest but curious. As such, integrity can only be guaranteed if the users and the aggregator act honestly and are not compromised. Additionally, collusions are not allowed otherwise the confidentiality of the private reports may be compromised. Finally, a trusted setup, performed by the operation center, is required, which may discourage the usage of these schemes for applications other than smart grids. We also note that another method for multidimensional data packing relies on the Chinese Remainder Theorem, an example of which can be found in [69].

Erkin and Tsudik In [28], the authors introduce a scheme to aggregate smart grid measurements both spatially and temporally. Spatial aggregation refers to the aggregation of all user measurements at any given aggregation round, which is the type of aggregation all schemes introduced until now adopt. Temporal aggregation, on the other hand, refers to the aggregation of several measurements from a single user, taken at predefined intervals. The protocol assumes an honest-but-curious adversarial model, where all participants honestly follow the protocol but may try to infer private information about other participants. The protocol is also resistant to up to $n - 2$ collusions, where n is the number of users. Paillier encryption is adopted for its homomorphic properties and instantiated using a trusted authority. However, both the public and private keys are made public. The reason for this choice is to allow any party that is authorized to learn the total energy consumption to decrypt the aggregate. Nonetheless, individual users must not be able to decrypt individual measurements. To overcome this, each smart meter shares a random seed with every other user in the system. This seed is used to initialize a pseudorandom number generator and to generate random masks. Then, in round t, each user i generates a random value $r_{i,t}$:

$$r_{i,t} = n + \sum_{i=1, i \neq j}^{N} r_{i \to j,t} - \sum_{j=1, i \neq j}^{N} r_{j \to i,t}, \tag{3.9}$$

where $r_{i \to j,t}$ indicates a random number generated in round t with the seed sent by user i to user j. Intuitively, when all $r_{i,t}$ values are added together, we are left with $N \cdot n$, since the random masks add up to zero. With $r_{i,t}$, each user can create a Paillier ciphertext for its private measurement:

$$c_{i,t} = g^{x_{i,t}} \cdot H(t)^{r_{i,t}}, \tag{3.10}$$

where $H : \{0,1\}^* \to \mathbb{Z}_n^*$ is a secure hash function. Thus, the private measurements are now blinded by some randomness and only the final spatial aggregation can be decrypted by whichever party possesses the decryption key.

Temporal aggregation, which is useful for billing purposes, can be realized in two ways. The first method is by having each smart meter send its encrypted measurements to the utility supplier, which, in turn, only aggregates them after a certain number of measurements have been received, say M. Each measurement is blinded with noise that cancels out when M successive measurements are aggregated to prevent the supplier from decrypting the individual measurements. In the second method, the smart meter itself periodically reports its temporally-aggregated measurement to the supplier encrypted with the public key of the trusted smart meter manufacturer. As such, the manufacturer is required to decrypt the aggregation for the supplier.

The scheme requires a trusted entity to recover in case of a smart meter malfunction. Additionally, in the case of temporal aggregation, the intervention of the manufacturer is also required to decrypt the aggregated measurements when using the second method. The first method has, however, a higher space complexity for the utility supplier. We further note that in [88], the authors present a similar scheme that also adds support for public verifiability by letting each participant store a copy of the ciphertexts they receive and allowing them to query other users in the same neighborhood to ensure the value published by the aggregator was correct.

Bonawitz et al. In [7], the authors propose a round-based scheme for secure data aggregation for distributed machine learning. Their scheme supports high-dimensional data and ensures security even in the face of user dropouts, which is desirable for practical implementations. This means that the scheme is robust against users randomly leaving the protocol during any round of the protocol.

The basic protocol consists of four rounds. There are n users that can drop out at any point during the protocol and an aggregation server. The users do not communicate with each other directly but, rather, via the server using an authenticated channel.

There is a setup phase in which each party is given the public parameters. During the first round, each user generates two Diffie-Hellman key pairs (c_u^{PK}, c_u^{SK}), which is used to securely communicate with other users, and (s_u^{PK}, s_u^{SK}), which is, instead, used to agree on pairwise seeds that are later used to generate zero-sum masks. The users send their public keys are sent to the server, which then broadcasts them to all users in the next round. In the second round, each user randomly picks a random seed $\boldsymbol{b}_u$, to be used with a random number generator, and uses (t, n)-secret sharing to generate n shares of both b_u and s_u^{SK} for all users. Each share is encrypted with the public key of every other user and sent to them via the aggregator. In the third round, each user u masks their inputs using the output from a pseudo-random number generator initialized with seeds $\boldsymbol{s}_{u,v}$ shared with all other $n-1$ users v. The masks are computed in such a way that they all sum up to zero when added together:

$$\boldsymbol{y}_u = \boldsymbol{x}_u + \text{PRNG}(\boldsymbol{b}_u) + \sum_{v \in \mathbb{U}: u<v} \text{PRNG}(\boldsymbol{s}_{u,v}) - \sum_{v \in \mathbb{U}: u>v} \text{PRNG}(\boldsymbol{s}_{v,u}) \mod p. \tag{3.11}$$

In this equation, $\boldsymbol{x}_u$ is the value that user u wants to hide, and $\text{PRNG}(\boldsymbol{b}_u)$ is used to ensure that a lying server cannot trick other users into sharing their shares of some user u. If all users contribute, then only the $\text{PRNG}(\boldsymbol{b}_u)$ masks of every user remain and they can be removed by letting the aggregator ask the users for t shares of $\boldsymbol{b}_u$. Alternatively, if a user u drops out, then the aggregator only needs to worry about collecting t shares of s_u^{SK} to reconstruct and remove the missing masks. However, honest users only reply to one request: either a share of $\boldsymbol{b}_u$ or one of s_u^{SK}, but not both. Thus, privacy and fault-tolerance are guaranteed as long as less than $1/3$ of the users are malicious and collude with the aggregator. The authors also present some extensions to the scheme that guarantee its security in the presence of malicious internal adversaries, but they only focus on the privacy of the input values and not on the correctness of the final result, which cannot be guaranteed in this stronger model. Additionally, the solution involves verifying many signatures that are linear in the number of users, leading to a high computational cost.

The protocol has, additionally, another main drawback. It is highly interactive as, in every round, all users have to re-generate the keys and share secret shares of them with everyone else. Consequently, this also leads to a quadratic communication and storage overhead for the aggregator.

Zhuo et al. In [89], the authors propose a privacy-preserving and verifiable data aggregation scheme for mobile crowdsourcing. The adopted system model involves a requester that outsources data aggregation tasks that are executed by a set of workers, usually mobile devices. To reduce the computational burdens of mobile devices, a cloud server is introduced to perform most of the computation. The system is initialized by a trusted authority that handles the registration of workers, the generation of the public parameters, and key distribution. All internal parties, i.e. the requester, the workers, and the cloud server, are assumed to be honest but curious and collusion attacks are not considered. The main goal of the scheme is to protect the identity and privacy of the workers, to allow the requester to offload a data aggregation task, and to verify the correctness of the computation in case the integrity of the submitted values is affected during transmission. The data aggregation scheme is based on BGN encryption, which offers both additive and multiplicative homomorphic operations in the ciphertext domain. The ciphertexts are encrypted using the public key of the requester and submitted by the workers to the cloud server. The cloud server handles the heavy data aggregation tasks. Identity privacy is achieved using ring signatures, while verification of the computation is achieved using special hash functions computed over the ciphertexts that can be verified individually by the cloud server and in an aggregated fashion by the requester using bilinear maps. The authors provide extensions that allow the computation of several aggregation functions such as sum, mean, variance, and uncentered correlation coefficient.

3.2. Privacy-preserving data aggregation with malicious users

In this section, we present an overview of data aggregation schemes that allow users to behave maliciously. More specifically, these users may submit malformed data that is outside of the expected range of values and may either cause an incorrect statistic to be computed by the aggregator or even make the statistic impossible to compute. The common solution to prevent these types of attacks is to ensure the values submitted by the users are well formed, for example, by ensuring they lie in a valid range. Zero-knowledge proofs, like bulletproofs, represent a straightforward technique to perform this task. Here, instead, we present some schemes from the literature that adopt different techniques.

Fan et al. The authors of [44] propose a scheme for mobile sensing that is privacy-preserving and can withstand malicious users who attempt to poison the aggregation with invalid data. In their model, the aggregator is honest-but-curious and, as such, it follows the protocol but it may attempt to infer private information. The users can be malicious and collude with each other to cause a disturbance in the final statistic. Additionally, a trusted authority is responsible for the distribution of all key material to the users and the aggregator. The general technique used by the protocol is to restrict the range of values that each user can submit. A data vector $D = [d_1, \ldots, d_w]$ is defined during the setup phase which contains all the allowed values that can be submitted. Each user, in this case, mobile sensing devices, picks the value that is closest to the sensed value from the data vector. Next, the individual values are blindly signed by the server, using its RSA secret key, so that the users cannot change the values afterward. Then, the users encrypt their reports and send them to the aggregator. The decryption key of the aggregator k_0 is set to be equal to additive inverse of the sum of all user keys, i.e. $k_0 = \sum_{i=1}^{n} k_i$ with k_i being the encryption key of user i. Thus, the aggregator can only decrypt the final aggregate, but not the individual reports. As in [79], the aggregator must solve a discrete logarithm to extract the statistic. To validate the signed reports, the aggregator constructs a validation vector V from D, using its RSA secret key. Each user multiplies its signed report with the corresponding validation value in V to get $\Delta = (H(t)x_i)^d$ and then sends this to the aggregator. The aggregator, knowing d, x_i, and $H(t)$ can verify whether Δ is valid or not and, in the latter case, report the user.

The scheme has, however, some drawbacks. First, the integrity of the statistic can only be guaranteed if we assume that the aggregator is honest-but-curious. Second, the aggregator must solve a discrete logarithm to extract the aggregate, which is an expensive operation and limits the size of plaintext space. Additionally, only a selected number of values can be submitted, which may not be suitable for situations in which more fine-grained data are needed. Finally, other arbitrary behavior

from the users, such as submitting random ciphertexts, is not tolerated.

Karakoc et al. The authors of [48] propose a secure data aggregation scheme to counter malicious users that input incorrect data into the aggregation protocol. The paper mainly focuses on the use of data aggregation to train machine learning models in a federated learning scenario. The primary goal of the protocol is to prevent model poisoning and backdoor injection attacks. Model poisoning occurs when users send arbitrary data that renders the trained model unusable, whereas backdoor injection refers to a technique whereby malicious users send carefully-crafted model updates that do not affect the trained model performance if not for a selected set of inputs, for which the model outputs values of the malicious user's choosing. In the paper, the aggregator is assumed to be honest but curious while the users are malicious with respect to the correctness of the data they submit. The solution proposed by the authors is to enforce the input values of each user to be in a specific interval that minimizes the chance of either type of attack happening. This scheme is reminiscent of other data aggregation schemes with range validation, with the main novelty being that the solution is not based on zero-knowledge proofs. The scheme uses the aforementioned PUDA scheme [54] as a sub-protocol to compute the final statistic, which is the sum of all input values, as well as to check the validity of the result. To check whether the result was computed from inputs belonging to a legitimate range, each user engages in an interactive protocol with the aggregator to compute a verification tag compatible with the verification procedure of the PUDA scheme. These tags are valid only if the input values of every user belong to a legitimate interval, otherwise, they are random and will cause the verification to fail. These tags are computed by each user through an interactive protocol with the aggregator based on an Oblivious Programmable Pseudo-random Function (OPPRF). In short, for each bit of the input data, the user will engage in a special oblivious transfer with the aggregator after which the user will receive a piece of the final verification tag based on the bit value obliviously sent to the aggregator. The protocol is set in such a way that if the bit sequence corresponds to a number that is smaller than the threshold, then the resulting verification tag is valid, otherwise, it is random.

The scheme uses the aforementioned PUDA scheme [54] as a sub-protocol and, as such, it inherits all of its drawbacks. Additionally, the scheme is highly interactive, since it requires nl oblivious transfers, where n is the number of users and l is the maximum bitlength of the input values. Furthermore, verification only indicates that possibly malicious behavior occurred during the execution of the protocol. However, the scheme does not allow the aggregator to identify exactly which users acted maliciously. Finally, the aggregator is assumed to be honest but curious during the execution of the protocol.

Kursawe et al. In [51], the authors also propose another class of schemes they call *comparison* schemes. These rely on the aggregator already knowing the approximate value V that the aggregate sum will be equal to. In these protocols, the aggregate is hidden as a discrete logarithm $g^{\sum_{i=1}^{n} x_i}$ and, thus, the aggregator must explore the plaintext space around the aggregate until it is found to roughly match the expected value V. If after a bounded number of trials, this is not the case, then one or more users probably sent an incorrect value for aggregation. The main drawback of the scheme is that the aggregator cannot directly identify which users acted maliciously during a round. The aggregator only knows that something possibly went wrong during the execution of the protocol, but not where.

3.3. Privacy-preserving data aggregation with a malicious aggregator

In this section, we summarize some works that assume the presence of a malicious aggregator that may attempt to publish an incorrect aggregation result. The users, however, are assumed to be honest but curious and do not collude with the aggregator or otherwise act dishonestly.

Li et al. In [56], the authors propose a data aggregation scheme for wireless sensor networks (WSN). WSNs are networks of low-powered devices used for several monitoring scenarios, such as military surveillance and wildfire tracking. A key aspect of WSNs is any algorithm that runs on them must be energy efficient and this is one of the objectives of this scheme. Additionally, the scheme allows for the identification of malicious aggregators that tamper with the aggregation results. The scheme focuses on in-network aggregation, where the participants are arranged in a tree-like structure with the source

nodes being the leaves of the tree and the aggregators being the remaining internal nodes, and the root being the final node where the full aggregation result is computed. Data aggregation starts from the leaf nodes and proceeds toward the root. At each aggregator node, the aggregate of its children is computed and sent upward. Each sensor node can verify whether its parent has computed the correct aggregate value by re-computing the aggregate using the values from its siblings and comparing it with that output by its parent. If an inconsistency is found, the parent node is reported to the whole network. The main drawback of the scheme is that it is not privacy-preserving. The individual values submitted for aggregation can be directly inspected by other nodes during the verification phase. Therefore, while for many WSN applications privacy is not required, this scheme cannot be used in scenarios privacy preservation is of paramount importance. We mention this scheme in this section because, to the best of our knowledge, it is one of the earliest ones to tackle the problem of data tampering at the hand of the aggregator.

Leontiadis et al. (2015) The authors of [54] were among the first to propose a publicly-verifiable private data aggregation scheme to defend against a malicious aggregator. Malicious, in this context, means that it might provide a bogus aggregate result instead of the correct one computed from the inputs submitted by the users. To defend against this, the aggregator is required to provide a proof of correctness in addition to the aggregate value. The users are assumed to be honest but curious and trusted to submit the correct input values. The aggregation scheme is based on the scheme presented in [79], which is augmented with a verification protocol. For this purpose, in addition to *aggregator obliviousness*, the authors propose a new security notion they call *aggregate unforgeability*, which guarantees that an aggregator cannot forge a valid proof for an aggregate sum that was not computed correctly from the original users' inputs. In addition to the users, which provide their private data, and the aggregator, which aggregates the inputs, the scheme also consists of a trusted key dealer, which performs a setup in the beginning and then goes offline, and a semi-trusted data analyzer whose task is to collect the aggregate values from the aggregator and verify their correctness. No party should learn anything about the private inputs of the participating users, other than their aggregate value. The basic aggregation scheme is identical to [79], as summarized above, thus here only the verification protocol will be described. It consists of four main phases: setup, tag encryption, aggregation, and verification. In the setup phase, the trusted key dealer generates the public parameters and secret keys, which are distributed among the participants. The public parameters include a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, a cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}_1$, and random generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$. Each user is handed a tag key tk_i and a secret key a, which is shared by all users. The verifier receives the verification key $vk = (g_2^{\sum_{i=1}^{n} tk_i}, g_2^a)$. During submission, each user creates a tag

$$\sigma_{i,t} = H(t)^{tk_i}(g_1^a)^{x_{i,t}}, \tag{3.12}$$

and sends it to the aggregator. The aggregator then computes a global tag:

$$\sigma_t = \prod_{i=1}^{n} \sigma_{i,t} = H(t)^{\sum_{i=1}^{n} tk_i}(g_1^a)^{\sum_{i=1}^{n} x_{i,t}}. \tag{3.13}$$

Finally, given a sum $\sum_{i=1}^{n} x_{i,t}$, the verifier can check the following equation to ensure the aggregation was performed correctly:

$$e(\sigma_t, g_2) \stackrel{?}{=} e(H(t), g_2^{\sum_{i=1}^{n} tk_i}) e(g_1^{\sum_{i=1}^{n} x_{i,t}}, g_2^a). \tag{3.14}$$

Similar to [79], a drawback of this scheme is the limited input space, since decrypting the final aggregate value requires solving a discrete logarithm. The scheme is also not fault-tolerant. Most importantly, the scheme is not collusion resistant. Assuming a user u_i colludes with the aggregator by revealing its secret g_1^a to the latter, then the aggregator can forge a bogus tag from a valid one like so. Several other protocols are based on PUDA such as [48, 27, 53]. In [27], the authors extend the PUDA protocol to prove its security under simpler hardness assumption which, in practice, can translate to keys of smaller size, but also a more complex protocol.

Bakondi et al. Another protocol for time-series data with public verifiability is presented in [5]. Similar to the previously mentioned works, the statistic this paper focuses on is the sum. The scheme is mostly

non-interactive, except for a trusted setup at the beginning and the uploading of the input data to the aggregator's server. The aggregator is assumed to be malicious in the adversarial model adopted by the authors. The users are trusted to provide the correct input values and every registered user is required to participate in the protocol for the final aggregation to be computed correctly.

The scheme uses homomorphic signatures which can be combined by the aggregator to produce a publicly verifiable output, from which the aggregated sum can also be extracted. Bilinear maps are used for verification, thus we assume a pairing function $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Each signature must be encrypted first, as the verification procedure of the scheme also permits verifying individual signatures, which allow an adversary to extract the signed value in polynomial time. Thus, each user i is given a secret key $sk_i = (k_i, \alpha)$, with $k_i \leftarrow\!\$\, \mathbb{Z}_p$ and $\alpha \leftarrow\!\$\, \mathbb{Z}_p$, while the aggregator is given $k = \sum_{i=1}^{n} k_i$. Each user i can compute an encrypted signature as follows:

$$c^{(i)} = H_1(id)^{k_i} \left(\prod_{j=1}^{n} H_2(id, j)^{m_{N+j}^{(i)}} \prod_{u=1}^{N} g_u^{m_u^{(i)}} \right)^{\alpha} \in \mathbb{G}_1, \tag{3.15}$$

where H_1, H_2 are secure hash functions mapping to $\mathbb{G}_1$, id is the round identifier, g_u is a public generator, each $m_{N+j}^{(i)}$ is 0 unless $j = i$, in which case it is 1, and $m_u^{(i)}$ is the private data to be signed. The secret key of each user also includes α, which is a secret exponent that is shared by all users and that must be kept away from the aggregator. It ensures that the aggregator cannot modify any of the signatures. Next, the aggregate signature can be extracted:

$$\sigma(\mathbf{M}, id) = H_1(id)^{-k} \prod_{i=1}^{n} c^{(i)} = \left(\prod_{j=1}^{n} H_2(id, j)^{M_{N+j}} \prod_{u=1}^{N} g_u^{M_u} \right)^{\alpha} \in \mathbb{G}_1, \tag{3.16}$$

where $\mathbf{M}$ indicates the multidimensional sum resulting from the aggregation. It must be noted that, without proper care, the aggregate signature of this scheme is malleable, as a malicious aggregator can simply compute σ^2 to produce a forgery. To counter this type of attack, each user i adds a vector of n bit values to its private datum, one for each user in the system. Before submitting the signature, user i sets the i^{th} bit value to 1. During the evaluation procedure, the verifier ensures that the signature was created from exactly n input values by checking whether the vector contains n 1's. Trying to tamper with a signature would invalidate this condition. Verification can be performed using the properties of bilinear maps, the computed result $\mathbf{M}$, and a public parameter $h^{\alpha} \in \mathbb{G}_2$ by ensuring the following equation holds:

$$e(\sigma(\mathbf{M}, id), h) \stackrel{?}{=} e(\prod_{j=1}^{n} H_2(id, j)^{M_{N+j}} \prod_{u=1}^{N} g_u^{M_u}, h^{\alpha}), \tag{3.17}$$

and ensuring the last n bit values of $\mathbf{M}$ are all 1's.

The scheme has a few drawbacks. The message space of this protocol must be small for the decryption of the aggregate to run in polynomial time. This is common among many similar schemes that use discrete logarithm-based primitives. The scheme is not fault-tolerant and requires a full trusted setup to be executed whenever a user joins or leaves the system. Finally, collusion is not considered in the threat model adopted by the authors.

The drawback of their scheme is that both the aggregator and the users are assumed to be honest but curious. Collusions between users and the aggregator are allowed, but only to infer information about the remaining honest users.

Ni et al. In [65], the authors introduce a data aggregation scheme for smart grids that protects the confidentiality, integrity, and authenticity of private metering data despite a malicious aggregator. The system model consists of a trusted operation center, which also handles the initial setup, a set of trusted users, and a malicious aggregator, which may attempt to modify the aggregate or infer information about the metering data of individual smart meters. The private data is encrypted using the Paillier cryptosystem with the public key of the operation center. As such, the aggregator can only collect and aggregate the encrypted data but it cannot decrypt the aggregate. Additionally, homomorphic signatures are created using commitment-like authentication tags of the form $\sigma_i = (H(ID_i \parallel t) g^{a_i} u^{m_i})^{\alpha}$, where $H(ID_i \parallel t)$ is the cryptographic hash of a user's identifier in aggregation round t, a_i is a secret

exponent computed from the user's secret key using a pseudorandom function, and m_i is the private datum. Similar to other schemes with malicious aggregators like [5, 54], each signature is further blinded using an exponent α shared by all users to prevent tampering from external adversaries or the aggregator. Finally, the operation center can decrypt the aggregate computed by the aggregator using its Paillier secret key, and verify whether the signature matches the aggregate using the properties of bilinear maps and the verification key computed during a setup phase in the beginning. If the verification fails, the operation center retrieves all the individual reports from the smart meters to identify the corrupted reports.

The scheme cannot tolerate collusions between a single user and the aggregator both with respect to the confidentiality of the private values and the integrity of the aggregate.

3.4. Privacy-preserving data aggregation with malicious users and aggregators

In this section, we summarize two schemes whose goal is to achieve confidentiality as well as integrity and authenticity of the aggregate statistic despite the presence of both a malicious aggregator and malicious users.

Leontiadis et al. (2021) The authors of [53] propose a publicly-verifiable protocol for private data aggregation with both malicious users and a malicious aggregator. In particular, in their model, the aggregator is allowed to collude with malicious users to forge the verification tag for a message of an honest user which, consequently, invalidates the final result of the aggregation. The users of the protocol are still assumed to submit genuine input data and not fake values.

The actual aggregation scheme is based on the scheme presented in [79], and is executed separately from the tagging scheme, which is used to compute the verification material. We now proceed to describe the latter.

The scheme works in a pairing group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order p, for which an efficiently-computable pairing function $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and a secure hash function $H\colon \{0,1\}^* \to \mathbb{G}_1$ are assumed to exist. During the setup phase, a trusted authority generates a random key $r \leftarrow\!\!\$\, \mathbb{Z}_p$, which is sent to a honest-but-curious party called the Converter, and a generator $w \leftarrow\!\!\$\, \mathbb{G}_2$, which is distributed to all users. Each user also generates a random value $r_i \leftarrow\!\!\$\, \mathbb{Z}_p$, and sends it to the trusted authority. The trusted authority finally generates the private verification key:

$$vk = (vk_1, vk_2, vk_3) = (w, r, \sum_{i=1}^{n} r_i), \tag{3.18}$$

A user i computes a metatag for an aggregation round t as follows:

$$mtag_{i,t} = (mtag^1_{i,t}, mtag^2_{i,t}) = ([H(t)^{r_i} g^{x_{i,t}}]^{tk_i}, w^{\frac{1}{tk_i}}), \tag{3.19}$$

where tk_i is randomly generated once by the users during the setup phase, and $x_{i,t}$ is the user's input value to be tagged. Each $mtag_{i,t}$ is sent to the converter along with a zero-knowledge proof of commitment opening. This is done to ensure that $mtag^1_{i,t}$ is of the form $h^a g^b$. We note that the proof presented in this paper allows for the exponents of the tag to be 0. If the proof concludes successfully, the converter uses r to further randomize the metatag:

$$st^1_{i,t} = e(mtag^1_{i,t}, mtag^2_{i,t})^r, \tag{3.20}$$

$$st^2_{i,t} = (mtag^1_{i,t})^r. \tag{3.21}$$

The converter, then, sends $(st^1_{i,t}, st^2_{i,t})$ back to the user. The user can then inspect the tag for a possible forge by checking

$$e(st^2_{i,t}, w)^{\frac{1}{w}} \stackrel{?}{=} st^1_{i,t}. \tag{3.22}$$

If the equation holds, the tag was not tampered with and $st^1_{i,t}$ is the final tag for user i. The aggregator combines all tags together by computing $\sigma_t = \prod_{i=1}^{n} st^1_{i,t}$ and sends $(\sum_{i=1}^{n} x_{i,t}, \sigma_t)$ to the data analyzer for verification. Here, $\sum_{i=1}^{n} x_{i,t}$ is computed using the aggregation protocol presented in [79].

Finally, the data analyzer can check the validity of the result by verifying the aggregate signature against the received sum using the private verification key.

The scheme suffers from several drawbacks. Firstly, it relies on a semi-trusted party during every round to ensure the unforgeability of the verification tags. Moreover, if the malicious aggregator, a malicious user, and the converter were to collude with each other at the same time, the scheme cannot guarantee unforgeability anymore. We also found that malicious users can send a malformed tag to the converter where $[H(t)^{r_i} g^{x_{i,t}}]^{tk_i}$, with either $r_i = 0$ or $x_{i,t} = 0$, from which they can then compute g^r and send (g^r, w) to the aggregator. The aggregator can use this value to output tags for any values of its choosing. Thirdly, the scheme is not fault-tolerant so every user must contribute in every round. Users are also entrusted to submit well-formed values. If a malicious user sends a random tag to the aggregator, the verification procedure fails and there is no mechanism to detect which user acted dishonestly. Additionally, the scheme must assume a trusted external verifier, as the verification key contains secret information and, thus, cannot be public. Finally, as with all other discrete logarithm-based schemes mentioned so far, the message space of the input values must be small for the aggregator to decrypt the final result in polynomial time.

Mouris and Tsoutsos In [63], the authors propose a private data aggregation scheme that supports the computation of several statistics such as the sum, average, histograms, and even categorical data. The scheme is also publicly verifiable and can detect users who attempt to poison the aggregate. The system model consists of a set of users, some of which may be malicious and submit invalid data. A curator takes on the role of the aggregator and is assumed to be possibly malicious. However, it does not possess the secret key required to decrypt the aggregate. Another party, the analyst, is instead responsible for decrypting, verifying, and publishing the final statistic. Finally, a trusted ledger is used to store special commitments from the users.

Each user encrypts their private values using the Paillier cryptosystem, then commits to the ciphertext using an RSA-based multiplicative commitment scheme introduced in the same paper, which is then published on a ledger. Additionally, each user computes a zero-knowledge proof to prove that its private value belongs to a valid range, and it is sent to the curator along with the ciphertext. The curator then verifies the proof to ensure the ciphertext is over valid data. Once all ciphertexts are received by the curator, the encrypted aggregate statistic is published by the curator to the analyst. Finally, the analyst collects all commitments from the ledger and aggregates them by computing their product to produce a commitment over the aggregate. Finally, the analyst decrypts the aggregate and verifies its validity using the commitment.

The main drawback of the scheme is that it is not collusion resistant. Indeed, if the analyst and curator collude then confidentiality of the private data cannot be guaranteed. Furthermore, users colluding with the curator may still affect the correctness of the aggregate. Moreover, the scheme relies on a trusted ledger which may not always be available or may be expensive to use and store large amounts of data on.

3.5. Concluding remarks

In this chapter, we have presented several works from the related literature on privacy-preserving data aggregation. The works range from those working in the honest-but-curious model to others that assume malicious behavior other than the users or the aggregator. To the best of our knowledge, only two schemes appear to work in a model in which both the aggregator and the users may be malicious, not only with respect to confidentiality but also integrity. However, both have several downsides. In [53], the system model assumes a semi-trusted third party and only allows for pairwise collusions between this party and the aggregator or the users. On the other hand, in [63], the authors present a protocol packed with many desirable features, however, it also works in a slightly more complex system model where the aggregator or the users may be both independently malicious but cannot collude.

For these reasons, there is a need for a simple and practical private data aggregation scheme that can ensure the integrity and authenticity of the computed statistic in the presence of a malicious aggregator and malicious users, that works in a stronger model without additional semi-trusted parties, and that is efficient in practice.

4

Publicly Verifiable Data Aggregation Against Internal Adversaries

In this chapter, we describe a privacy-preserving and publicly-verifiable protocol to compute the sum over a set of privately-held values. We introduce two separate schemes that are executed sequentially. The first computes an aggregate signature over the sum of the private inputs, by taking advantage of the homomorphic properties of Pedersen commitments. The second scheme is used to compute the plaintext sum itself. If the space of plaintext messages is small enough, say in $[0, 2^{32})$ or smaller, it is possible to extract the sum from the aggregate signature itself in polynomial time. However, this is not very efficient in practice because it requires brute force or a large lookup table. As such, the second protocol provides a faster way to compute the sum.

We first introduce the notation adopted throughout this work to describe the protocols, then we summarize the system model and assumptions, then we proceed to describe the two schemes in more detail.

Notation

Symbol	Meaning
n	The number of users
k	The number of malicious users the system can tolerate
t	An integer identifier denoting a particular aggregation round
$x_{i,t}$	The private value submitted by user u_i during round t
p	A large prime number indicating the order of a group
$\mathbb{G}_i$	A a group of prime order p
$\mathbb{Z}_p$	The set of integers modulo p
u_i	Identifier for the i-th user, $1 \leq i \leq n$
$\mathbb{U}$	The set of all users
$\mathcal{U}_k^i$	The signing set of user u_i
$[s]_i$	The secret share of user u_i
$[s]_i^*$	The secret share of user u_i after combining it with the Lagrange coefficients. Adding k distinct $[s]_i^*$ together allows one to recover s
$\sigma_{i,t}^1$	The initial partial partial signature of user u_i in round t
$\sigma_{i,t}^{2,j}$	The second partial partial signature of user u_i in round t after the secret share $[s]_j^*$ of user $u_j \in \mathcal{U}_k^i$ is added to it
$\sigma_{i,t}^3$	The third partial signature of user u_i in round t after all of its k partial signatures $\sigma_{i,t}^{2,j}$ have been aggregated together
$\sigma_{i,t}$	The final signature of user u_i in round t after u_i adds its secret share $[s]_i^*$ to it
σ_t	The aggregate signature for round t
$\leftarrow\!\!\$$	A random sampling operation
$\mathcal{G}_i$	The i-th group (mPVAS+ extension)

Symbol	Meaning
$\mathcal{G}_{u_i}$	The group of user u_i (mPVAS+ extension)
ZKPNeqZero$(\sigma^1_{i,t})$	A ZKP of inequality to zero of $\sigma^1_{i,t}$
$DH(i,j)$	Diffie-Hellman key agreed between users u_i and u_j
$\boldsymbol{s}_{i,j}$	Shared seed between users u_i and u_j
PRNG$(\boldsymbol{s}_{i,j}, t)$	The t-th pseudorandom integer output by the PRNG seeded with $\boldsymbol{s}_{i,j}$
$c_{i,t}$	PPDA ciphertext of user u_i during round t

4.1. System model and assumptions

The system consists of a set of n users $\mathbb{U} = \{u_1, \ldots, u_n\}$, an aggregator, a trusted dealer, and one or more verifiers. In any given aggregation round t, each user u_i possesses a private integer value $x_{i,t}$. The goal of the protocol is to compute the sum of all private values in round t such that the following properties hold:

- **Confidentiality:** No unauthorized party should be able to learn the private value of a user.
- **Unforgeability:** The aggregator should not be able to publish an authenticated sum that is not equal to the sum of the values submitted by the users.
- **Availability:** Malicious users that attempt to disrupt the execution of the protocol by causing the verification to fail are detected and removed so that the system can continue to operate properly.

In the protocol presented in this chapter, we assume the availability property is not affected. We drop this assumption in Chapter 5, where we present an extension to tackle malicious behavior from the user with respect to availability. We further note that unforgeability is a property of signature schemes that also implies integrity and authenticity of the signed message [81], which is the research goal of this thesis.
We now define three types of behavior:

- **Fully trusted:** A fully trusted party does not deviate from the protocol and does not attempt to affect any of the three properties.
- **Honest-but-curious:** An honest-but-curious party also correctly follows the protocol but may attempt to infer information about the private values of other users using previously-held information or information from messages obtained legally during the execution of the protocol. An honest-but-curious user does not collude with other users.
- **Malicious:** A malicious user may actively try to deviate from the protocol in order to learn information about the private value of other users or to affect the unforgeability and availability properties of the protocol. A malicious user may collude with the aggregator.

4.1.1. Participating parties

In this section, we provide an overview of all parties that participate in the protocol. Table 4.2 summarizes the adversarial assumptions about each entity. All adversaries are assumed to be probabilistic and polynomially bounded.

Aggregator

The aggregator is the party that is tasked to collect the encrypted inputs and signatures from all users, aggregate them, and publish the plaintext result of the aggregation as well as an aggregate signature proving the correctness of the result to a verifier. The aggregator is assumed to be malicious with respect to confidentiality and unforgeability. This means that the aggregator may deviate from the protocol and collude with other malicious users in order to learn the private values of honest-but-curious users or to tamper with their signatures to output an authenticated aggregation result of its choosing. The aggregator does not, however, actively try to disrupt the protocol, for example, by outputting random values. As such, the aggregator does not attempt to cause the verification of the aggregate to constantly fail and will actively try to identify and report users that, instead, attempt to do so.

Users
The users are a collection of parties that hold some private data over which a statistic, in this case, the sum, is to be computed in a distributed and privacy-preserving manner. No other unauthorized party should be able to learn the private data of a user by participating in the protocol. Additionally, each user is required to submit a special homomorphic signature that, when aggregated with the signatures of other users, allows the verifier to check whether the statistic was computed correctly or whether there was an attempt to falsify it. Each user is assumed to have encrypted and authenticated bidirectional communication channels with the aggregator and the dealer, but not with other users. We also assume that the integrity of in-transit messages is preserved. All users are somewhat time-synchronized and participate in the aggregation at the same time and inherently know what the round identifier t is. When users need to interact with each other, messages have to be relayed through the aggregator. We assume a heterogeneous population of both honest-but-curious and malicious users. In particular, we allow the system administrator to choose the number of malicious users that the protocol should be able to withstand. Throughout the paper, we refer to this threshold as k. The aggregator, although also assumed to be malicious, is not included in this number. While the three main properties analyzed in this work (confidentiality, unforgeability, availability) are guaranteed when $k \leq n - 2$, we assume that in an average, real-world execution of this protocol the number of malicious users will be much smaller. Similar to [53, 5], in the scheme described in this chapter, all users are at least trusted to send valid information and not random or fake information that causes the verification of the result to fail. In Chapter 5, we describe an extension that allows the aggregator to detect malicious users that also attempt to disrupt the execution of the protocol and, thus, its availability.

Verifier
The verifier is a party that verifies whether the aggregation was performed correctly. In short, we say that the aggregation is performed correctly if the statistic is computed only from the values that are originally sent out by all registered users during a given round of the protocol. If a party manages to inject a new value and tamper with the values sent by the registered users, then the verifiers should be able to detect it. We do not put restrictions on which entities are allowed to verify the results of the aggregation: any party holding the public verification key can be a verifier. Verifiers may include external auditors, the system administrator, the aggregator, and internal users.

Dealer
All the protocols described in this work assume a trusted setup that is performed to bootstrap the system. This is an assumption that is also employed by many other similar schemes that work in the malicious model, such as [5, 53, 54, 27, 79]. A trusted setup makes it easier to bootstrap a system such as this one in the presence of malicious participants. While not always ideal, it is a sensible assumption for several practical applications such as smart grids and medical data sharing, where a trusted institution or manufacturer can take on the role of the trusted authority.

A fully-trusted dealer is tasked to generate and distribute the public and private parameters of the protocols to the other involved parties. Once the setup is finished, the dealer goes offline for the remainder of the protocol execution.

Entity	Confidentiality	Unforgeability	Availability
Aggregator	Malicious		Trusted
$n-k$ Users	Honest-but-curious	Trusted	
k Users	Malicious		Trusted
Verifier	Honest-but-curious	-	
Dealer	Trusted		

Table 4.2: The adversarial assumptions for each participant of the protocol.

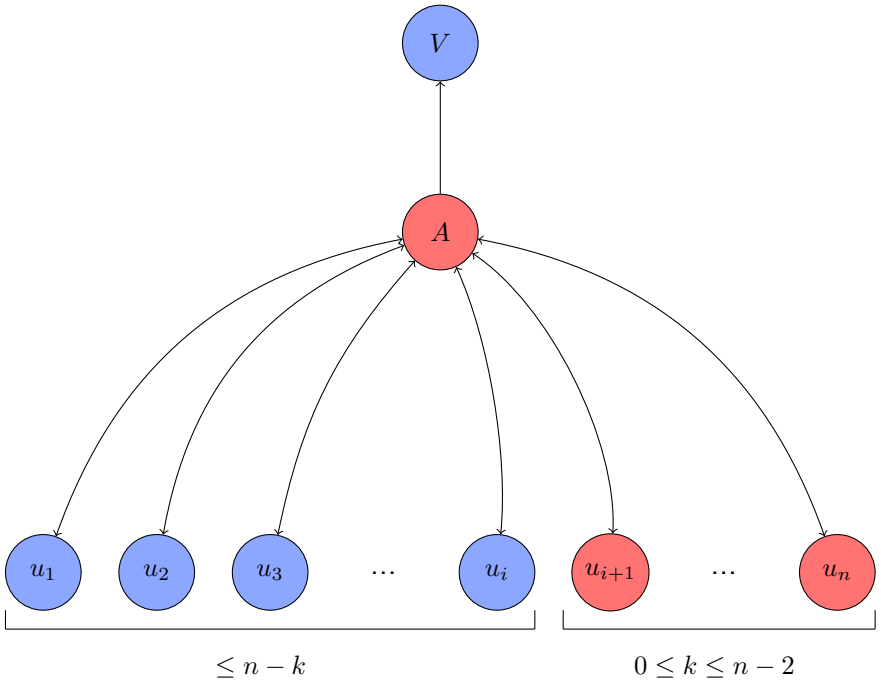


Figure 4.1: Diagram showing the system model assumed by the protocols. The blue nodes indicate honest-but-curious users. They do not collude or deviate from the protocol but may be nonetheless interested in learning the private values of other users. The red nodes indicate malicious users. They can deviate from the protocol and collude with each other to learn the private values of other users or to affect the integrity and authenticity of the result of the aggregation.

4.2. Publicly-verifiable aggregate signatures with malicious users and aggregators (mPVAS)

In this section, we present a protocol that allows users to compute special signatures over their private values that can be aggregated into one single signature over the sum of the private values. The aggregate signature is unforgeable, assuming a bounded number k of malicious users, and allows a verifier that holds the public verification key to verify the integrity and authenticity of the computed statistic. Confidentiality of private values is still preserved throughout the execution of the protocol. The core idea behind the scheme is to create commitment-like signatures and wrap each one of them under a common secret exponent, which we call s. This is a technique used by other schemes with verifiability, for example [5, 54, 57], as also shown in Equation 3.15 and Equation 3.12. However, these schemes assume honest behavior from the users who are all entrusted with the secret. When users behave maliciously and collude with the aggregator by revealing the secret exponent, then the unforgeability of the aggregate signature cannot be guaranteed anymore since the aggregator can now tamper with the individual signatures. We overcome this limitation by splitting the secret exponent s into n secret shares, where n denotes the number of users in the system. We adopt Shamir Secret Sharing with a threshold of $k+1$, where k is the maximum number of tolerated malicious users. When users decide to create a signature for their private input, they start with a partial signature and then collaborate with other users to reconstruct s via interpolation in the exponent. The result is that malicious users and the aggregator, even by colluding with each other, do not have enough secret shares to fully reconstruct the secret s. As a result, the individual signatures of honest users and the final aggregate signature cannot be tampered with without invalidating the end result, which is detected during the verification phase. The underlying structure of our signatures is based on that of the ciphertexts of the data aggregation scheme presented in [79] and is reminiscent of Pedersen commitments. We also adopt the structure of the verification keys as well as the verification procedure of the publicly verifiable PUDA scheme [54]. The computations are performed over a pairing group. Signing is performed in the first group $\mathbb{G}_1$, the verification keys are generated in the second group $\mathbb{G}_2$, and verification is carried over in the target group $\mathbb{G}_T$. Finally, Shamir Secret Sharing and a zero-knowledge proof of inequality to zero are used to construct valid signatures without letting dishonest users learn any information that could allow them to tamper with the result without being detected.

In short, the scheme consists of five phases: setup, signing, signature aggregation, and verification. During the setup phase, users are handed the public and secret information required to engage in the protocol. In the signing phase, each user interacts with other users to construct their final signature. All individual signatures are then aggregated together during the signature aggregation phase. Finally, the verifier can verify whether the output sum was computed correctly by verifying the aggregate signature. We now describe each phase of the protocol in more detail.

4.2.1. Setup

During the setup phase, the trusted dealer chooses and publishes the public parameters $pp = (H, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, k, p)$, generated according to a strong security parameter λ. Each $\mathbb{G}_i$ is a cyclic group order p, where p is a large prime number. g_1 and g_2 are random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a type-3 bilinear map in which the SXDH assumption holds (see Definition 6). $H : \{0,1\}^* \to \mathbb{G}_1$ is a cryptographically-secure hash function. k is the maximum number of malicious users the system is capable of tolerating.

Each user is assigned a unique integer identifier i and broadcast to all other users. For simplicity, we will assume that $i \in \{1, 2, \ldots, n\}$.

The dealer also randomly chooses a secret $s \leftarrow\!\$\, \mathbb{Z}_p$, creates n secret shares $[s]_i$ using $(k+1, n)$ Shamir Secret Sharing. Recall that a secret share consists of a coordinate (x_i, y_i) on the 2D plane. We implicitly assume that the x_i coordinate of user u_i corresponds to its own integer identifier. Thus, $[s]_i$ denotes the secret y_i coordinate of user u_i. These shares are distributed to the respective users u_i. If $k = 0$, then each user receives a copy of s. Next, the dealer then requests a randomly-sampled signature key $sk_i \leftarrow\!\$\, \mathbb{Z}_p$ from each user u_i. Users reply by sending $g_2^{sk_i}$ back to the dealer. When the dealer has received n replies, it computes the verification key

$$vk = \left((g_2^s)^{\sum sk_i}, g_2^s \right), \tag{4.1}$$

which is sent to the verifiers.

4.2.2. Signing

When a user intends to create a signature for its private value, interaction with other users is required. The creation of a signature consists of four steps.

1. Create partial signature

In the first step, a user u_i in round t chooses a random value $r_{i,t} \leftarrow\!\$\, \mathbb{Z}_p$ and computes its initial partial signature $\sigma^1_{i,t}$ as follows:

$$\sigma'_{i,t} = \left(H(t)^{sk_i} g_1^{x_{i,t}} \right)^{r_{i,t}} \in \mathbb{G}_1. \tag{4.2}$$

The reason for the random exponent $r_{i,t}$ is to prevent other, possibly malicious users, from tampering with the signature, since g_1 is public. Next, the user also creates a zero-knowledge proof of inequality to zero to prove that neither exponent of $H(t)$ nor g_1 is equal to zero, i.e.

$$PK\{(sk_i, x_{i,t}, r_{i,t}) : \sigma_{i,t} = H(t)^{sk_i \cdot r_{i,t}} g_1^{x_{i,t} \cdot r_{i,t}} \wedge sk_i \cdot r_{i,t} \neq 0 \wedge x_{i,t} \cdot r_{i,t} \neq 0\}. \tag{4.3}$$

This step is necessary to prevent malicious users from sending malformed partial signatures and obtaining g_1^s, which can be used to tamper with any signature. We denote the information required to prove this condition by *ZKPNeqZero*$(\sigma^1_{i,t})$. Finally, the user sends the tuple $(\sigma^1_{i,t}, \textit{ZKPNeqZero}(\sigma^1_{i,t}))$ to the aggregator.

2. Add secret share

The aggregator relays any received $(\sigma^1_{i,t}, \textit{ZKPNeqZero}(\sigma^1_{i,t}))$ to k users other than u_i. We call this set of users the signing set of user u_i and we denote it by $\mathfrak{U}^i_k$. The method in which this set is constructed is left as an implementation detail. It is possible to choose a fixed subset of k users or a sliding window in which every user communicates with the next k users. We assume the latter in our evaluation of the protocol. In either case, we also assume that every user knows the identifiers of all members in its signing set and how many requests it is supposed to reply to. Honest users do not reply to further requests. Each user $u_j \in \mathfrak{U}^i_k$ first verifies *ZKPNeqZero*$(\sigma^1_{i,t})$. If the proof is invalid, u_i is reported and

the protocol aborts. Notice that if u_i is malicious, then at least one user $u_j \in \mathcal{U}_k^i$ must be honest and will, thus, detect dishonest behavior. If the proof is valid, each u_j computes:

$$\sigma_{i,t}^{2,j} = (\sigma_{i,t}^1)^{[s]_j^*} = \left((H(t)^{sk_i} g_1^{x_{i,t}})^{r_{i,t}}\right)^{[s]_j^*} \in \mathbb{G}_1. \tag{4.4}$$

Here, $[s]_j^*$ denotes the partially-reconstructed secret of user u_j computed via interpolation by multiplying $[s]_j$ with the Lagrange basis polynomial corresponding to user u_j. Finally, u_j sends $\sigma_{i,t}^{2,j}$ to the aggregator.

3. Sum secret shares

When the aggregator has received k partial signatures $\sigma_{i,t}^{2,j}$ for user u_i, it aggregates the secret shares in the exponent by computing the product of all k signatures:

$$\sigma_{i,t}^3 = \prod_{u_j \in \mathcal{U}_k^i} (\sigma_{i,t}^{2,j})^{[s]_j^*} = (\sigma_{i,t}^1)^{\sum_{u_j \in \mathcal{U}_k^i} [s]_j^*} \in \mathbb{G}_1. \tag{4.5}$$

The aggregator finally sends $\sigma_{i,t}^3$ back to the original user u_i.

4. Compute final signature

At this point, k secret shares have been added to the exponent. Thus, there is only one left to add in order to finally reconstruct s. Additionally, the random value $r_{i,t}$ must be also removed by the original user for the signature to be valid. The final signature is computed by the original user as follows:

$$\begin{aligned} \sigma_{i,t} &= \left(\sigma_{i,t}^3\right)^{\frac{1}{r_{i,t}}} \cdot (H(t)^{sk_i} g_1^{x_{i,t}})^{[s]_i^*} \\ &= (H(t)^{sk_i} g_1^{x_{i,t}})^{[s]_i^* + \sum_{u_j \in \mathcal{U}_k^i} [s]_j^*} \\ &= (H(t)^{sk_i} g_1^{x_{i,t}})^s \in \mathbb{G}_1. \end{aligned} \tag{4.6}$$

Finally, u_i submits its final signature $\sigma_{i,t}$ to the aggregator. Notice that final user signatures cannot be verified as the verification key only allows for the verification of aggregate signatures, described in the next paragraph. This is by design, as verifying individual user signatures would trivially allow an adversary to learn the private input of a user by brute force and, thus, break the confidentiality property.

4.2.3. Signature aggregation

Once every user has computed and submitted its final signature for round t, the aggregator can aggregate them together by computing their product:

$$\sigma_t = \prod_{i=1}^{n} \sigma_{i,t} = \prod_{i=1}^{n} \left(H(t)^{sk_i} g_1^{x_{i,t}}\right)^s = (H(t)^s)^{\sum sk_i} \, (g_1^s)^{\sum x_{i,t}} \in \mathbb{G}_1. \tag{4.7}$$

The aggregator then sends the aggregate signature σ_t to the verifier.

4.2.4. Verification

The aggregate signature is a signature over the sum of all inputs submitted by all registered users. The aggregator cannot create an aggregate signature for a sum that was not computed from the inputs submitted by all registered users, because doing so requires knowledge of s. Once both the aggregate signature and the sum are published by the aggregator, the verifier can check whether they both match by verifying the following equation:

$$\begin{aligned} & e\Big(H(t), vk_1\Big) e\left(g_1^{\sum x_{i,t}}, vk_2\right) \stackrel{?}{=} e\left(\sigma_t, g_2\right) \\ \Rightarrow\ & e\left(H(t), (g_2^s)^{\sum sk_i}\right) e\left(g_1^{\sum x_{i,t}}, g_2^s\right) \stackrel{?}{=} e\left(\sigma_t, g_2\right), \end{aligned} \tag{4.8}$$

where

$$
\begin{aligned}
e\left(\sigma_t, g_2\right) &= e\left(\left(H(t)^s\right)^{\sum sk_i}\left(g_1^s\right)^{\sum x_{i,t}}, g_2\right) \\
&= e\left(\left(H(t)^s\right)^{\sum sk_i}, g_2\right) e\left(\left(g_1^s\right)^{\sum x_{i,t}}, g_2\right) \\
&= e\left(H(t), \left(g_2^s\right)^{\sum sk_i}\right) e\left(g_1^{\sum x_{i,t}}, g_2^s\right) \in \mathbb{G}_T.
\end{aligned} \tag{4.9}
$$

4.2.5. Final remarks

As a final remark, we note that since the exponents of the signatures cannot be equal to 0, then it follows that none of the submitted data values can be 0 either. Thus, if 0 is a valid value that can be aggregated, additional steps must be taken. For example, we can ask each user to add 1 to their data values and then, eventually, subtract n from the sum. Finally, Figure 4.2 shows a sequence diagram of the signing and signature aggregation phases of the protocol, which succinctly summarizes the main core of the protocol.

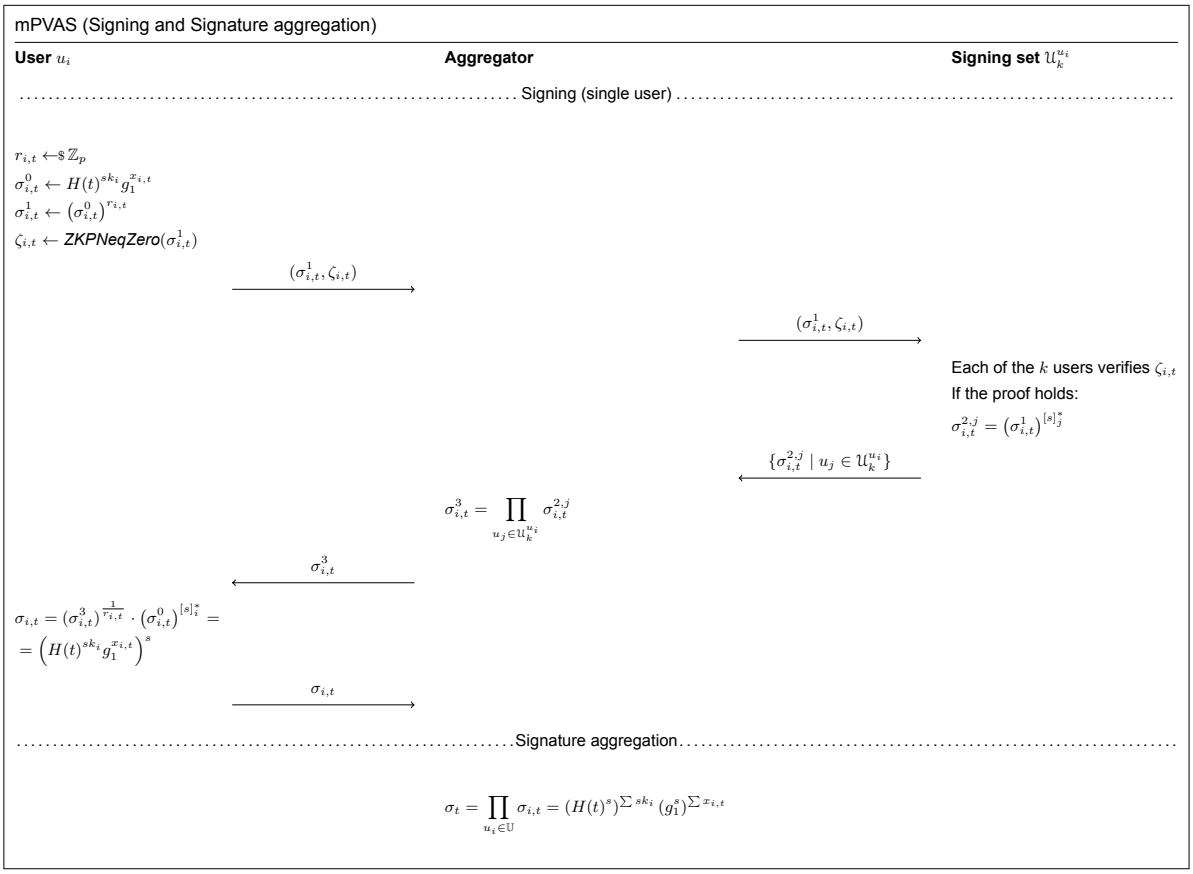


Figure 4.2: Sequence diagram of the signing and signature aggregation phases.

4.3. Privacy-preserving data aggregation protocol (PPDA)

Most of the schemes with public verifiability mentioned in Chapter 3 rely on discrete logarithm-based constructions in order to encrypt the private data. However, that means that revealing the aggregate involves computing a discrete logarithm and, as such, decrypting the ciphertexts becomes computationally expensive, although not intractable if the plaintext space is small enough to be traversed in polynomial time. Since the mPVAS scheme assumes that all users behave honestly with respect to availability, i.e. they do not send malformed data, we can speed up the extraction of the sum with an additional protocol presented here. The protocol is partially based on that of [7], with the difference that the setup is performed only once by having each user share a random seed with everyone else and we do not adopt any of its features for fault tolerance. The protocol consists of three phases: setup, sub-

mission, and aggregation. We provide a short overview of each phase before describing the protocol in more detail.

1. Setup: During the setup phase each user registers with the protocol by contacting the trusted dealer. The dealer distributes the necessary public and secret information to each user.
2. Submission: When a new aggregation round begins, each user will encrypt its private value by blinding it with zero-sum masks. These masks are random but cancel out when all ciphertexts are added together.
3. Aggregation: The aggregator collects n encrypted values from users and aggregates them by adding them together. Since the masks add up to 0, only the final sum is revealed.

4.3.1. Setup

The users signal their intention to register with the protocol to the trusted dealer. The dealer publishes the public parameters $pp_{\text{PPDA}} = (\mathbb{G}, g, H_1, p)$ generated according to a strong security parameter λ, where $\mathbb{G}$ is a cyclic group in which the Decisional Diffie-Hellman problem is hard (see Definition 3), g is a generator of $\mathbb{G}$, and $H_1 : \{0,1\}^* \to \{0,1\}^n$ is a cryptographically-secure hash function, and p is the prime order of $\mathbb{G}_1$. We also assume a cryptographic pseudo-random number generation algorithm *PRNG*, with the same output bit length as p.

Afterward, all pairs of users (u_i, u_j) engage in a Diffie-Hellman key agreement in order to compute a shared secret $DH(i,j)$. Finally, the shared seed $\boldsymbol{s}_{i,j}$ that is used by the *PRNG* is computed by letting $\boldsymbol{s}_{i,j} = H(DH(i,j))$. All messages are routed through the trusted dealer.

4.3.2. Submission

In round t, when a user u_i wants to submit its private value for aggregation, u_i creates a mask for its private value $x_{i,t}$ as follows:

$$m_{i,t} = \sum_{u_j \in U : i<j} \mathit{PRNG}(\boldsymbol{s}_{i,j}, t) - \sum_{u_j \in U : i>j} \mathit{PRNG}(\boldsymbol{s}_{j,i}, t). \tag{4.10}$$

Here, $\mathit{PRNG}(\boldsymbol{s}_{i,j}, t)$ indicates the t-th pseudorandom integer output by the PRNG seeded with $\boldsymbol{s}_{i,j}$. Then, the ciphertext is computed by adding the mask directly to the input value:

$$c_{i,t} = x_{i,t} + m_{i,t}. \tag{4.11}$$

The user then sends $c_{i,t}$ to the aggregator.

4.3.3. Aggregation

During the aggregation phase, the aggregator can combine all ciphertexts together and decrypt the final sum for round t by adding all ciphertexts together:

$$\begin{aligned} \mathit{sum}_t &= \sum_{i=1}^{n} c_{i,t} \\ &= \sum_{i=1}^{n} x_{i,t} + m_{i,t} \\ &= \sum_{i=1}^{n} x_{i,t} + \sum_{i=1}^{n} \left(\sum_{u_j \in U : i<j} \mathit{PRNG}(\boldsymbol{s}_{i,j}, t) - \sum_{u_j \in U : i>j} \mathit{PRNG}(\boldsymbol{s}_{j,i}, t) \right) \\ &= \sum_{i=1}^{n} x_{i,t} \mod p. \end{aligned} \tag{4.12}$$

Therefore, in the end, only the aggregate sum is revealed.

5

Extensions

In this chapter, we present two extensions to the main protocol described in Chapter 4. In Section 5.1, we describe an extension that reduces the communication overhead assuming non-adaptive corruptions of users. In Section 5.2, we show an extension to allow the aggregator to identify users who behave maliciously and attempt to disrupt the protocol.

5.1. Extension for lower communication overhead (mPVAS+)

In this section, we describe an extension of the mPVAS scheme that can grant a significant improvement in communication complexity. In the mPVAS scheme, assuming perfect load balancing[1], then each user needs to send its partial signature to the aggregator, which will relay it to k other users, and then check the partial signatures of k other users. This leads to a total of $k + 1$ messages exchanged, ignoring the zero-knowledge proofs and additional metadata. Although we assume that the number of malicious users in the system is much smaller than that of honest users, this level of interaction may still be too high. Fortunately, if we assume non-adaptive corruptions, i.e. the corrupted users are fixed from the start, then it is possible to decrease the number of messages exchanged using a divide-and-conquer strategy. Intuitively, users are randomly grouped into small-size groups and are provided with secret shares of s generated with different instances of secret sharing, such that the shares of two different groups are incompatible. Then, each user only needs to communicate within their own group to construct its final signature, drastically reducing the number of messages exchanged. A fully malicious group of users is required to break the unforgeability property of the scheme.

5.1.1. Setup

The key difference between the setup phase of the mPVAS+ extension and that of the main mPVAS protocol is that, after every user has signaled the intention of registering with the protocol, the dealer randomly assigns users to groups of size $c \leq k$. If $c \nmid n$, the last group will also contain the remainder $n \mod c$ of the users, and the secret sharing threshold of this group is adjusted accordingly. Next, the dealer chooses a random secret $s \leftarrow\$ \mathbb{Z}_p$ and, for each group $\mathcal{G}_i$, the dealer creates $|\mathcal{G}_i|$ shares $[s]_i$ using (c, c) Shamir Secret Sharing. We use the notation $\mathcal{G}_{u_i}$ to indicate the set of users belonging to the group of user u_i, u_i excluded. Notice that a different instance of the secret sharing protocol is used for each group. Since each instance is created using different randomness, shares that are generated for different groups are incompatible with each other. Afterward, each share $[s]_i$ is sent to the corresponding user u_i, along with a list of the other users in the same group, and the same public parameters and verification key of the mPVAS scheme.

5.1.2. Signing

The signing procedure only differs from that of the mPVAS scheme in that the partial signature $\sigma^1_{i,t}$ of each user u_i needs to be sent to the $c - 1$ other users assigned to the group of u_i, instead of k other users in the system.

[1] For example, assuming a total order on the user identifiers, when each user sends its partial signature to the k next users.

5.1.3. Signature aggregation and verification

When the signatures are finally computed, they are sent to the aggregator for aggregation and verification using the same procedures used in the mPVAS protocol.

5.1.4. Choosing the threshold *c*

The reason why this scheme has a lower communication overhead than the main scheme mPVAS is that every user only needs to communicate with $c - 1 < k$ other users in order to construct their signatures. In practice, c can be much smaller than k. Note, however, that in this case the unforgeability property of the scheme can only be guaranteed under probabilistic bounds. In fact, if c malicious users end up being assigned to the same group, then unforgeability is clearly broken since they have enough shares to collectively reconstruct s. However, the probability of this scenario happening can be made negligibly small by increasing the threshold c. As an example, let us assume $n = 1000$ users, $k = 100$ of which can be malicious. Then, choosing $c = 5$ means that each user will have to check the partial signatures of 4 other users, instead of 100 of them. The probability of obtaining a group with 5 malicious users is only $p \approx 0.002$ and it decreases exponentially as we increase the group size c: $p \approx 0.0002$ with $c = 6$ and $p \approx 0$ with $c = 7$. These probabilities were computed using a simulation with $100,000$ trials. The simulated probabilities for a wider range of scenarios are plotted in Appendix A.

5.2. Extension for input validation (mPVAS-IV)

Public verifiability allows any entity that possesses the verification key to ensure that the data aggregation was performed correctly on the inputs provided by the registered users. However, when verification fails it is useful to take steps to find what caused the failure. Since we work in an interactive setting, malicious users interested in disrupting the protocol have a rather large attack surface. In this section, we present an extension to the mPVAS protocol that allows the aggregator to identify malicious users that attempt to affect the result of the verification procedure. We assume that the aggregator is trusted with respect to identifying malicious behavior and not disrupting the protocol, but still malicious with respect to confidentiality and unforgeability. The reason why this is a fair assumption is as follows. First, note that the aggregator is very powerful with regard to the results it may output. In fact, the aggregator may decide to always output an incorrect sum that causes the verification to fail, thereby disrupting the protocol indefinitely. Additionally, the aggregator may lie about which users acted maliciously when trying to identify malicious behavior. However, we argue that if verification fails too many times, for example, more than k times, it may raise the suspicion that the aggregator may not be acting honestly since the expected number of malicious users must have been thrown out at this point. As such, in the worst case, all that a fully malicious aggregator can do is disrupt the protocol for k rounds, while the properties of confidentiality and unforgeability of the verification material are still maintained. For this reason, we argue that it is not worth it for the aggregator to act maliciously with respect to availability. Furthermore, we claim that having a mechanism that can allow the protocol to continue functioning properly after a small number of failures as opposed to an indefinitely large number of failures is worth having.

In the original mPVAS scheme, there are three ways in which a malicious user u_m might try to invalidate the verification procedure: tampering with the partial signature of another user, sending a malformed final signature, and lying about another user sending an invalid proof of inequality to zero. Additionally, a user might send a malformed PPDA ciphertext, but we avoid this by dropping the need for the PPDA scheme altogether and letting the aggregator extract the sum from the aggregate signature. We now present an extension to the mPVAS protocol, which we call mPVAS-IV, which allows the aggregator to identify malicious users in each of the three aforementioned scenarios. For each of the four phases, i.e setup, signing, aggregation, and verification, we describe the changes that must be applied to the mPVAS protocol. In Table 5.1, we summarize the updated adversarial assumptions adopted by the mPVAS-IV extension.

5.2.1. Setup

The setup phase of the mPVAS-IV protocol runs similarly to that of the main mPVAS scheme. The only difference is that the aggregator is handed additional information that can be used to detect malicious behavior. Specifically, once the signing keys sk_i and secret shares $[s]_i$ of s are generated, the dealer sends the sets $SS = \{g_2^{\frac{1}{[s]_1}}, \ldots, g_2^{\frac{1}{[s]_n}}\}$, $SK = \{h_T^{r_1} g_T^{sk_1}, \ldots, h_T^{r_n} g_T^{sk_n}\}$ to the aggregator, where $r_i \leftarrow\!\!\$\, \mathbb{Z}_p$,

h_T, g_T are random generators of $\mathbb{G}_T$. The corresponding users u_i are also given r_i, which they must know in order to generate a valid zero-knowledge proof of commitment equality, explained in more detail below.

SS contains the inverses of the secret shares of each user and allows the aggregator to ensure all final signatures are correctly signed with s. Notice that since they are stored as exponents of g_2, the aggregator cannot efficiently recover them.

SK contains commitments of the signing keys of every user. These allow the aggregator to verify whether each user signed their signature using the correct signing key and not, for example, some random number.

The public verification key includes $g_2^{\frac{1}{s}}$ and is now $vk = (g_2^{s\sum_{i=1}^{n} sk_i}, g_2^s, g_2^{\frac{1}{s}})$. Finally, dealer randomly chooses $h_1 \leftarrow\!\!\$\ \mathbb{G}_1$, and publishes the public parameters $pp = (H, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, h_1, g_2, g_T, h_T, e, k, p)$.

Entity	Confidentiality	Unforgeability	Availability
Aggregator	Malicious		Trusted
$n-k$ Users	Honest-but-curious	Trusted	
k Users	Malicious		
Verifier	Honest-but-curious	-	
Dealer	Trusted		

Table 5.1: Adversarial assumptions adopted in the mPVAS-IV scheme.

5.2.2. Signing

Tampering with the partial signature of another user

The first way in which a malicious user u_m may try to invalidate the verification procedure in the mPVAS scheme is by tampering with the partial signature $\sigma_{i,t}^1$ of some user u_i by sending, for example, a random value back instead of the value described in Equation 4.4. Users can verify whether their final signature is well-formed by ensuring that the following equation holds:

$$e(H(t), (g_2^s)^{sk_i}) e(g_1^{x_{i,t}}, g_2^s) \stackrel{?}{=} e(\sigma_{i,t}, g_2) \tag{5.1}$$

where

$$\begin{aligned} e(\sigma_{i,t}, g_2) &= e\big((H(t)^s)^{sk_i} (g_1^s)^{x_{i,t}}, g_2\big) \\ &= e\big((H(t)^s)^{sk_i}, g_2\big) e\big((g_1^s)^{x_{i,t}}, g_2\big) \\ &= e\big(H(t), (g_2^s)^{sk_i}\big) e\big(g_1^{x_{i,t}}, g_2^s\big). \end{aligned} \tag{5.2}$$

If the equality holds it means that none of the partial signatures $\sigma_{i,t}^{2,j}, u_j \in \mathcal{U}_k^i$ were tampered with, otherwise at least one of the users in the signing set $\mathcal{U}_k^i$ must have submitted an invalid $\sigma_{i,t}^{2,j}$. In this case, user u_i informs the aggregator. Recall that the aggregator possesses both the partial signature $\sigma_{i,t}^1$ of user u_i as well as all the $\sigma_{i,t}^{2,j}$, for every $u_j \in \mathcal{U}_k^i$. For each user u_j in the signing set $\mathcal{U}_k^i$, the aggregator computes:

$$\begin{aligned} \sigma_{i,t}^{2,j^*} &= e\left(\sigma_{i,t}^{2,j}, g_2^{\frac{1}{[s]_j^*}}\right) \\ &= e\left(\left((H(t)^{sk_i} g_1^{x_{i,t}})^{r_{i,t}}\right)^{[s]_j^*}, g_2^{\frac{1}{[s]_j^*}}\right) \\ &= e\left((H(t)^{sk_i} g_1^{x_{i,t}})^{r_{i,t}}, g_2\right) \\ &= e\left(\sigma_{i,t}^{1'}, g_2\right), \end{aligned} \tag{5.3}$$

where $\sigma_{i,t}^{1'}$ is expected to be equal to the original $\sigma_{i,t}^1$ sent by u_i. Note that the aggregator can find $g_2^{\frac{1}{[s]_j^*}}$, by first computing the Lagrange coefficient using the set of integer identifiers $L_i = \{i\} \cup \{j \mid u_j \in \mathcal{U}_k^i\}$:

$$\ell(j) = \prod_{h \in L_i, h \neq j} \frac{h}{h-j} \in \mathbb{Z}_p, \tag{5.4}$$

and then computing:

$$g_2^{\frac{1}{[s]_j^*}} = \left(g_2^{\frac{1}{[s]_j}}\right)^{\frac{1}{\ell(j)}} \in \mathbb{G}_2. \tag{5.5}$$

Finally, the aggregator verifies that the expected $\sigma_{i,t}^{1'}$ is, indeed, equal to the $\sigma_{i,t}^{1}$ it originally received from user u_i:

$$e\left(\sigma_{i,t}^{1}, g_2\right) \stackrel{?}{=} \sigma_{i,t}^{2,j^*}. \tag{5.6}$$

If this is the case, then u_j did not act maliciously. Otherwise, u_j will be reported as malicious. The same process is repeated for the remaining users in $\mathcal{U}_k^i$.

Lying about another user sending an invalid zero-knowledge proof of inequality to zero
In order to overcome the case where a malicious user incorrectly reports another user for having sent an invalid proof of inequality to zero, the aggregator must verify all proofs. The aggregator already possesses the proofs and the respective partial signatures of every user, so there are no additional steps required.

5.2.3. Signature aggregation

During the signature aggregation phase, the aggregator receives signatures $\sigma_{i,t}$ from every user and, then, proceeds to aggregate them. The signatures are aggregated into one signature σ_t using the procedure described in the mPVAS scheme. Thus, no changes are required for the signature aggregation phase.

5.2.4. Verification

The verification is also performed as explained in Chapter 4. However, if the verification fails, then the aggregator is expected to detect which users caused it to fail. There are two possible reasons why this might happen at this point in the protocol. One or more users sent a malformed ciphertext or malformed final signature. The first cause is avoided because the mPVAS-IV extension drops the need for a separate data aggregation protocol to be run after the aggregate signature one. However, this requires the aggregator to solve a discrete logarithm to extract the sum from the aggregate signature. As such, the plaintext space must be small enough so that the aggregator can extract the sum in polynomial time. The second cause can be detected by allowing the aggregator to check whether each final signature $\sigma_{i,t}$ is well formed.

Malformed signature detection
A malicious user u_m may try to invalidate the verification procedure by sending a malformed signature $\sigma_{m,t}$ to the aggregator. Detecting this entails verifying that the secret exponent s wraps the entire signature and that the signing key sk_m is the exponent of $H(t)$ and belongs to user u_m. Additionally, for the decryption of the sum to be tractable, the signed value must be within a confined range of values $[min, max]$ that can be traversed in polynomial time, e.g 32-bit values or smaller. In order to check for the first condition, both the aggregator and u_m can compute, using $vk_3 = g_2^{\frac{1}{s}}$:

$$\begin{aligned}
\sigma_{m,t}^{'} &= e\left(\sigma_{m,t}, g_2^{\frac{1}{s}}\right) \\
&= e\left((H(t)^{sk_m'} g_1^{x_{m,t}'})^s, g_2^{\frac{1}{s}}\right) \\
&= e\left(H(t)^{sk_m'} g_1^{x_{m,t}'}, g_2\right) \\
&= e\left(H(t), g_2\right)^{sk_m'} e\left(g_1, g_2\right)^{x_{m,t}'} \in \mathbb{G}_T.
\end{aligned} \tag{5.7}$$

Now, a non-interactive zero-knowledge proof of commitment equality is performed between the aggregator and u_m. User u_m must prove to the aggregator that the exponent of the first factor of $\sigma_{m,t}^{'}$, namely $e(H(t), g_2)^{sk_m'}$ is indeed equal to the committed value $h_T^{r_m} g_T^{sk_m} \in SK$, using the protocol shown in Figure 2.2, with inputs $S = \sigma_{m,t}^{'}$ and $T = h_T^{r_m} g_T^{sk_m}$. This ensures that the first two conditions hold. Finally, the aggregator must also ensure that the signed value is within the expected range, say in $[1, 2^{32})$. We use a bulletproof to allow users to prove the range of their signed values. However, we did

not find any implementation of bulletproofs that allows for proofs in $\mathbb{G}_T$. As such, since we cannot prove the range of $\sigma^{'}_{m,t}$, we require users to send the aggregator an additional commitment $C = h_1^r g_1^{x_{i,t}} \in \mathbb{G}_1$, along with a bulletproof to prove that $x_{i,t} \in [min, max]$ and a proof of equality (Figure 2.2) to prove that the committed value $C(r, x_{i,t})$ matches that of $\sigma^{'}_{m,t}$.

5.3. Dealing with a detected malicious user

The aggregator informs the system administrator of any detected malicious users. If the system administrator decides to kick out the dishonest users, a new trusted setup must be performed in order to re-generate the secret keys as well as the verification key. This is necessary because the verification key contains information about all the secret signing keys and, for this reason, verification fails if less than n registered users contribute to the aggregation. While the intervention of the trusted authority is not always ideal or practical, it is necessary because of the absence of a simpler recovery system in the event of faults.

6

Security analysis

In this chapter, we provide arguments in support of the security of the protocols introduced in this work. We start by providing evidence of the privacy-preserving properties of both the PPDA and mPVAS schemes, along with the extensions. We then proceed by presenting arguments showing the unforgeability property holds in all of the schemes.

6.1. Privacy proofs

We begin our privacy proof by introducing an important concept that captures the general goal of private data aggregation schemes.

6.1.1. Aggregator Obliviousness

In order to show that the private values submitted by the users stay confidential, we adopt the notion of *Aggregator Obliviousness* (AO, first formalized in [79] and later also used and extended in [54, 53]. Intuitively, a data aggregation scheme is Aggregator Oblivious if, given ciphertexts $c_{i,t}$ and signatures $\sigma_{i,t}$, the aggregator cannot learn anything about the individual plaintext inputs $x_{i,t}$, other than their sum $\sum_{i=1}^{n} x_{i,t}$. If a subset of the users $U_k \subset \mathbb{U}$ of size $|U_k| \leq k$ colludes with the aggregator by revealing their private inputs, then the aggregator can only learn $\sum_{u_i \in \mathbb{U} \setminus U_k} x_{i,t}$. We focus on the aggregator because it is the party that has access to most of the information since all messages are routed through the aggregator.

6.1.2. The PPDA scheme is Aggregator Oblivious

The PPDA scheme from Section 4.3 can be easily shown to be aggregator oblivious by proving the following theorem.

Theorem 1 *Given a ciphertext $c_{i,t}$ of a honest user u_h computed according to the PPDA scheme, a malicious aggregator that colludes with $w \leq n-2$ users cannot recover the underlying plaintexts $x_{i,t}$ of the remaining $n-w$ users.*

Proof Each ciphertext $c_{i,t} = x_{i,t} + m_{i,t}$ is blinded by a random mask $m_{i,t}$ which is computed using a cryptographically-secure PRNG as shown in Equation 4.10. The PRNG is initialized by a seed that is generated uniformly at random according to the security parameter λ. As such, each ciphertext appears virtually indistinguishable from a random value to the aggregator. Recall that each pair of users (u_i, u_j) shares a seed $s_{i,j}$, which is generated during the setup phase and distributed by the trusted dealer. It follows that there are $s = \binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$ unique, randomly-generated seeds in total, one for each pair of users. Any subset of $1 \leq w \leq n$ users collectively possesses $\sum_{i=1}^{w}(n-i)$ unique seeds. If $w \leq n-2$, then an aggregator that colludes with w users possesses $s_A \leq \sum_{i=1}^{n-2}(n-i) = \frac{(n-1)(n-2)}{2} = \frac{n^2-n-2}{2} < s$. As such, there is at least one pairwise seed that is not known by the aggregator and that is used by the remaining $n-w$ users to generate their random masks. Therefore, the ciphertexts of the remaining users will still look indistinguishable from random

to the aggregator. All the aggregator can learn is the sum of the inputs of the remaining users, but not their individual inputs. ☺

6.1.3. The mPVAS scheme is Aggregator Oblivious

In order to prove the AO property of the mPVAS scheme, we adopt a more formal approach based on a security game. In short, we show that if a probabilistic polynomial-time adversary has a non-negligible advantage of breaking the AO property of our scheme, then it also has a non-negligible advantage of breaking the AO property of the Shi et al. [79] scheme, which is proven under the Decisional Diffie-Hellman assumption. The proof of this property follows an indistinguishability-based game and it provides an adversary with access to the following oracles.

- $\mathcal{O}_{\mathsf{Setup}}(1^\lambda)$: performs the setup of the mPVAS scheme using the given security parameter λ and replies with the public parameters pp and the verification key vk. The secret values of each user $([s]_i, sk_i)$ are kept secret.
- $\mathcal{O}_{\mathsf{Compromise}^1}(u_i \in \mathbb{U})$: when queried on a user id u_i, the oracle replies with the secret of information of user u_i: $([s]_i, sk_i)$.
- $\mathcal{O}_{\mathsf{Sign}}(u_i \in \mathbb{U}, t, x_{i,t})$: given an input $x_{i,t}$ of user u_i in round t, the oracle replies with $\left(\sigma^1_{i,t}, \{\sigma^{2,j}_{i,t}\}_{u_j \in \mathcal{U}^i_k}, \sigma^3_{i,t}, \sigma_{i,t}\right)$, where each $\sigma^l_{i,t}, 1 \leq l \leq 3$ is a partial signature, and $\sigma_{i,t}$ is the final signature.
- $\mathcal{O}_{\mathsf{Challenge}}(\mathcal{X}^0_{t^*}, \mathcal{X}^1_{t^*})$: given two sets of input values $\mathcal{X}^0_{t^*}, \mathcal{X}^1_{t^*}$ of size $|\mathcal{X}^i_{t^*}| = n$, such that $\sum_{u_i \in U^*} x^0_{i,t^*} = \sum_{u_i \in U^*} x^1_{i,t^*}$, the oracle randomly flips a coin $b \leftarrow\$ \{0,1\}$ and, for set $\mathcal{X}^b_{t^*}$, it returns all the corresponding partial and final signatures of its inputs.

AO security game

The AO security game is depicted below and based on the game introduced in [79].

1. **Setup**. The adversary $\mathcal{A}$ queries the $\mathcal{O}_{\mathsf{Setup}}(1^\lambda)$ once, which returns the public parameters and verification key.
2. **Learning I.** $\mathcal{A}$ can adaptively make the following type of queries.

   Compromise. $\mathcal{A}$ specifies a uid $u_i \in \mathbb{U}$ and queries the $\mathcal{O}_{\mathsf{Compromise}^1}(u_i)$ oracle, which returns the secret information of user u_i.

   Sign. $\mathcal{A}$ can ask for the two types of partial signature, and the final signature any users $u_i \in \mathbb{U}$ by querying the oracles $\mathcal{O}_{\mathsf{Sign}}$.

   Verify. $\mathcal{A}$ is allowed to test whether a sum matches an aggregate signature by using the verification key vk obtained during the setup phase.
3. **Challenge.** Only performed once during the game. $\mathcal{A}$ chooses a set of users $U^* \subseteq \mathbb{U}$, with $|U^*| \geq 2$, that were not compromised in the learning phase and an aggregation round t^* for which no sign queries were made in the learning phase. $\mathcal{A}$ also chooses two distinct sets of inputs of users $u_i \in U^*$ for round t^*, $\mathcal{X}^0_{t^*} = \{x^0_{i,t^*}\}_{u_i \in U^*}$ and $\mathcal{X}^1_{t^*} = \{x^1_{i,t^*}\}_{u_i \in U^*}$. We require that $\sum_{u_i \in U^*} x^0_{i,t^*} = \sum_{u_i \in U^*} x^1_{i,t^*}$, otherwise $\mathcal{A}$ can trivially guess which set of inputs was chosen by the challenger. $\mathcal{A}$ sends $(U^*, t^*, \mathcal{X}^0_{t^*}, \mathcal{X}^1_{t^*})$ to $\mathcal{C}$. $\mathcal{C}$ flips a coin $b \leftarrow\$ \{0,1\}$ and returns $\left(\{\sigma^{1^b}_{i,t^*}\}_{u_i \in U^*}, \{\{\sigma^{2^b,j}_{i,t^*}\}_{u_j \in \mathcal{U}^i_k}\}_{u_i \in U^*}, \{\sigma^{3^b}_{i,t}\}_{u_i \in U^*}, \{\sigma^b_{i,t^*}\}_{u_i \in U^*}\right)$ to $\mathcal{A}$. $\mathcal{A}$ is allowed to perform more verification tests on the aggregate signatures received by $\mathcal{C}$ during the challenge phase.
4. **Learning II.** After the Challenge phase, $\mathcal{A}$ can continue to make Learning queries, as long as the aggregation round $t \neq t^*$ and the compromised users $u_i \notin U^*$.
5. **Guess.** $\mathcal{A}$ outputs a guess $b^* \in \{0,1\}$ to indicate which set of inputs was chosen by the challenger. $\mathcal{A}$ wins if $b^* = b$.

We can now define the AO property. We follow the definitions seen in [54, 53].

Definition 1 (Aggregator Oblivious) *Let* $\Pr[\mathcal{A}^{AO}]$ *denote the probability that aggregator* $\mathcal{A}$ *outputs* $b^* = b$ *in the AO game. A data aggregation protocol is said to be Aggregator Oblivious if, any polynomially bounded* $\mathcal{A}$ *has negligible advantage* $\Pr[\mathcal{A}^{AO}] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$ *of winning the AO game.*

In order to prove the security of the mPVAS scheme, we will also introduce the following hard problems, following the definitions of [49].

Definition 2 (Discrete Logarithm Problem (DLP)) *The discrete logarithm problem in a cyclic group $\mathbb{G}$ of order q and with generator g is to compute a given (g, g^a) with $a \in \mathbb{Z}_q$. We say that the DLP is hard relative to some cyclic group $\mathbb{G}$ generated with security parameter λ if, for all probabilistic polynomial-time algorithms $\mathcal{A}$, there exists a negligible function* negl *such that* $\mathsf{Pr}[\mathcal{A}(\mathbb{G}, g, g^a) \rightarrow a] \leq \mathsf{negl}(\lambda)$.

Definition 3 (Decisional Diffie-Hellman (DDH) problem) *Given $(\mathbb{G}, g, g^a, g^b, h)$, where $\mathbb{G}$ is a cyclic group of order q with generator $g \in \mathbb{G}$, $a, b \leftarrow\!\!\$\, \mathbb{Z}_q$ are uniformly sampled, and $h \in \mathbb{G}$, the DDH problem is to decide whether $h = g^{ab}$ or $h = g^c$, for a uniformly sampled $c \in \mathbb{Z}_q$. We say that the DDH problem is hard relative to some cyclic group $\mathcal{G}$ with security parameter λ if, for all probabilistic polynomial-time algorithms $\mathcal{A}$, there is a negligible function* negl *such that*

$$\left|\mathsf{Pr}\left[\mathcal{A}\left(\mathbb{G}, q, g, g^a, g^b, g^c\right) = 1\right] - \mathsf{Pr}\left[\mathcal{A}\left(\mathbb{G}, q, g, g^a, g^b, g^{ab}\right) = 1\right]\right| \leq \mathsf{negl}(\lambda), \tag{6.1}$$

where q is the order of $\mathbb{G}$, $g \in \mathbb{G}$ is a generator, $a, b, c \in \mathbb{Z}_q$ are uniformly sampled.

Definition 4 (Computational Diffie-Hellman (CDH) problem) *Given $(\mathbb{G}, g, g^a, g^b)$, where $\mathbb{G}$ is a cyclic group of order q with generator g, and $a, b \in \mathbb{Z}_q$ are uniformly sampled, the CDH problem is to compute $h = g^{ab}$. We say that the CDH is hard relative to some cyclic group $\mathbb{G}$ generated with security parameter λ if, for all probabilistic polynomial-time algorithms $\mathcal{A}$, there exists a negligible function* negl *such that* $\mathsf{Pr}[\mathcal{A}(\mathbb{G}, q, g, g^a, g^b) \rightarrow g^{ab}] \leq \mathsf{negl}(\lambda)$.

The next problem definition comes from [9].

Definition 5 (Co-Computational Diffie-Hellman (Co-CDH) problem) *Given $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, g_1, g_2, g_2^a)$ where $\mathbb{G}_1, \mathbb{G}_2$ are cyclic groups of prime order q with generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, e is a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, and $a \in \mathbb{Z}_q$ is uniformly sampled. The Co-CDH problem is to compute $g_1^a \in \mathbb{G}_1$. We say that the Co-CDH problem is hard relative to some cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ generated with security parameter λ if, for all probabilistic polynomial-time algorithms $\mathcal{A}$, there exists a negligible function* negl *such that* $\mathsf{Pr}[\mathcal{A}(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, q, e, g_1, g_2, g_2^a) \rightarrow g_1^a] \leq \mathsf{negl}(\lambda)$.

For each of the problems defined above we can define a corresponding assumption stating that there exist groups in which the problem is considered hard. The security of the mPVAS scheme relies on the following hardness assumption. We will follow the definition of [5].

Definition 6 (Symmetric External Diffie-Hellman (SXDH) assumption) *The SXDH assumption states that there exist cyclic groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, generated with security parameter λ, and an efficiently computable bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ such that DLP, CDH, Co-CDH, and DDH are all hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Given the above definitions, we can now show that the scheme guarantees the AO property by proving the following theorem.

Theorem 2 *The mPVAS scheme is Aggregator Oblivious in the random oracle model under the SXDH assumption in $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Proof Let us assume an adversary $\mathcal{A}$ that can win the AO game with a non-negligible advantage. We will show how a polynomial time algorithm $\mathcal{B}$ can break the AO scheme of [79], henceforth referred to as PPATS (Privacy-Preserving Aggregation of Time-Series data), which is provably secure under the DDH assumption, by using $\mathcal{A}$ as a subroutine. A summary of the scheme is also provided in Section 3.1. We will refer to the oracles provided by the PPATS scheme, respectively, as $\mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Setup}}, \mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Encrypt}}, \mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Compromise}}, \mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Challenge}}$. $\mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Setup}}$ returns the public parameters. $\mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Encrypt}}$ returns the ciphertext $c_{i,t}$ of a given input $x_{i,t}$ in round t using the PPATS scheme. $\mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Compromise}}$ returns the secret encryption key $sk_i^{'}$ of a specified user $u_i \in \mathbb{U}$. Finally, $\mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Challenge}}$, only called once during the game, randomly flips a coin $b \leftarrow\!\!\$\, \{0, 1\}$ and, similarly to the challenge phase described above, encrypts one of the two plaintext sets chosen by the adversary $\mathcal{X}_{t^*}^b = \{x_{i,t^*}\}_{u_i \in \mathbb{U}^*}$.

We follow the AO security game and show how $\mathcal{B}$ reacts to the queries of $\mathcal{A}$.

1. **Setup**. When $\mathcal{A}$ queries the $\mathcal{O}_{\mathsf{Setup}}(1^\lambda)$ oracle, $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Setup}}(1^\lambda)$. The latter returns the public parameters $pp_{\mathsf{PPATS}} = (H, \mathbb{G}_1, g_1, p)$. $\mathcal{B}$ also queries the $\mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Compromise}}(0)$, which returns the secret key of the aggregator $sk_A = -\sum_{i=1}^{n} sk_i^{'}$. $\mathcal{B}$ will additionally choose the remaining public parameters of the mPVAS scheme $pp = (H, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, k)$. Finally, $\mathcal{B}$ also chooses the secret keys $(s, \{sk_i\}_{u_i \in \mathbb{U}})$, creates n secret shares $[s]_i$ using $(k+1, n)-$Shamir Secret Sharing, and creates the verification key as follows:

$$vk = \left((g_2^s)^{-sk_A}, g_2^s\right) = \left((g_2^s)^{\sum_{i=1}^{n} sk_i^{'}}, g_2^s\right). \tag{6.2}$$

   Then, $\mathcal{B}$ returns the public parameters $pp = (H, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$, and the verification key vk to $\mathcal{A}$.
2. **Learning**.
   (a) **Compromise**. When $\mathcal{A}$ queries the $\mathcal{O}_{\mathsf{Compromise}^1}(u_i \in \mathbb{U})$ oracle, $\mathcal{B}$ will, in turn, query $\mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Compromise}}(u_i \in \mathbb{U})$ and return the corresponding secret key $sk_i^{'}$ of user u_i. Additionally, the secret share $[s]_i$ is also sent to $\mathcal{A}$.
   (b) **Sign**. When $\mathcal{A}$ calls $\mathcal{O}_{\mathsf{Sign}}(u_i \in \mathbb{U}, t, x_{i,t})$, $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Encrypt}}(u_i \in \mathbb{U}, t, x_{i,t})$ to obtain $c_{i,t}^{\mathsf{PPATS}} = H(t)^{sk_i^{'}} g_1^{x_{i,t}}$. $\mathcal{B}$ then computes:
   $\sigma_{i,t}^1 = (c_{i,t}^{\mathsf{PPATS}})^{r_{i,t}} = \left(H(t)^{sk_i^{'}} g_1^{x_{i,t}}\right)^{r_{i,t}}$, for $r_{i,t} \leftarrow\$\, \mathbb{Z}_p$;
   $\sigma_{i,t}^{2,j} = \left(\sigma_{i,t}^1\right)^{[s]_j^*}$, for $u_j \in \mathcal{U}_k^i$;
   $\sigma_{i,t}^3 = \prod_{u_j \in \mathcal{U}_k^i} \sigma_{i,t}^{2,j}$;
   $\sigma_{i,t} = \left(\prod_{u_j \in \mathcal{U}_k^i} \sigma_{i,t}^{2,j}\right)^{\frac{1}{r_{i,t}}} \cdot \left(\sigma_{i,t}^0\right)^{[s]_i^*} = \left(H(t)^{sk_i^{'}} g_1^{x_{i,t}}\right)^{s}$.
   Notice how each partial signature and the final signature $\sigma_{i,t}$ are constructed from the ciphertext output by the encryption algorithm of PPATS but perfectly simulate a partial or final signature of the mPVAS scheme. Finally, $\mathcal{B}$ returns $\left(\sigma_{i,t}^1, \{\sigma_{i,t}^{2,j}\}_{u_j \in \mathcal{U}_k^i}, \sigma_{i,t}\right)$ to $\mathcal{A}$.
   (c) **Verify**. $\mathcal{A}$ can test the correctness of an aggregate sum using the verification key vk obtained during the setup according to Equation 4.8.
3. **Challenge**.
   (a) $\mathcal{A}$ chooses a set of uncompromised users $U^* \subseteq \mathbb{U}$, with $|U^*| \geq 2$ and an aggregation round t^* for which no sign queries were made in the learning phase. Then, $\mathcal{A}$ also chooses two sets of ciphertexts $X_{t^*}^0 = \{x_{i,t}^0\}_{u_i \in U^*}$ and $X_{t^*}^1 = \{x_{i,t}^1\}_{u_i \in U^*}$ such that $\sum_{u_i \in U^*} x_{i,t^*}^0 = \sum_{u_i \in U^*} x_{i,t^*}^1$. When $\mathcal{A}$ calls the $\mathcal{O}_{\mathsf{Challenge}}(\mathcal{X}_{t^*}^1, \mathcal{X}_{t^*}^1)$ oracle, $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{PPATS}}_{\mathsf{Challenge}}(\mathcal{X}_{t^*}^0, \mathcal{X}_{t^*}^1)$. The oracle flips a coin $b \leftarrow\$\, \{0, 1\}$ and returns the encrypted ciphertexts of the b^{th} set $\{c_{i,t}^{\mathsf{PPATS}^b}\}_{u_i \in U^*}$. $\mathcal{B}$ computes the partial and final signatures using the $\mathcal{O}_{\mathsf{Sign}}$ oracle, returning

$$\left(\{\sigma_{i,t^*}^{1^b}\}_{u_i \in U^*}, \{\{\sigma_{i,t^*}^{2^b,j}\}_{u_j \in \mathcal{U}_k^i}\}_{u_i \in U^*}, \{\sigma_{i,t^*}^{3^b}\}_{u_i \in U^*}, \{\sigma_{i,t^*}^b\}_{u_i \in U^*}\right)$$

   to $\mathcal{A}$, In particular, the final signature is:

$$\sigma_{i,t^*}^b = \left(H(t^*)^{sk_i^{'}} g_1^{x_{i,t^*}^b}\right)^s, \text{ for } u_i \in U^*. \tag{6.3}$$

   Notice how σ_{i,t^*}^b, and all partial signatures are computed from the ciphertexts output by the encryption algorithm of the PPATS scheme and perfectly simulate the ciphertexts, partial and final signatures of the mPVAS scheme. The aggregation of all such final signatures is also valid and, thus, correctly verify using the verification key vk, as shown below:

$$\sigma_{t^*}^b = \prod_{u_i \in U^*} \sigma_{i,t^*}^b = (H(t^*)^s)^{\sum_{u_i \in U^*} sk_i^{'}} (g_1^s)^{\sum_{u_i \in U^*} x_{i,t^*}^b}. \tag{6.4}$$

If $\mathcal{A}$ has a non-negligible advantage ϵ of guessing the correct bit b^* in the AO game of the mPVAS scheme, then $\mathcal{B}$ can also win the AO game of the PPATS scheme with the same non-negligible advantage ϵ by guessing the same bit b^*. This would contradict the DDH assumption in $\mathbb{G}_1$, because the security of the PPATS scheme relies on this assumption. Additionally, if the DDH does not hold in $\mathbb{G}_1$, then the SXDH assumption does not hold either since it requires that the DDH problem be hard in $\mathbb{G}_1$. Therefore, the mPVAS scheme must be AO in the random oracle model under the SXDH assumption. ☺

6.1.4. Aggregator Obliviousness of the mPVAS+ extension

The mPVAS+ extension improves the communication overhead of the mPVAS scheme by grouping users randomly into small-size groups in which distinct secret-sharing schemes are employed with a lower threshold than the one used in mPVAS.

Theorem 3 *The mPVAS+ extension does not affect the AO property of the PPDA and mPVAS schemes.*

Proof The mPVAS+ extension does not affect the execution of the PPDA scheme which, thus, remains Aggregator Oblivious as proven in Theorem 1.

The mPVAS+ extension does, however, change the behavior of the base mPVAS scheme. Users are grouped into groups of size $c \leq k$ and the secret exponent s is shared among members of each group using (c, c) secret sharing. It is, thus, possible for c malicious users to end up in the same group. When that happens, they can collectively reconstruct s and share it with the aggregator, thus allowing it to tamper with the signatures of honest users. However, note that even with knowledge of s, the aggregator still cannot learn the private values of individual users because they are also blinded by either some randomness $r_{i,t}$, randomly sampled by the users, or by the secret key sk_i. When s is known by the aggregator, the mPVAS scheme directly reduces to the PPATS scheme of [79], which is Aggregator Oblivious. ☺

6.1.5. Aggregator Obliviousness of the mPVAS-IV extension

We now show why the mPVAS-IV extension is still Aggregator Oblivious despite the additional knowledge provided to the aggregator.

Theorem 4 *The mPVAS-IV extension maintains the Aggregator Oblivious property.*

Proof The additional information received by the aggregator does not yield any advantage to breaking the AO property. In fact, the secret shares the aggregator receives in the set SS cannot be efficiently extracted due to the DLP problem being hard in $\mathbb{G}_2$. The commitments contained in the set SK are hiding, thus the aggregator cannot extract the signing keys either.

Furthermore, as with the mPVAS scheme, all partial signatures $\sigma_{i,t}^1, \sigma_{i,t}^{2,j}, \sigma_{i,t}^3$ are all blinded with a random exponent $r_{i,t}$ chosen by the user, which makes extracting $x_{i,t}$ from these partial signatures computationally hard.

The value shown in Equation 5.7, namely $e\left(H(t), g_2\right)^{sk_m'} e\left(g_1, g_2\right)^{x_{m,t}'}$, is hiding under the random oracle model, hence the private value $x_{m,t}^{'}$ cannot be extracted, either.

Finally, all the zero-knowledge proofs used in the extension, namely ZKPNeqEq, ZKPEq, and the bulletproof do not leak any information about the private witness because of their zero-knowledge property. As such, the aggregator cannot learn the private value of honest users and, thus, the mPVAS-IV scheme remains Aggregator Oblivious. ☺

6.2. Unforgeability proofs

In this section, we provide evidence of the unforgeability of the aggregate signature schemes presented in this work. In order to do so, we adopt the concept of *Aggregate Unforgeability* (AU), which was first formalized in [53, 54, 27] and denotes the notion that in round t, the aggregator cannot produce a valid proof of correctness σ_t for a sum that was not computed from inputs submitted by the registered users.

6.2.1. Types of forgeries

We say that an adversary $\mathcal{A}$ successfully forges an aggregate signature σ_t for some round t if it outputs the tuple (sum_t, σ_t) such that $\text{Verify}(t, vk, \text{sum}_t, \sigma_t) = 1$ and $\text{sum}_t \neq \sum_{i=1}^{n} x_{i,t}$. In other words, $\mathcal{A}$ can provide a valid aggregate signature that successfully authenticates an incorrect sum. In line with the works of [5, 54, 53, 27, 84], we distinguish between two types of forgeries:

1. **Type-I**, when an adversary $\mathcal{A}$ forges an aggregate signature for a round t^* in which $\mathcal{A}$ did not see any signatures from the users. This implies forgeries for future rounds of the protocol.
2. **Type-II**, when an adversary $\mathcal{A}$ forges an aggregate signature for a round t^* in which $\mathcal{A}$ saw all signatures from the users. This implies forgeries for present or past rounds of the protocol.

6.2.2. Aggregate Unforgeability of the mPVAS scheme

In this section, we show that it is computationally infeasible for the aggregator to produce either Type-I or Type-II forgeries in the mPVAS scheme.

Theorem 5 *The mPVAS scheme is Aggregate Unforgeable against Type-I forgeries.*

Proof A Type-I forgery occurs when the aggregator manages to output a valid aggregate signature σ_t in a round t in which it did not receive any signatures from the users. Thus, the aggregator can only use knowledge from previous rounds or knowledge obtained by colluding with users. First, we note how signatures from different rounds are incompatible with each other. Since we assume a cryptographically-secure hash function $H : \{0,1\}^* \to \mathbb{G}_1$, it means that, under the random oracle model, the output of such a hash function can be considered random. As such, in each round t, each signature has a different random factor $H(t)$. Even assuming the aggregator chooses the round identifier t, because of the collision resistance property of H, it has a negligible probability of finding two different round identifiers t, t' such that $H(t) = H(t')$. Similarly, because of the second pre-image resistance property of H, given t, the aggregator has negligible probability of finding another t' such that $H(t) = H(t')$. It follows that the aggregator cannot reuse signatures from previous rounds. The only other option left for the aggregator is to construct new signatures itself. However, in order to do so, all secret signing keys sk_i are required but, assuming colluding users, the aggregator has only access to at most k of them. The aggregator also needs the secret exponent s to compute a valid signature, but it has only access to at most k secret shares of s, which are not enough to reconstruct s. ☺

Theorem 6 *The mPVAS scheme is Aggregate Unforgeable against Type-II forgeries.*

Proof There are three pieces of information that can allow the aggregator to successfully forge an aggregate signature in a round in which it received all signatures from the users: the secret exponent s, the factor g_1^s, or, when $k > 1$, the factor $H(t)^s$.

The exponent s is secret-shared by all users using $(k+1, n)$-Shamir Secret Sharing. Since we assume at most k malicious users who collude with each other and k shares leak no information about the underlying secret s, then no dishonest party can directly learn s. Additionally, if $(H(t)g_1)^s \in \mathbb{G}_1$ is known, recovering s is considered computationally infeasible because DLP is assumed to be hard in $\mathbb{G}_1$. The same argument applies to the value g_2^s, which is part of the verification key and known by everyone. Note that, while the aggregator knows g_2^s, since the Co-CDH problem is assumed to be hard in $\mathbb{G}_1$ and $\mathbb{G}_2$, obtaining g_1^s is still considered hard. In a malicious setting where users can behave arbitrarily, sending malformed signatures may allow them to gain additional information that will allow them to break the AU property. Individual signatures cannot be tampered with without the knowledge of s, or g_1^s. Additionally, when $k > 1$, $H(t)^s$ can also be used to learn g_1^s by letting two users collaborate with each other. By sending a partial signature of the form $\sigma_{i,t}^1 = (H(t)^a g_1^b)^c$, where either $a = 0$ or $b = 0$, a user could learn, respectively, $H(t)^s$ or g_1^s. $H(t)$ can be used to derive g_1^s from the signature of another malicious user. With g_1^s, a malicious aggregator can tamper with the aggregate signature and publish an authenticated aggregate of its choosing. However, even when malicious users deviate from the protocol, they cannot learn these two values. When a user u_i submits a partial signature $\sigma_{i,t}^1 = (H(t)^{sk_i} g_1^{x_{i,t}})^{r_{i,t}}$, u_i is required to prove in zero-knowledge that $sk_i \cdot r_{i,t} \neq 0$ and $x_{i,t} \cdot r_{i,t} \neq 0$. If u_i is honest, then the k proofs will be correctly verified because the signature will be well-formed. If u_i is malicious, then at least one of the k verifiers, say u_j, must be honest. This means that u_j will abort

the protocol and report u_i if ZKPNeqZero($\sigma^1_{i,t}$) $= \perp$, meaning $\sigma^1_{i,t}$ is malformed. Hence, u_i can only learn $(H(t)g_1)^s$, which is not enough to create forgeries.

□

6.2.3. Aggregate Unforgeability of the mPVAS+ scheme

The mPVAS+ scheme can be thought of as multiple instances of the regular mPVAS scheme running on multiple groups of users but with different instances of Shamir Secret Sharing used to generate the secret shares of s. Thus, as long as each group contains at least one honest user, the AU property still holds following the same logic presented in the previous section for the mPVAS scheme.

However, we restate that the mPVAS+ only provides AU with k malicious users only if we assume non-adaptive corruptions. If this is not the case, the security of the scheme is downgraded to that of an mPVAS instance with $k = c - 1$.

6.2.4. Aggregate Unforgeability of the mPVAS-IV extension

Despite the aggregator being trusted to detect users who attempt to disrupt the protocol and being trusted not to disrupt the execution of the protocol, the aggregator is still considered malicious with respect to the unforgeability property of the mPVAS-IV extension. This extension provides the aggregator with additional knowledge that is not available in the main scheme. As such, in this section, we provide additional arguments to show why the AU property is still maintained in the mPVAS-IV extension.

Theorem 7 *The mPVAS-IV scheme is Aggregate Unforgeable against Type-I forgeries.*

Proof The first new piece of information that the aggregator has in the mPVAS-IV extension is the set $SS = \{g_2^{\frac{1}{[s]_1}}, \ldots, g_2^{\frac{1}{[s]_n}}\}$. Since DLP is assumed to be intractable in $\mathbb{G}_2$, the aggregator has negligible probability of obtaining the secret share $[s]_i$ of a user u_i from $g_2^{\frac{1}{[s]_i}}$. The aggregator is also handed $g_2^{\frac{1}{s}}$ and, using the same argument, it is easy to see that recovering s from it is also hard. Finally, the aggregator possesses $SK = \{h_T^{r_1} g_T^{sk_1}, \ldots, h_T^{r_n} g_T^{sk_n}\}$, but, because of the perfect hiding property of Pedersen commitments, the aggregator cannot learn any information about the signing key sk_i from its corresponding commitment. Hence, the additional information that is handed to the aggregator in the mPVAS-IV extension gives the aggregator no advantage of learning the necessary information to create Type-I forgeries.

□

Theorem 8 *The mPVAS-IV scheme is Aggregate Unforgeable against Type-II forgeries.*

Proof There is no additional piece of information handed to the aggregator in the mPVAS-IV extension that could allow it to create Type-II forgeries. Intuitively, this is because all of the additional values are members of either $\mathbb{G}_2$ or $\mathbb{G}_T$, but the signatures are elements of $\mathbb{G}_1$. As such, there is no additional information that could be used by the aggregator to tamper with the signatures in $\mathbb{G}_1$, assuming the SXDH assumption holds in the chosen pairing group. □

6.3. Completeness, soundness, and zero-knowledge of the Sigma protocols

In this section, we prove completeness, soundness, and honest-verifier zero-knowledge for each of the Sigma protocols shown in Figure 2.2 and Figure 2.3.

6.3.1. Proofs for the ZKP of equality between commitments (ZKPEq)

Completeness This property directly follows from inspection of the protocol in Figure 2.2. The first equation that the verifier has to check holds because:

$$g_1^{s_1} h_1^{s_2} = g_1^{r_1 + xc} h_1^{r_2 + yc} = g_1^{xc} g_1^{r_1} h_1^{yx} h_1^{r_2} = S^c t_1. \tag{6.5}$$

Similarly, the second equation also holds:

$$g_2^{s_1} h_2^{s_3} = g_2^{r_1 + xc} h_2^{r_3 + zc} = g_2^{xc} g_2^{r_1} h_2^{zx} h_2^{r_3} = T^c t_2. \tag{6.6}$$

(Special) Soundness This property is proved by showing that given two accepting conversations between a prover and a verifier, obtained with different challenges, it is possible to extract a witness that satisfies the statement to be proven. For the ZKPEq protocol, given two accepting conversations $(t_1, t_2, c, s_1, s_2, s_3)$ and $(t_1, t_2, c', s_1', s_2', s_3')$, with $c \neq c'$. Then,

$$\begin{aligned} & g_1^{s_1} h_1^{s_2} = S^c t_1,\ g_2^{s_1} h_2^{s_3} = T^c t_2,\ g_1^{s_1'} h_1^{s_2'} = S^{c'} t_1,\ g_2^{s_1'} h_2^{s_3'} = T^{c'} t_2 \\ \Rightarrow & g_1^{s_1 - s_1'} h_1^{s_2 - s_2'} = S^{c-c'} \text{ and } g_2^{s_1 - s_1'} h_2^{s_3 - s_3'} = T^{c-c'} \\ \Rightarrow & S = g_1^{\frac{s_1 - s_1'}{c-c'}} h_1^{\frac{s_2 - s_2'}{c-c'}} \text{ and } T = g_2^{\frac{s_1 - s_1'}{c-c'}} h_2^{\frac{s_3 - s_3'}{c-c'}}. \end{aligned} \tag{6.7}$$

We can extract the witness (x, y, z) by letting $x = \frac{s_1 - s_1'}{c-c'}$, $y = \frac{s_2 - s_2'}{c-c'}$, and $z = \frac{s_3 - s_3'}{c-c'}$.

(Special) Honest-verifier zero-knowledge Given an arbitrary challenge $c \in \mathbb{Z}_p$, the real-world conversation can be simulated by choosing random $r_1', r_2', r_3' \leftarrow\$ \mathbb{Z}_p^3$, and setting $t_1 = g_1^{r_1'} h_1^{r_2'} S^{-c}$ and $t_2 = g_2^{s_1} h_2^{s_3} T^{-c}$. The distribution of the simulated messages is the same as that of those sent in the real protocol. As such, an honest verifier cannot learn anything more from engaging in the protocol than it could already learn on its own.

6.3.2. Proofs for the ZKP of inequality to zero (ZKPNeqZero)

Completeness From the protocol in Figure 2.3, since $x \neq 0$, then x must have an inverse. Thus, we can rewrite S as:

$$\begin{aligned} & S = h^x g^y \\ \Rightarrow & S^{\frac{1}{x}} = h g^{\frac{y}{x}} \\ \Rightarrow & h = S^{\frac{1}{x}} g^{-\frac{y}{x}}. \end{aligned} \tag{6.8}$$

Similarly, since $y \neq 0$, we can write $g = S^{\frac{1}{y}} h^{-\frac{x}{y}}$. Then, we have:

$$S^{l_1} g^{r_1} = S^{u + \frac{c}{x}} g^{v - \frac{cy}{x}} = S^u g^v (S^{\frac{1}{x}} g^{-\frac{y}{x}})^c = a h^c, \tag{6.9}$$

and

$$S^{l_2} h^{r_2} = S^{w + \frac{c}{y}} h^{z - \frac{cx}{y}} = S^w h^z (S^{\frac{1}{y}} h^{-\frac{x}{y}})^c = b g^c. \tag{6.10}$$

(Special) Soundness Given accepting conversations $(a, b, c, r_1, l_1, r_2, l_2)$ and $(a, b, c', r_1', l_1', r_2', l_2')$, with $c \neq c'$, we have:

$$\begin{aligned} & S^{l_1} g^{r_1} = a h^c, S^{l_1'} g^{r_1'} = a h^{c'} \\ \Rightarrow & S^{l_1 - l_1'} g^{r_1 - r_1'} = h^{c-c'} \\ \Rightarrow & S = h^{\frac{c-c'}{l_1 - l_1'}} g^{-\frac{r_1 - r_1'}{l_1 - l_1'}}. \end{aligned} \tag{6.11}$$

Using the same method, we also get $S = h^{\frac{c-c'}{l_2 - l_2'}} g^{-\frac{r_2 - r_2'}{l_2 - l_2'}}$. Assuming $l_1 \neq l_1'$ and $l_2 \neq l_2'$, otherwise we would have divisions by zero, the witness (x, y) can be obtained with:

$$\begin{aligned} x &= \frac{c - c'}{l_1 - l_1'} \\ y &= -\frac{r_2 - r_2'}{l_2 - l_2'}. \end{aligned} \tag{6.12}$$

(Special) Honest-verifier zero-knowledge Given an arbitrary challenge $c \in \mathbb{Z}_p$, the real-world conversation can be simulated by choosing random $l_1', r_1', l_2', r_2' \leftarrow\$ \mathbb{Z}_p^4$, and setting $a \leftarrow S^{l_1'} g^{r_1'} h^{-c}$ and $b \leftarrow S^{l_2'} h^{r_2'} g^{-c}$. The distribution of the simulated messages is the same as that of those sent in the real protocol. As such, an honest verifier cannot learn anything more from engaging in the protocol than it could already learn on its own.

7

Performance evaluation

In this chapter, we analyze the theoretical computation and communication complexity of our protocols. We then continue to evaluate the practical running time of each protocol for different sets of parameters.

7.1. Computation complexity

In this section, we analyze the theoretical computation complexity for each of our protocols. Each protocol can be logically split into several operations that act as a unit. For each such operation, we calculate the total number of multiplications, multiplicative inverses, exponentiations, and hashes required to perform it. Since we work with bilinear maps, we also divide the algebraic operations based on the group in which they are executed because, in practice, they may perform differently in each group. The computation complexity of the zero-knowledge proofs and secret share reconstruction protocols are analyzed separately and then treated as a unit for simplicity.

7.1.1. Zero-knowledge proofs and secret sharing

We start the analysis of our protocols by first summarizing the complexities of the zero-knowledge proofs used in our schemes, which are then treated as a unit throughout the rest of this chapter.

In Table 7.1, we summarize the number of algebraic operations required to generate a zero-knowledge proof of inequality to zero, as well as to verify one. These are derived by direct inspection of the protocol in Figure 2.3. We denote this unit of computation by ZKPNeqZero.

Table 7.1: Summary of the computation complexity of the zero-knowledge proof of inequality to zero (ZKPNeqZero).

Operation	Mul.	Inv.	Exp.	Hash
Generation	$2:\mathbb{G}_1, 2:\mathbb{Z}_p$	$4:\mathbb{Z}_p$	$4:\mathbb{G}_1$	$1:\mathbb{Z}_p$
Verification	$4:\mathbb{G}_1$	0	$6:\mathbb{G}_1$	$1:\mathbb{Z}_p$

In Table 7.2, we summarize the number of operations required by the zero-knowledge proof of commitment equality presented in Figure 2.2. We implicitly use the Fiat-Shamir heuristic to make the proof non-interactive, hence the presence of the hashes in the analysis. This unit of computation is denoted by ZKPEq.

Table 7.2: Summary of the computation complexity of the zero-knowledge proof of equality between commitments(ZKPEq). Note, $\mathbb{G}'_1$ and $\mathbb{G}'_2$ do not necessarily correspond to the pairing groups $\mathbb{G}_1$ and $\mathbb{G}_2$. They just indicate two possibly distinct groups of the same order. In our protocol, we perform one proof with $\mathbb{G}'_1 = \mathbb{G}'_2 = \mathbb{G}_T$, another with $\mathbb{G}'_1 = \mathbb{G}'_2 = \mathbb{G}_1$, and one with $\mathbb{G}'_1 = \mathbb{G}_1, \mathbb{G}'_2 = \mathbb{G}_T$.

Operation	Mul.	Exp.	Hash
Generation	$1:\mathbb{G}'_1, 1:\mathbb{G}'_2, 3:\mathbb{Z}_p$	$2:\mathbb{G}'_1, 2:\mathbb{G}'_2$	$1:\mathbb{Z}_p$
Verification	$2:\mathbb{G}'_1, 2:\mathbb{G}'_2$	$3:\mathbb{G}'_1, 3:\mathbb{G}'_2$	$1:\mathbb{Z}_p$

In Table 7.3, instead, we summarize the number of operations that every user must perform in order to compute a partially-reconstructed share of s. We denote this unit of computation as SSS.

Table 7.3: Summary of the number of operations required to reconstruct a share. It requires $k-1$ multiplications and k inversions to compute the Lagrange coefficient of a signing set. Finally, the reconstructed share can be computed by multiplying the coefficient with the secret share.

Operation	Mul.	Inv.
Share construction	$k : \mathbb{Z}_p$	$k : \mathbb{Z}_p$

The computation complexity of bulletproofs is much more complex, thus we refer the reader to the original paper for more details [10].

7.1.2. PPDA

During the setup phase, every pair of users exchanges random seeds using the Diffie-Hellman key agreement protocol. For each user, this costs n exponentiations in $\mathbb{G}$ and $n-1$ hashes.

During an aggregation round, the PPDA scheme only requires that each user generate $n-1$ random numbers using a seeded PRNG and add these together with the private datum they want to hide, for a total of n additions in $\mathbb{Z}_p$. The aggregator, on the other hand, needs to add $n-1$ ciphertexts together in order to find the sum.

7.1.3. mPVAS and mPVAS+

We now proceed to analyze the theoretical computation complexity for the users, the aggregator, and the verifier.

Dealer During the setup phase, which is only performed once, the dealer has to draw one random number $s \leftarrow\$ \mathbb{Z}_p$ and perform k exponentiations in $\mathbb{G}_2$ to generate the verification key vk. Finally, the dealer has to generate n secret shares of s using (t, n)-Shamir secret sharing; more details can be found here [78]. In the case of the mPVAS+ extension, the dealer still needs to generate n secret shares of s, but using a different instance of (c, c) secret sharing for each group.

Users In the setup phase, each user only needs to draw a random integer and compute one exponentiation, which is a negligible number of operations compared to the rest of the protocol.

During the signing phase, users interact with each other in order to construct valid signatures. We assume perfect load balancing since we can allow users to stop replying to new signing requests after the expected number of received requests has already been reached. With perfect load balancing, each user sends its partial signature $\sigma^1_{i,t}$ to the next k users, along with a zero-knowledge proof of inequality to zero of the exponents. These users first check any received ZKPNeqZero proof to ensure the partial signatures are well formed, then they add their secret share in the exponent, creating k partial signatures $\sigma^{2,j}_{i,t}$, which are sent to the aggregator. The aggregator then aggregates these and sends the partially aggregated signature $\sigma^3_{i,t}$ back to the original user u_i. Finally, u_i removes the randomness $r_{i,t}$ from the partial signature and adds its own secret share to finally construct its final signature. All of these operations are performed either in $\mathbb{G}_1$ or in $\mathbb{Z}_p$, and a summary of the exact number and type of operations is shown in Table 7.4. We prioritize multiplication operations, which are cheaper, and we try to reduce the number of exponentiations and pairing operations as much as possible.

Table 7.4: Summary of the computation complexity of signing operations for a single user in a single round of the mPVAS scheme.

Operation	Mul.	Inv.	Exp.	Hash	Gen. ZKPNeqZero	Ver. ZKPNeqZero	SSS
Create partial sig.	$1 : \mathbb{G}_1, 2 : \mathbb{Z}_p$	0	$2 : \mathbb{G}_1$	1	1	0	0
Add secret share	0	0	$k : \mathbb{G}_1$	0	0	k	k
Compute final sig.	$1 : \mathbb{G}_1$	$1 : \mathbb{Z}_p$	$2 : \mathbb{G}_1$	0	0	0	1

Signing in the mPVAS+ scheme requires a smaller number of messages to be exchanged between the users. Users are assigned to groups of size $c \leq k$ and, as such, each user communicates with $c-1$

other users in its group. In practice, c can be made much smaller than k. This leads to a decrease in the number of exponentiations and verifications of ZKPNeqZero, as well as the number of shares computed.

Table 7.5: Summary of the computation complexity of signing operations for a single user in the mPVAS+ scheme.

Operation	Mul.	Inv.	Exp.	Hash	Gen. ZKPNeqZero	Ver. ZKPNeqZero	SSS
Create partial sig.	$1:\mathbb{G}_1, 2:\mathbb{Z}_p$	0	$2:\mathbb{G}_1$	1	1	0	0
Add secret share	0	0	$c-1:\mathbb{G}_1$	0	0	$c-1$	$c-1$
Compute final sig.	$1:\mathbb{G}_1$	$1:\mathbb{Z}_p$	$2:\mathbb{G}_1$	0	0	0	1

Aggregator The aggregator is involved in two steps of the computation of each signature. First, for each user u_i, the aggregator sums k secret shares added by users in u_i's signing set by multiplying the partial signatures $\sigma_{i,t}^{2,j}$ together and exploiting their additive homomorphic property. Finally, the aggregator also aggregates all signatures together to get the final aggregate signature over all submitted values. In the mPVAS+ scheme, the aggregator needs to aggregate fewer partial signatures for each user because of the smaller group size. All of these operations consist of multiplications in $\mathbb{G}_1$ and are summarized in Table 7.6.

Table 7.6: Summary of the computation complexity for the aggregator in the mPVAS and mPVAS+ schemes.

(a) mPVAS

Operation	Mul.
Sum secret shares	$n(k-1):\mathbb{G}_1$
Aggregate signatures	$n-1:\mathbb{G}_1$

(b) mPVAS+

Operation	Mul.
Sum secret shares	$n(c-2):\mathbb{G}_1$
Aggregate signatures	$n-1:\mathbb{G}_1$

Verifier The computation overhead for the verifier is very low. In fact, verification of the result of the aggregation can be done in constant time, regardless of the number of users in the system. The exact computation overhead is shown in Table 7.7. Verification in the mPVAS+ scheme does not differ from the main mPVAS scheme and has the same complexity.

Table 7.7: Summary of the computation complexity for the verifier in the mPVAS and mPVAS+ schemes.

Operation	Mul.	Exp.	Pair.	Hash
Verification	$1:\mathbb{G}_T$	$1:\mathbb{G}_1$	3	1

7.1.4. mPVAS-IV

We analyze the worst-case scenario, when the partial signature of every user is tampered with and when there is at least one malicious user submitting an invalid signature. We analyze the additional computation complexity of the input validation operations for users and the aggregator.

Dealer The dealer has to compute $2n$ additional exponentiations in $\mathbb{G}_T$, n multiplications in $\mathbb{G}_2$, $n-1$ additional multiplications in $\mathbb{G}_T$, and n inverse in $\mathbb{Z}_p$ in order to generate the sets SS and SK. The verification key also requires an additional exponentiation in $\mathbb{G}_2$ and an inverse in $\mathbb{Z}_p$ to be computed.

User The additional computation overhead for each user is the same as that of the verifier, as shown in Table 7.7, since verifying the validity of a partial signature amounts to checking Equation 4.8, but with different values. Moreover, each user generates 2 ZKPEq proofs and 1 Bulletproof to prove their final signature is well formed.

Aggregator The aggregator, on the other hand, has to verify the partial signatures $\sigma_{i,t}^{2,j}$ for every user $u_j \in \mathcal{U}_k^i$, and repeat the process for every user in the system. This leads to an overall computation complexity of $\mathcal{O}(kn)$. The exact number and types of operations are shown in Table 7.8. The aggregator must also decrypt the sum embedded in the signature by performing the verification procedure on all possible messages until the correct one is found. This involves $\mathcal{O}(2^l)$ verifications, where l is the bit

length of the encrypted messages. It follows that l must be small enough to allow decryption to be run in polynomial time. This is analyzed in Section 7.4.

Table 7.8: Summary of the (additional) computation complexity for the aggregator and the users in the mPVAS-IV scheme.

(a) Aggregator

Operation	Mul.	Inv.	Exp.	Pair.	Ver. ZKPEq	Ver. Bull.	SSS
Verifying all $\sigma_{i,t}^{2,j}$	0	$kn : \mathbb{Z}_p$	$kn : \mathbb{G}_2$	$kn+1$	0	0	kn
Verifying all $\sigma_{i,t}$	0	0	0	n	$2n$	n	0

(b) Users

Operation	Mul.	Exp.	Pair.	Hash	Gen. ZKPEq	Gen. Bull.
Generating proofs	$1 : \mathbb{G}_T$	$1 : \mathbb{G}_1$	3	1	2	1

7.1.5. Summary

In this section, we summarize the asymptotic computation complexity of our schemes.

Table 7.9 shows the asymptotic computation complexity of the PPDA scheme. The scheme is very efficient since it requires $\mathcal{O}(n)$ additions for both the aggregator and the users, which are cheap operations. The users are also required to compute $\mathcal{O}(n)$ exponentiations and hashes to generate the shared seeds, but these operations are only performed once during the setup.

Table 7.9: Asymptotic computation complexity of the PPDA scheme.

Participant	Add.	Exp.	Hash
Dealer	-	-	-
User	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Aggregator	$\mathcal{O}(n)$	-	-
Verifier	-	-	-

In Table 7.10, we show the complexity for the mPVAS scheme. The user performs $\mathcal{O}(k)$ exponentiations, share reconstructions, and proof verifications, while aggregator performs $\mathcal{O}(kn)$ multiplications, which are less expensive than exponentiations and ZKP operations. As such, we expect the practical running time to be slower for the users during an average aggregation round because they perform more complex operations.

Table 7.10: Asymptotic computation complexity of the mPVAS scheme. For the mPVAS+ scheme, replace k with c.

Participant	Mul.	Inv.	Exp.	Pair.	Hash	Gen. ZKPNeqZero	Ver. ZKPNeqZero	SSS
Dealer	$\mathcal{O}(n)$	-	$\mathcal{O}(1)$	-	-	-	-	-
User	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(k)$	-	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(k)$	$\mathcal{O}(k)$
Aggregator	$\mathcal{O}(kn)$	-	-	-	-	-	-	-
Verifier	$\mathcal{O}(1)$	-	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-	-	-

In Table 7.11, instead, we show the asymptotic computation complexity for each participant in the mPVAS-IV scheme. Intuitively, the performance of the aggregator is greatly reduced due to the increased number of expensive operations, which are in the order of $\mathcal{O}(kn)$. For the users, the complexity remains the same.

Table 7.11: Asymptotic computation complexity of the mPVAS-IV scheme.

(a)

Participant	Mul.	Inv.	Exp.	Pair.	Hash	Gen. ZKPNeqZero	Ver. ZKPNeqZero
Dealer	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	-	-	-	-
User	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(k)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(k)$
Aggregator	$\mathcal{O}(kn)$	$\mathcal{O}(kn)$	$\mathcal{O}(kn)$	$\mathcal{O}(n)$	-	-	$\mathcal{O}(n)$
Verifier	$\mathcal{O}(1)$	-	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-	-

(b)

Participant	Gen. ZKPEq	Ver. ZKPEq	Gen. Bull.	Ver. Bull.	SSS
Dealer	-	-	-	-	-
User	$\mathcal{O}(1)$	-	$\mathcal{O}(1)$	-	$\mathcal{O}(k)$
Aggregator	-	$\mathcal{O}(n)$	-	$\mathcal{O}(n)$	$\mathcal{O}(kn)$
Verifier	-	-	-	-	-

7.2. Communication complexity

In this section, we analyze the communication complexity of our protocols. We denote by $\|\mathbb{G}\|$ the bit length of an element in $\mathbb{G}$, which is dependent on the chosen elliptic curve for the pairing operations and on its security parameter. In our experimental analysis, we used curves whose element sizes are summarized in Table 7.13. In Table 7.12, we summarize the asymptotic communication complexities of all schemes.

7.2.1. PPDA

During the setup phase, each user sends their Diffie-Hellman public key to the dealer, which relays it to the other users. This results in $n(n-1)$ messages sent by the dealer and 1 sent by the users. During an aggregation round, each user sends one message, namely their ciphertext, to the aggregator. The aggregator does not send any messages in this protocol, besides publishing the resulting sum to the verifiers.

7.2.2. mPVAS and mPVAS+

The size of a zero-knowledge proof of inequality to zero message requires approximately $2\|\mathbb{G}_1\|+4\|\mathbb{Z}_p\|$ bits and we denote this amount by $\|$ ZKPNeqZero $\|$.

Dealer During the setup phase, the dealer sends n secret shares of s, one for each user. These shares are elements of $\mathbb{Z}_p$ and, thus, the approximate size of the messages sent during the setup phase is $n\|\mathbb{Z}_p\|$.

Users In the mPVAS and mPVAS+ schemes, all signing operations happen in $\mathbb{G}_1$. In the mPVAS scheme, each user sends 1 partial signature $\sigma_{i,t}^1$ to its signing set, along with a zero-knowledge proof of inequality to zero. Additionally, each user also replies to k signing requests from other users. Finally, users send their final signature to the aggregator. As a result, in every round, each user sends out approximately $(2+k)\|\mathbb{G}_1\| + \|$ ZKPNeqZero $\|$ bits, without accounting for implementation-specific metadata. In the mPVAS+ scheme, users reply to $c-1$ signing requests, instead, leading to $(1+c)\|\mathbb{G}_1\| + \|$ ZKPNeqZero $\|$ bits exchanged.

Aggregator In the mPVAS and mPVAS+ schemes, the aggregator relays messages from the users. For each of the n users, the aggregator relays k partial signatures $\sigma_{i,t}^1$ to their respective signing sets, together with k zero-knowledge proofs of inequality to zero. Finally, the aggregator also returns a partial signature $\sigma_{i,t}^3$ to each user, for a total of $(kn+n)\|\mathbb{G}_1\| + kn\|$ ZKPNeqZero $\|$ bits sent out by the aggregator in the mPVAS scheme. Similarly, in the mPVAS+ scheme, the total bits the aggregator has to send out in a round approximately amounts to $(cn)\|\mathbb{G}_1\| + (c-1)n\|$ ZKPNeqZero $\|$, where c is the group size.

7.2.3. mPVAS-IV

We denote by $\| \text{ZKPEq} \| = \|\mathbb{G}_1^{'}\| + \|\mathbb{G}_2^{'}\| + 2\|\mathbb{Z}_p\|$ the approximate bit length of a zero-knowledge proof of commitment equality and by $\| \text{BP} \|$ the bit length of a 32-bit bulletproof. More specific details on the sizes of bulletproofs can be found in the original paper [10]. In the following section, we only analyze the communication complexity of the mPVAS-IV scheme in the case in which a malformed final signature is sent to the aggregator, resulting in the failure of the verification procedure. The other malicious behavior does not require additional messages to be exchanged by the participants.

Dealer In addition to the $n\|\mathbb{Z}_p\|$ bits sent to send the secret shares to the respective users, the dealer also sends additional information to the aggregator. The set SS consists of approximately $n\|\mathbb{G}_2\|$ bits, while the set SK consists of $n\|\mathbb{G}_T\|$ bits. Each user u_i also receives a integer $\|\mathbb{Z}_p\|$ from the dealer.

Users When a malicious user sends a malformed final signature, the verification of the aggregate signature will fail. In order to detect which user sent out invalid data, the aggregator requires additional information from each user. Specifically, each user u_i needs to prove that their signature was signed using the correct signing sk_i, which can be done with a zero-knowledge proof of commitment equality. Additionally, the aggregator wants to make sure the aggregate signature is decryptable, i.e. the sum can be efficiently extracted from it. Thus, each user must prove the signed value is in a valid range, such as $[1, 2^{32})$. In order to do so, however, bulletproofs internally require an additional commitment on the private value, which must be equal to that of the signature. Thus, a second proof of equality between commitments must be produced by the users. Finally, each user must also generate and send out a bulletproof to prove their private value is in the correct range. The total communication complexity for a single user results in $(2 + k)\|\mathbb{G}_1\| + \| \text{ZKPNeqZero} \| + 2\| \text{ZKPEq} \| + \| \text{BP} \|$.

Aggregator The aggregator does not have to send additional messages in the mPVAS-IV extension. As such, the communication complexity remains the same as that of the mPVAS scheme.

7.2.4. Summary

In Table 7.12, we summarize the asymptotic communication complexity of all schemes. The dealer has to share information with every user, which leads to a complexity of $\mathcal{O}(n)$ for all signature schemes. Additionally, in the PPDA scheme, it needs to broadcast each user's public key to every other user, for a total of $\mathcal{O}(n^2)$ messages. Users only communicate within their own signing set, for a complexity of $\mathcal{O}(k)$, or $\mathcal{O}(c)$ in the mPVAS+ scheme. The aggregator needs to relay messages between each user and their signing set, which leads to a complexity of $\mathcal{O}(kn)$ for the mPVAS and mPVAS-IV schemes, and $\mathcal{O}(cn)$ for the mPVAS+ scheme. Verifiers do not actively participate in the protocol.

Table 7.12: Asymptotic communication complexities of all protocols.

Participant	PPDA	mPVAS	mPVAS+	mPVAS-IV
Dealer	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Aggregator	-	$\mathcal{O}(kn)$	$\mathcal{O}(cn)$	$\mathcal{O}(kn)$
User	$\mathcal{O}(1)$	$\mathcal{O}(k)$	$\mathcal{O}(c)$	$\mathcal{O}(k)$
Verifier	-	-	-	-

7.3. Comparison with related schemes

We conclude the section with a quick comparison with other related schemes. We first note that similar schemes such as [5, 54, 53, 63], work in a different system and adversarial model where there is little to no interaction between the participants except for the initial setup. As such, the communication complexities for these schemes is $\mathcal{O}(1)$ for both the aggregator and the users. The computation complexity is $\mathcal{O}(1)$ for the users and $\mathcal{O}(n)$ for the aggregator. While this is better than any of our schemes, their adversarial model is also weaker than the one adopted by us. As discussed in Chapter 3, these schemes either assume honest behavior from the users, like [54, 5], or no collusions between the aggregator and the users [63], or they rely on a semi-trusted party [63, 53]. Nonetheless, it is easy to notice that the mPVAS scheme can also generalize all of these scenarios. The case in which all users behave

honestly can be simulated by choosing $k = 0$, thus leading to a non-interactive scheme with $\mathcal{O}(1)$ communication complexity and a computation complexity nearly identical to that of the PUDA scheme [54]. Furthermore, the mPVAS scheme can also generalize the scheme presented in [53], which entrusts a semi-trusted third party with the secret signing key, by choosing $k = 1$. This scenario also leads to constant communication complexity.

7.4. Experimental running time

In this section, we analyze the experimental running times of our protocols. We use the Charm framework [1, 16] to develop a proof-of-concept implementation of our schemes. The framework is widely used for the prototyping and benchmarking of cryptographic schemes, e.g. [3, 74]. For the mPVAS and mPVAS+ schemes, the experiments are performed over the MNT224 elliptic curve. The MNT224 curve is pairing friendly, it allows for type-3 pairings, which are necessary for the SXDH assumption to hold, and provides 112 bits of security [19, 86]. This is the most secure curve provided by the Charm framework that is compatible with our schemes. While the current recommendation is to use curves that provide 128 bits of security as a conservative choice, 112 bits is the minimum security level required by NIST for the US Federal Government [6]. For the mPVAS-IV extension, the experiments are performed on the BN254 curve, instead. The reason for this choice is that it is the only pairing-friendly curve provided by the Charm framework for which we were able to find an experimental implementation of bulletproofs [39]. Although at first glance, the BN254 curve appears to be more secure than the MNT224 due to the larger prime field (254 bits as opposed to 224 bits), a recently published attack [50] on elliptic curves dramatically reduced the security of several pairing-friendly curves, including BN254, whose security level is downgraded from 128 to 100 bits [75]. For this reason, we stress this curve should not be used in real-world implementations of the protocol. We only use it here to evaluate our protocol, since the performance of this curve should be comparable to that of other curves offering 128 bits of security that do not suffer from the aforementioned attack.

The experiments were run on a 2018 MacBook Pro with a 2.20 GHz 6-core Intel Core i7 CPU with 16GB of RAM. The results show the total running time for the aggregator and for a single user, averaged over 10 runs. The protocol was executed sequentially on a single core with no special optimizations. Messages between users are assumed to be delivered instantaneously.

Table 7.13 shows the size of an element in each of the three types of groups, as well as $\mathbb{Z}_p$, according to [47] and also experimentally confirmed by us. The size of the elements is a parameter that influences the performance of the various algebraic operations performed in each group.

Table 7.13: Sizes of the elements in each group on the MNT224 and BN254 curves.

Curve	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	$\mathbb{Z}_p$
MNT224	56 bytes	168 bytes	168 bytes	28 bytes
BN254	64 bytes	128 bytes	384 bytes	32 bytes

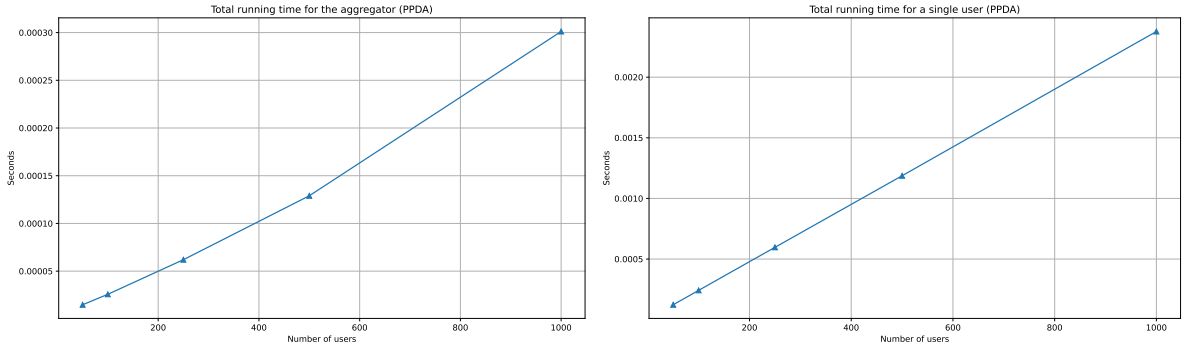
7.4.1. PPDA running time

In this section, we briefly analyze the performance of the PPDA scheme. As can be seen in Figure 7.1, the scheme performs very efficiently for both the users and the aggregator. This is expected since the scheme is very simple and only involves generating $n - 1$ random numbers for each user, as well as adding these numbers together. The aggregator only needs to add n ciphertexts up. Additions are very cheap compared to the more expensive multiplications and pairings required in the other schemes. In both cases, the running time increases linearly in the number of users participating in the protocol.

7.4.2. mPVAS running time

Here, we analyze the practical performance of the mPVAS scheme. Figure 7.2a shows the running time for the aggregator. As stated before, all computations were performed on a single core of the test machine. The results show that even in the worst tested case when the system has 1000 users and the threshold k is set to withstand 30% of malicious users, the total running time is slightly over 1 second. Additionally, the graph shows that when $k = 0$ and the aggregator does not have to perform the *sum secret shares* step for every user, the running time is close to zero.

For the users, the running time is slightly worse. While the asymptotic complexity for the users is

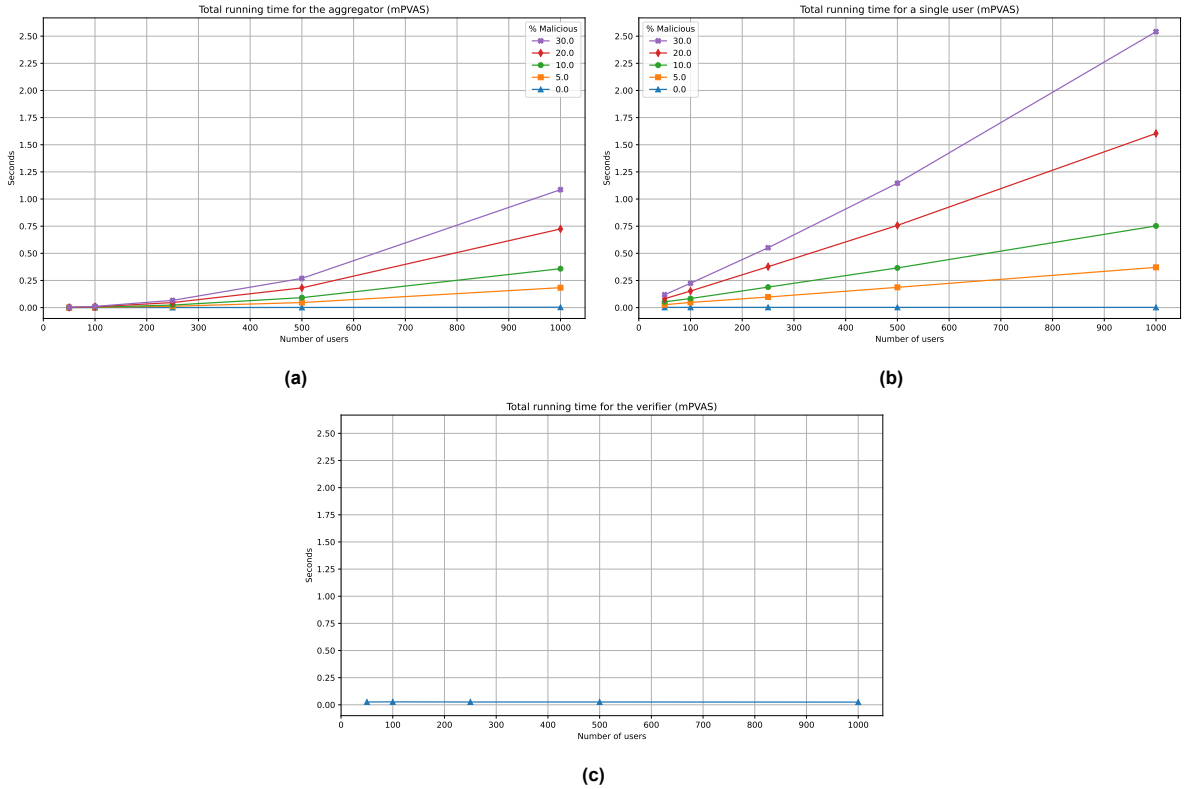


(a) (b)

Figure 7.1: Experimental running time for the PPDA scheme.

$\mathcal{O}(k)$, verifying k zero-knowledge proofs of inequality to zero represents a clear bottleneck for the users in light of the hidden constants of the ZKP. Nonetheless, as can be seen from Figure 7.2b, the running time is only around 2.50 seconds for a single user even in the extreme case when $k = 300$, i.e. 30% of 1000 users are malicious, and it gets progressively lower as k decreases.

Finally, Figure 7.2c shows the running time for the verifier. As expected, since the verifier only needs to compute three pairings, regardless of the number of users, the running time is essentially constant.



(a) (b)

(c)

Figure 7.2: Experimental running time for the mPVAS scheme for the aggregator, a single user, and a verifier.

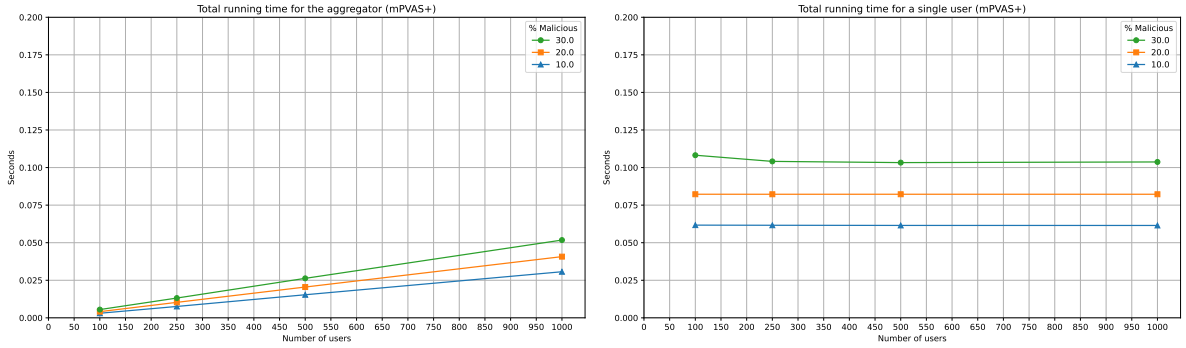
7.4.3. mPVAS+ running time

Unlike the main mPVAS scheme, the mPVAS+ scheme ensures confidentiality and unforgeability only against non-adaptive corruptions under probabilistic bounds. However, this translates to much faster running times for both the users and the aggregator. Recall that the scheme works by randomly grouping users and using a different instance of Shamir Secret Sharing for each group. Users only need

to communicate with other users belonging to the same group. As such, we only need to choose the group size c to be small enough so that the probability of c malicious users ending up in the same group is negligible. In practice, this means the group size c can often be much smaller than k. In our experiments, we only used $10\%, 20\%, 30\%$ as the percentages of malicious users, as with 5% the advantage offered by the mPVAS+ extension is negligible for some scenarios. We ran a simulation over $100,000$ trials to select the appropriate group size c for each of the three parameters and found that, when 10% of users are malicious, a group size of $c = 7$ is sufficient to ensure the probability of grouping c malicious users together is negligible. Similarly, $c = 10$ and $c = 13$ were also found to be ideal when the percentages of malicious users are 20% and 30%, respectively. The results of the simulations we used to guide our decisions can be found in Appendix A.

Figure 7.3a shows the results of our experiments for the aggregator. As explained in the previous section, the main bottleneck for the aggregator is the *sum secret shares* step, while the running time required to aggregate the final signatures is usually negligible. However, since the mPVAS+ extension sets fixed group sizes, it follows that even with 1000 users, 30% of which malicious, the aggregator only needs to aggregate $13 - 1 = 12$ partial signatures $\sigma_{i,t}^{2,j}$ for each user, instead of 300. This explains why the running time is so low if compared to the main mPVAS scheme.

The same logic can be applied to the users. The main bottleneck for users is verifying the zero-knowledge proofs of inequality to zero. However, the number of such proofs is also bounded by the group size. Indeed, each user only needs to verify $c - 1$ proofs. So, in the worst tested case, even with 300 malicious users, each user only needs to verify $13 - 1 = 12$ proofs instead of 300. In Figure 7.3b we can see the actual results of our experiments. The running time for each user is almost constant for most practical scenarios, albeit we stress that the mPVAS+ extension is only secure against non-adaptive corruptions.



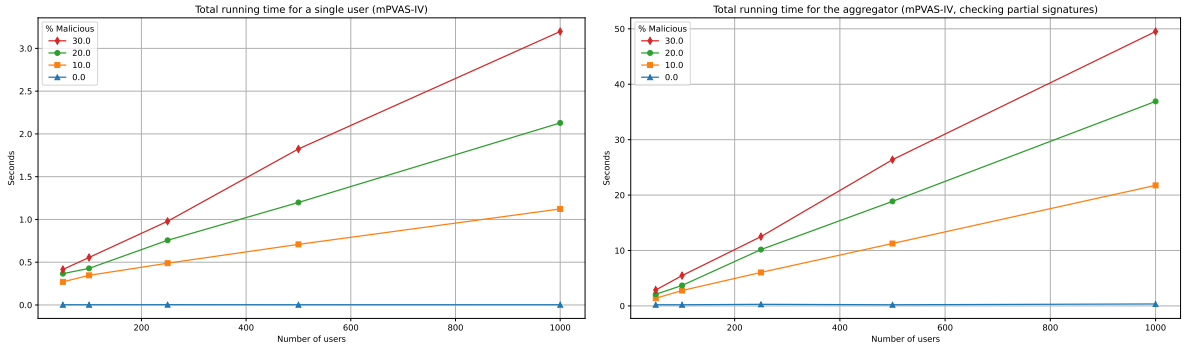
(a) (b)

Figure 7.3: Experimental running time of the mPVAS+ scheme for the aggregator and a single user.

7.4.4. mPVAS-IV running time

The mPVAS-IV scheme deals with cases in which malicious users either submit malformed values to be aggregated or tamper with those of other users. As such, the aggregator is required to perform additional checks on the submitted signatures in order to ensure they are well-formed and, if not, identify which users submitted incorrect values. Figure 7.4 shows the running times for a single user and the aggregator for the case in which a malicious user sends back an incorrect signature $\sigma_{i,t}^{2}$. In Figure 7.4a, we see the total running time for a single user, which is slightly higher than in the regular mPVAS scheme because of the additional checks on the partial signatures and the generation of the required zero-knowledge proofs. For the aggregator, the running time is much higher than in the mPVAS extension. The reason for this increase is due to the additional exponentiations and pairings operations required to check the validity of a partial signature, which must be repeated for every user in a signing set. As such, the running time will increase proportionally with increasingly larger signing set sizes. We remark that our implementation does not include any specific optimizations, such as parallelization, which can in practice drastically reduce the running time for the aggregator.

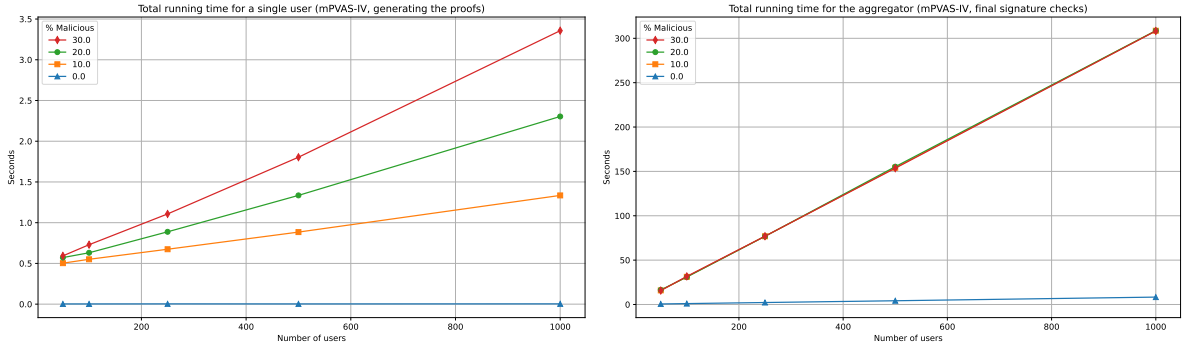
Figure 7.5, instead, shows the case in which a malicious user submits a malformed final signature to the aggregator. When this happens, the verification of the aggregate signature will fail. The aggregator can,



(a) (b)

Figure 7.4: Running time of the mPVAS-IV scheme for the aggregator and a single user. We evaluate the case in which a malicious user tampers with the partial signature $\sigma_{i,t}^1$ of another user.

then, detect which user submitted an invalid signature by performing several checks using bulletproofs for range validation, zero-knowledge proofs of commitment equality to ensure the correct signing keys were used, and the properties of bilinear maps to verify that the correct secret exponent s was used. These checks must be performed on all n users and, of course, they linearly add up to the total running for the aggregator. In fact, in Figure 7.5b, we can see that the running time for the aggregator is even higher than in the case with one malformed partial signature and can reach up to 300 seconds on our test machine. Generating the bulletproof requires roughly 100ms for each individual user while verifying a single bulletproof takes 8ms for the aggregator. Despite the much larger running times for the aggregator, we remark that the mPVAS-IV extension deals with very extreme cases that in a normal execution of the protocol we would expect to encounter only a limited number of times before all malicious users are kicked out and the protocol can continue to operate more smoothly.



(a) (b)

Figure 7.5: Running time of the mPVAS-IV scheme for the aggregator and a single user. We evaluate the case in which a malicious user submits a malformed final signature $\sigma_{i,t}$ that causes the verification to fail.

7.5. Concluding remarks

From the evaluation presented in this chapter, we believe that the performance of the schemes presented in this work is practical for real-world applications. For the users, the practical running time does not go beyond the 3.5-seconds mark even in the mPVAS-IV extension with 300 adaptive malicious users. With non-adaptive malicious users, the running time for each user is firmly below 0.25 seconds. The real-world performance of the protocol is heavily dependent on the parameter k. The lower k is, the faster the protocol performs, and vice versa. The performance is even better for the aggregator running the mPVAS and mPVAS+ schemes, while it degrades quickly with the mPVAS-IV scheme. In the latter case, we remark that the performance degradation is to be expected only in a handful of aggregation rounds because, once all malicious users have been kicked out, the remaining rounds of aggregation

will perform more efficiently once again.

While we performed our experiments on a relatively powerful machine from 2018, we also note that our implementation is a proof-of-concept lacking many optimizations. Additionally, despite most of the expensive cryptographic operations being executed by compiled C code, the rest of our implementation was written in Python, which is a slow, interpreted language. An implementation fully written in faster languages such as C++ or Rust, would have yielded even faster running times.

We note that even in smart grids, it is often possible to develop low-budget smart meters that can perform complex cryptographic operations [61]. This, coupled with the fact that meters only report their measurements every few minutes and that manufacturers can act as trusted authorities, renders our schemes suitable for smart grid deployments, although a low choice of k is recommended. Applications like federated learning, medical data sharing, and others where dedicated servers can be used to participate in data aggregation protocols are also suitable for our scheme.

As always, in security, there is often a trade-off between security and performance that must be evaluated on a case-by-case basis. Fortunately, our protocols are parametrized and allow system administrators to choose parameters that can strike their preferred balance between security and performance.

8

Conclusion

In this conclusive chapter we provide a brief discussion of this research project, delineate possible future lines of work, and provide some concluding thoughts.

8.1. Discussion

In this work, we presented a private data aggregation protocol that guarantees confidentiality, integrity, and authenticity of the aggregate even in the presence of a malicious aggregator and a bounded number k of malicious users. In other words, once users submit a private value for aggregation, nobody can either learn what this value is nor can they change it. The scheme is publicly-verifiable in constant-time by anyone holding the public verification key. The scheme does not rely on other semi-trusted party during any aggregation round, but only on a trusted authority during the initial setup. The protocol comprises two schemes that are executed sequentially. The first is an aggregate signature scheme in which each user starts with a commitment-like partial signature that is "re-signed" by k other users and the originating user itself using shares of a secret key s, which is generated during setup. When $k+1$ such partial signatures are combined, each user obtains a new signature that is signed with s. As long as no more than k malicious users are present in the system, no malicious user nor the aggregator can directly learn s or any information that might allow them to tamper with the signatures of honest users, thus guaranteeing the integrity and authenticity of the result. Each private value is hidden inside a Pedersen commitment-like signature and only the final aggregate signature, obtained by combining n user signatures, can actually be verified so as not to break the confidentiality property. The second is a simple privacy-preserving data aggregation scheme based on zero-sum masks that are freshly generated during each aggregation round using a seeded pseudo-random number generator. The purpose of this scheme is to allow for faster decryption of the aggregate as well as for a large plaintext space. The scheme uses random masks generated using the outputs from a secure PRNG and, as long as at least two users behave honestly, no one can learn the private value of an individual honest user.

We also provided an extension to our scheme that dramatically improves the communication complexity of our scheme under probabilistic bounds at the expense of a weaker security model with non-adaptive user corruptions. In other words, as long as the users that behave maliciously are fixed from the beginning, then we can achieve the same properties of the original scheme but with fewer messages exchanged by the users. The extension works by randomly assigning each user to a group of size $c \leq k$ and by splitting the common secret key s using different secret sharing instances for each group. Thus, malicious users from different groups cannot collude with each other. However, the signature scheme is not unforgeable anymore if c malicious users end up in the same group, but we can make the probability of this happening arbitrarily small by choosing a larger c.

Finally, we also provided an extension to identify users that behave arbitrarily maliciously. These include users that send invalid messages that cause the aggregation and verification to fail. This is a desirable feature because the schemes from the related literature assume that users will send well-formed messages, although this may not always be true. Schemes with public verifiability allow a verifier to detect when the statistic is computed incorrectly, but cannot pinpoint the cause. It is beneficial to

take steps to identify the cause thereof and to take action in order to avoid more failures in the future.

All of the schemes only rely on an aggregator and a set of users and do not require an additional trusted or semi-trusted party during any aggregation round. A trusted party is only required once during the setup phase.

We evaluated the security of our protocols and showed that they achieve the security properties set forth in our research question, namely confidentiality, integrity, and authenticity. Additionally, we evaluated the performance of our mPVAS protocol and found that its performance is acceptable for several real-world applications since an aggregation round can be executed in a matter of seconds, even without any optimization. The mPVAS-IV extension is significantly slower for the aggregator, although it is meant to deal with extreme cases of malicious behavior, which are not to be expected in every aggregation round and its performance can be greatly improved using parallelization. In general, the performance of our protocols is entirely dependent on the chosen threshold k: the higher k is, the worse the protocol performs, and vice versa. This means that a system administrator can choose k according to the desired levels of security and performance.

Despite the many desirable properties, our protocols have also several downsides. The only summarization function directly supported by our protocols is the sum, although other functions can also be derived from it, such as the average. Other useful functions, such as finding the minimum or maximum value, or the standard deviation are also desirable statistics. Additionally, the protocols require interaction from the users, which must be online during every aggregation round, and cannot withstand any user dropout. As such, if a system is prone to users dropping out or being unable to be constantly online, then our protocols would not be a suitable choice.

8.2. Future work

While the schemes introduced in this work achieve many desirable features, they also have several constraints which make them unsuited for some applications. In this section, we provide a summary of what we consider to be the most important drawbacks of our scheme that should be addressed in the future.

Fault tolerance None of the schemes presented in this work are fault tolerant. This means that the schemes cannot withstand any user suddenly dropping out of the system. Indeed, all users are required to participate in the protocol in order to compute the aggregate and the corresponding verification material. Fault tolerance is a desirable feature in many scenarios, such as participatory sensing, where the network connection of one or more mobile devices could suddenly drop during an aggregation round.

Better support for multi-dimensional data Data is often multi-dimensional in nature and in many scenarios it would be beneficial to support multi-dimensional data by design. Our protocol can only aggregate data across one dimension at a time and, as such, when aggregating multiple dimensions the performance of the protocol will decrease linearly in the number of dimensions. There are already private data aggregation schemes that allow for the efficient packing of multi-dimensional data into one single value to keep the complexity low like [58].

Removing the need for a trusted setup The mPVAS and mPVAS+ extensions require a trusted setup, where a trusted authority needs to intervene in order to help with the key generation and distribution. We note that this requirement is shared by many other schemes that, similarly, deal with malicious participants [5, 53, 54]. While in principle this is only required once, it is not always practical for all real-world applications. For smart grid deployments or for coordinated data sharing in healthcare, one could expect that one or more reputable institutions could be trusted enough to perform this task. However, in general, this is not always possible. As such, it is desirable to have a scheme that can also deal with situations where a trusted authority is not available.

Additionally, we note that the mPVAS-IV extension requires a new trusted setup whenever a malicious user is identified because the verification keys need to be updated to account for the removed users. Thus, it would be convenient to have a procedure to rebootstrap the system without requiring the intervention of a trusted authority. A fault-tolerant scheme could, in principle, help with this problem

since the kicked-out users could be considered dropouts and, thus, ignored during aggregation and verification.

Range validation One malicious behavior that our schemes cannot fully protect against is data poisoning, which occurs when one or more malicious users decide to submit incorrect data to either render the aggregation useless or to skew it towards a specific value [48]. Note that this type of attack can happen even if the aggregator is honest. Since we work in a privacy-preserving setting where the individual values cannot be directly inspected, the general strategy to mitigate these types of attacks is to ensure that the submitted data is within a valid range. The specific range depends on the context. For example, if we want to compute the average age of a set of patients, then it would be unrealistic to expect someone to be 200 years old, thus we can limit the range to something like $[0, 100]$. Extending our schemes with range validation would ensure that not only the aggregator nor any malicious user cannot tamper with the values submitted by honest users but also that the values submitted by malicious users are within a valid range, increasing the trust in the correctness of the aggregation. We conclude by noting that the mPVAS-IV extension already uses bulletproofs to ensure that the range of the submitted values is within a range that can allow the aggregator to decrypt the aggregate efficiently. Thus, if we can trust the aggregator to also perform range validation honestly, then the scheme already offers this feature. However, in this work, we do not make this assumption. As a result, a different party or the users themselves must collaborate in order to ensure the correct range of all submitted values.

8.3. Concluding remarks

”Data is the new oil” that is fueling the Fourth Industrial Revolution. However, just like oil, unregulated and uncontrolled use can lead to severe issues for individuals and society as a whole, especially when privacy is at stake. Privacy-preserving data aggregation is a privacy-enhancing technology aimed at protecting sensitive data during processing, but it is not a definitive solution. Many private data aggregation schemes work in the honest-but-curious model and do not offer any protection against internal parties that behave maliciously not only with respect to the confidentiality of the data but also its integrity. Protocols that attempt to solve this problem only partially do so by defending against either malicious aggregators or malicious users, but fail when both the aggregator and users are malicious and may also collude with each other. The data aggregation protocol presented in this work aims at solving this problem by not only ensuring the confidentiality of the input values but also the integrity and authenticity of the aggregate even in the presence of a malicious aggregator and a subset of malicious users that collaborate to tamper with the result of the aggregation. Ensuring not only confidentiality, but also integrity and authenticity even in the presence of malicious adversaries helps to develop more trust in the results of privacy-preserving schemes and make such schemes appealing to a wider range of scenarios.

References

[1] Joseph A. Akinyele et al. “Charm: A Framework for Rapidly Prototyping Cryptosystems”. In: *Journal of Cryptographic Engineering* 3.2 (June 2013), pp. 111–128. ISSN: 2190-8508, 2190-8516. DOI: 10.1007/s13389-013-0057-3. URL: http://link.springer.com/10.1007/s13389-013-0057-3 (visited on 06/29/2022).

[2] Istemi Ekin Akkus et al. “Non-Tracking Web Analytics”. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security - CCS ’12*. The 2012 ACM Conference. Raleigh, North Carolina, USA: ACM Press, 2012, p. 687. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382268. URL: http://dl.acm.org/citation.cfm?doid=2382196.2382268 (visited on 08/29/2022).

[3] Anees Ara et al. “A Secure Privacy-Preserving Data Aggregation Scheme Based on Bilinear ElGamal Cryptosystem for Remote Health Monitoring Systems”. In: *IEEE Access* 5 (2017), pp. 12601–12617. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2716439. URL: http://ieeexplore.ieee.org/document/7953637/ (visited on 07/14/2022).

[4] Bernard Baffour, Thomas King, and Paolo Valente. “The Modern Census: Evolution, Examples and Evaluation”. In: *International Statistical Review* 81.3 (2013), pp. 407–425. ISSN: 1751-5823. DOI: 10.1111/insr.12036. URL: http://onlinelibrary.wiley.com/doi/abs/10.1111/insr.12036 (visited on 08/29/2022).

[5] Bence Gabor Bakondi et al. “Publicly Verifiable Private Aggregation of Time-Series Data”. In: *2015 10th International Conference on Availability, Reliability and Security*. 2015 10th International Conference on Availability, Reliability and Security (ARES). Toulouse, France: IEEE, Aug. 2015, pp. 50–59. ISBN: 978-1-4673-6590-1. DOI: 10.1109/ARES.2015.82. URL: http://ieeexplore.ieee.org/document/7299898/ (visited on 10/29/2021).

[6] Elaine Barker and Allen Roginsky. *Transitioning the Use of Cryptographic Algorithms and Key Lengths*. National Institute of Standards and Technology, 2018.

[7] Keith Bonawitz et al. “Practical Secure Aggregation for Privacy-Preserving Machine Learning”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’17: 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas Texas USA: ACM, Oct. 30, 2017, pp. 1175–1191. ISBN: 978-1-4503-4946-8. DOI: 10.1145/3133956.3133982. URL: https://dl.acm.org/doi/10.1145/3133956.3133982 (visited on 05/16/2022).

[8] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. “Evaluating 2-DNF Formulas on Ciphertexts”. In: *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*. Ed. by Joe Kilian. Vol. 3378. Lecture Notes in Computer Science. Springer, 2005, pp. 325–341. DOI: 10.1007/978-3-540-30576-7_18. URL: https://doi.org/10.1007/978-3-540-30576-7%5C_18 (visited on 09/30/2022).

[9] Dan Boneh, Ben Lynn, and Hovav Shacham. “Short Signatures from the Weil Pairing”. In: *J. Cryptol.* 17.4 (2004), pp. 297–319. DOI: 10.1007/s00145-004-0314-9.

[10] Benedikt Bunz et al. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018 IEEE Symposium on Security and Privacy (SP). San Francisco, CA: IEEE, May 2018, pp. 315–334. ISBN: 978-1-5386-4353-2. DOI: 10.1109/SP.2018.00020. URL: https://ieeexplore.ieee.org/document/8418611/ (visited on 05/25/2022).

[11] US Census Bureau. *Meeting Information and Agenda: September 14-15, 2017*. Census.gov. 2017. URL: https://www.census.gov/about/cac/sac/meetings/2017-09-meeting.html (visited on 08/30/2022).

[12] US Census Bureau. *Why We Conduct the Decennial Census of Population and Housing*. Census.gov. URL: https://www.census.gov/programs-surveys/decennial-census/about/why.html (visited on 08/29/2022).

[13] Jan Camenisch and Markus Stadler. “Proof Systems for General Statements about Discrete Logarithms”. In: *Technical Report/ETH Zurich, Department of Computer Science* 260 (1997).

[14] *Census Privacy*. EPIC - Electronic Privacy Information Center. URL: https://epic.org/issues/democracy-free-speech/census-privacy/ (visited on 08/30/2022).

[15] Rosalie Chan. *The Cambridge Analytica whistleblower explains how the firm used Facebook data to sway elections*. Business Insider Nederland. Oct. 5, 2019. URL: https://www.businessinsider.nl/cambridge-analytica-whistleblower-christopher-wylie-facebook-data-2019-10/ (visited on 10/16/2022).

[16] *Charm-Crypto Docs! — Charm-Crypto 0.50 Documentation*. URL: https://jhuisi.github.io/charm/ (visited on 07/14/2022).

[17] Marco Ciotti et al. “The COVID-19 Pandemic”. In: *Critical Reviews in Clinical Laboratory Sciences* 57.6 (Aug. 17, 2020), pp. 365–388. ISSN: 1040-8363. DOI: 10.1080/10408363.2020.1783198. pmid: 32645276. URL: https://doi.org/10.1080/10408363.2020.1783198 (visited on 08/29/2022).

[18] *CNN.Com - Major Power Outage Hits New York, Other Large Cities - Aug. 14, 2003*. URL: https://edition.cnn.com/2003/US/08/14/power.outage/ (visited on 10/10/2022).

[19] Hui Cui et al. “Efficient and Expressive Keyword Search Over Encrypted Data in Cloud”. In: *IEEE Transactions on Dependable and Secure Computing* 15.3 (May 2018), pp. 409–422. ISSN: 1941-0018. DOI: 10.1109/TDSC.2016.2599883.

[20] Ivan Damgård. “On Σ-protocols”. In: *Lecture Notes, University of Aarhus, Department for Computer Science* (2002), p. 84. URL: https://www.cs.au.dk/~ivan/Sigma.pdf.

[21] Helen Davidson. “Around 20% of Global Population under Coronavirus Lockdown”. In: *The Guardian. World news* (Mar. 24, 2020). ISSN: 0261-3077. URL: https://www.theguardian.com/world/2020/mar/24/nearly-20-of-global-population-under-coronavirus-lockdown (visited on 10/06/2022).

[22] Allan DeBlasio. *Effects of Catastrophic Events on Transportation System Management and Operations: (438442008-001)*. American Psychological Association, 2003. DOI: 10.1037/e438442008-001. URL: http://doi.apa.org/get-pe-doi.cfm?doi=10.1037/e438442008-001 (visited on 10/06/2022).

[23] Sanket S. Dhruva et al. “Aggregating Multiple Real-World Data Sources Using a Patient-Centered Health-Data-Sharing Platform”. In: *NPJ Digital Medicine* 3 (Apr. 20, 2020), p. 60. ISSN: 2398-6352. DOI: 10.1038/s41746-020-0265-z. pmid: 32352038. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7170944/ (visited on 08/29/2022).

[24] Whitfield Diffie and Martin E. Hellman. “New Directions in Cryptography”. In: *IEEE Trans. Inf. Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[25] Halbert L. Dunn. “Census—Past and Future”. In: *Journal of the American Statistical Association* 35 (209b Mar. 1940), pp. 242–251. ISSN: 0162-1459, 1537-274X. DOI: 10.1080/01621459.1940.10500562. URL: http://www.tandfonline.com/doi/abs/10.1080/01621459.1940.10500562 (visited on 08/30/2022).

[26] Cynthia Dwork. “Differential Privacy: A Survey of Results”. In: *Theory and Applications of Models of Computation*. Ed. by Manindra Agrawal et al. Vol. 4978. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–19. ISBN: 978-3-540-79227-7 978-3-540-79228-4. DOI: 10.1007/978-3-540-79228-4_1. URL: http://link.springer.com/10.1007/978-3-540-79228-4_1 (visited on 08/30/2022).

[27] Keita Emura. “Privacy-Preserving Aggregation of Time-Series Data with Public Verifiability from Simple Assumptions”. In: *Information Security and Privacy*. Ed. by Josef Pieprzyk and Suriadi Suriadi. Vol. 10343. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 193–213. ISBN: 978-3-319-59869-7 978-3-319-59870-3. DOI: 10.1007/978-3-319-59870-3_11. URL: http://link.springer.com/10.1007/978-3-319-59870-3_11 (visited on 02/22/2022).

[28] Zekeriya Erkin and Gene Tsudik. “Private Computation of Spatial and Temporal Power Consumption with Smart Meters”. In: *Applied Cryptography and Network Security - 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings*. Ed. by Feng Bao, Pierangela Samarati, and Jianying Zhou. Vol. 7341. Lecture Notes in Computer Science. Springer, 2012, pp. 561–577. DOI: 10.1007/978-3-642-31284-7_33. URL: https://doi.org/10.1007/978-3-642-31284-7%5C_33 (visited on 09/13/2022).

[29] Xi Fang et al. “Smart Grid — The New and Improved Power Grid: A Survey”. In: *IEEE Communications Surveys & Tutorials* 14.4 (2012), pp. 944–980. ISSN: 1553-877X. DOI: 10.1109/SURV.2011.101911.00087.

[30] Amos Fiat and Adi Shamir. “How to Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 186–194.

[31] Liz Freeman. *Anthem Settles a Security Breach Lawsuit Affecting 80M*. USA TODAY. June 26, 2017. URL: https://www.usatoday.com/story/money/business/2017/06/26/anthem-settles-security-breach-lawsuit-affecting-80m/103217152/ (visited on 10/06/2022).

[32] Jiri Fridrich. “Methods for Data Hiding”. In: *Center for Intelligent Systems & Department of Systems Science and Industrial Engineering, SUNY Binghamton, Methods for Data Hiding”, working paper* (1997).

[33] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. “Pairings for Cryptographers”. In: *Discrete Applied Mathematics* 156.16 (Sept. 2008), pp. 3113–3121. ISSN: 0166218X. DOI: 10.1016/j.dam.2007.12.010. URL: https://linkinghub.elsevier.com/retrieve/pii/S0166218X08000449 (visited on 06/29/2022).

[34] Flavio D. Garcia and Bart Jacobs. “Privacy-Friendly Energy-Metering via Homomorphic Encryption”. In: *Security and Trust Management - 6th International Workshop, STM 2010, Athens, Greece, September 23-24, 2010, Revised Selected Papers*. Ed. by Jorge Cuéllar, Gilles Barthe, and Alexander Pretschner. Vol. 6710. Lecture Notes in Computer Science. Springer, 2010, pp. 226–238. DOI: 10.1007/978-3-642-22444-7_15. URL: https://doi.org/10.1007/978-3-642-22444-7%5C_15 (visited on 10/06/2022).

[35] *General Data Protection Regulation (GDPR) – Official Legal Text*. General Data Protection Regulation (GDPR). URL: https://gdpr-info.eu/ (visited on 10/06/2022).

[36] Craig Gentry. “Fully Homomorphic Encryption Using Ideal Lattices”. In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. Ed. by Michael Mitzenmacher. ACM, 2009, pp. 169–178. DOI: 10.1145/1536414.1536440.

[37] Henri Gilbert and Helena Handschuh. “Security Analysis of SHA-256 and Sisters”. In: *Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003, Revised Papers*. Ed. by Mitsuru Matsui and Robert J. Zuccherato. Vol. 3006. Lecture Notes in Computer Science. Springer, 2003, pp. 175–193. DOI: 10.1007/978-3-540-24654-1_13. URL: https://doi.org/10.1007/978-3-540-24654-1%5C_13 (visited on 10/10/2022).

[38] Glenn Greenwald and Ewen MacAskill. “NSA Prism Program Taps in to User Data of Apple, Google and Others”. In: *The Guardian. US news* (June 7, 2013). ISSN: 0261-3077. URL: https://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data (visited on 10/16/2022).

[39] Lovesh Harchandani. *Bulletproofs*. Feb. 4, 2020. URL: https://github.com/lovesh/bulletproofs-amcl (visited on 10/10/2022).

[40] G.W. Hart. “Nonintrusive Appliance Load Monitoring”. In: *Proceedings of the IEEE* 80.12 (Dec. 1992), pp. 1870–1891. ISSN: 1558-2256. DOI: 10.1109/5.192069.

[41] *Healthcare Data Breach Statistics - Latest Data for 2022*. HIPAA Journal. URL: https://www.hipaajournal.com/healthcare-data-breach-statistics/ (visited on 09/19/2022).

[42] Jason Horowitz. “Italy Announces Restrictions Over Entire Country in Attempt to Halt Coronavirus”. In: *The New York Times. World* (Mar. 9, 2020). ISSN: 0362-4331. URL: https://www.nytimes.com/2020/03/09/world/europe/italy-lockdown-coronavirus.html (visited on 10/06/2022).

[43] Bret Hull et al. “CarTel: A Distributed Mobile Sensor Computing System”. In: *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems - SenSys ’06*. The 4th International Conference. Boulder, Colorado, USA: ACM Press, 2006, p. 125. ISBN: 978-1-59593-343-0. DOI: 10.1145/1182807.1182821. URL: http://portal.acm.org/citation.cfm?doid=1182807.1182821 (visited on 08/29/2022).

[44] Jingyao Fan, Qinghua Li, and Guohong Cao. “Privacy-Aware and Trustworthy Data Aggregation in Mobile Sensing”. In: *2015 IEEE Conference on Communications and Network Security (CNS)*. 2015 IEEE Conference on Communications and Network Security (CNS). Florence, Italy: IEEE, Sept. 2015, pp. 31–39. ISBN: 978-1-4673-7876-5. DOI: 10.1109/CNS.2015.7346807. URL: https://ieeexplore.ieee.org/document/7346807 (visited on 05/05/2022).

[45] HIPAA Journal. *What Are the Penalties for HIPAA Violations? 2022 Update*. HIPAA Journal. Jan. 23, 2022. URL: https://www.hipaajournal.com/what-are-the-penalties-for-hipaa-violations-7096/ (visited on 08/31/2022).

[46] Bonnie Kaplan. “How Should Health Data Be Used?” In: *Cambridge quarterly of healthcare ethics: CQ: the international journal of healthcare ethics committees* 25.2 (Apr. 2016), pp. 312–329. ISSN: 1469-2147. DOI: 10.1017/S0963180115000614. pmid: 26957456.

[47] Diptendu M. Kar and Indrajit Ray. *Systematization of Knowledge and Implementation: Short Identity-Based Signatures*. Aug. 14, 2019. arXiv: 1908.05366 [cs]. URL: http://arxiv.org/abs/1908.05366 (visited on 07/14/2022).

[48] Ferhat Karakoç, Melek Önen, and Zeki Bilgin. “Secure Aggregation Against Malicious Users”. In: *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*. SACMAT ’21: The 26th ACM Symposium on Access Control Models and Technologies. Virtual Event Spain: ACM, June 11, 2021, pp. 115–124. ISBN: 978-1-4503-8365-3. DOI: 10.1145/3450569.3463572. URL: https://dl.acm.org/doi/10.1145/3450569.3463572 (visited on 02/17/2022).

[49] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. 2nd edition. Chapman & Hall/CRC Cryptography and Network Security. Boca Raton: CRC Press, 2015. ISBN: 978-1-4665-7026-9.

[50] Taechan Kim and Razvan Barbulescu. “Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case”. In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, 2016, pp. 543–571. DOI: 10.1007/978-3-662-53018-4_20. URL: https://doi.org/10.1007/978-3-662-53018-4%5C_20 (visited on 08/04/2022).

[51] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. “Privacy-Friendly Aggregation for the Smart-Grid”. In: *Privacy Enhancing Technologies*. Ed. by Simone Fischer-Hübner and Nicholas Hopper. Red. by David Hutchison et al. Vol. 6794. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 175–191. DOI: 10.1007/978-3-642-22263-4_10. URL: http://link.springer.com/10.1007/978-3-642-22263-4_10 (visited on 12/02/2021).

[52] Troy Lennon. *Babylon’s Tablets Made More Census than Today’s Computers*. dailytelegraph. Aug. 10, 2016. URL: https://www.dailytelegraph.com.au/news/babylons-ancient-clay-tablets-made-more-census-than-todays-computers/news-story/3f76510db70c6bfd1185192a2e90badc (visited on 10/06/2022).

[53] Iraklis Leontiadis and Ming Li. “Secure and Collusion-Resistant Data Aggregation from Convertible Tags”. In: *International Journal of Information Security* 20.1 (Feb. 2021), pp. 1–20. ISSN: 1615-5262, 1615-5270. DOI: 10.1007/s10207-019-00485-4. URL: http://link.springer.com/10.1007/s10207-019-00485-4 (visited on 02/17/2022).

[54] Iraklis Leontiadis et al. “PUDA – Privacy and Unforgeability for Data Aggregation”. In: *Cryptology and Network Security*. Ed. by Michael Reiter and David Naccache. Vol. 9476. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 3–18. DOI: 10.1007/978-3-319-26823-1_1. URL: http://link.springer.com/10.1007/978-3-319-26823-1_1 (visited on 02/22/2022).

[55] Fengjun Li, Bo Luo, and Peng Liu. “Secure Information Aggregation for Smart Grids Using Homomorphic Encryption”. In: *2010 First IEEE International Conference on Smart Grid Communications*. 2010 1st IEEE International Conference on Smart Grid Communications (SmartGridComm). Gaithersburg, MD, USA: IEEE, Oct. 2010, pp. 327–332. ISBN: 978-1-4244-6510-1. DOI: 10/c3grjd. URL: http://ieeexplore.ieee.org/document/5622064/ (visited on 12/02/2021).

[56] Hongjuan Li et al. “Secure and Energy-Efficient Data Aggregation with Malicious Aggregator Identification in Wireless Sensor Networks”. In: *Future Generation Computer Systems* 37 (July 2014), pp. 108–116. ISSN: 0167739X. DOI: 10.1016/j.future.2013.12.021. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167739X13002859 (visited on 02/17/2022).

[57] Yongkai Li et al. “Collusion-Tolerable and Efficient Privacy-Preserving Time-Series Data Aggregation Protocol”. In: *International Journal of Distributed Sensor Networks* 12.7 (July 1, 2016), p. 1341606. ISSN: 1550-1477, 1550-1477. DOI: 10.1177/155014771341606. URL: http://journals.sagepub.com/doi/10.1177/155014771341606 (visited on 04/15/2022).

[58] Rongxing Lu et al. “EPPA: An Efficient and Privacy-Preserving Aggregation Scheme for Secure Smart Grid Communications”. In: *IEEE Transactions on Parallel and Distributed Systems* 23.9 (Sept. 2012), pp. 1621–1631. ISSN: 1558-2183. DOI: 10/f35tmf.

[59] B. A. Malin, K. E. Emam, and C. M. O’Keefe. “Biomedical Data Privacy: Problems, Perspectives, and Recent Advances”. In: *Journal of the American Medical Informatics Association* 20.1 (Jan. 1, 2013), pp. 2–6. ISSN: 1067-5027, 1527-974X. DOI: 10.1136/amiajnl-2012-001509. URL: https://academic.oup.com/jamia/article-lookup/doi/10.1136/amiajnl-2012-001509 (visited on 08/30/2022).

[60] Diego Méndez et al. “P-Sense: A Participatory Sensing System for Air Pollution Monitoring and Control”. In: *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops). Mar. 2011, pp. 344–347. DOI: 10.1109/PERCOMW.2011.5766902.

[61] Andres Molina-Markham et al. “Designing Privacy-preserving Smart Meters with Low-cost Microcontrollers”. In: *IACR Cryptol. ePrint Arch.* (2011), p. 544. URL: http://eprint.iacr.org/2011/544 (visited on 10/04/2022).

[62] Chris Morris. *LinkedIn Data Theft Exposes Personal Information of 700 Million People*. Fortune. 2021. URL: https://fortune.com/2021/06/30/linkedin-data-theft-700-million-users-personal-information-cybersecurity/ (visited on 10/16/2022).

[63] Dimitris Mouris and Nektarios Georgios Tsoutsos. *Masquerade: Verifiable Multi-Party Aggregation with Secure Multiplicative Commitments*. 2021. URL: https://eprint.iacr.org/2021/1370 (visited on 10/06/2022).

[64] A. Muir and J. Lopatto. “Final Report on the August 14, 2003 Blackout in the United States and Canada : Causes and Recommendations”. In: (Apr. 1, 2004). URL: https://www.osti.gov/etdeweb/biblio/20461178 (visited on 10/10/2022).

[65] Jianbing Ni et al. “Security-Enhanced Data Aggregation against Malicious Gateways in Smart Grid”. In: *2015 IEEE Global Communications Conference, GLOBECOM 2015, San Diego, CA, USA, December 6-10, 2015*. IEEE, 2015, pp. 1–6. DOI: 10.1109/GLOCOM.2014.7417140.

[66] Tatsuaki Okamoto. “Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes”. In: *Annual International Cryptology Conference*. Springer, 1992, pp. 31–53.

[67] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *Advances in Cryptology - EUROCRYPT ’99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16. URL: https://doi.org/10.1007/3-540-48910-X%5C_16 (visited on 09/30/2022).

[68] Torben P. Pedersen. “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing”. In: *Advances in Cryptology - CRYPTO ’91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Springer, 1991, pp. 129–140. DOI: 10.1007/3-540-46766-1_9. URL: https://doi.org/10.1007/3-540-46766-1%5C_9 (visited on 09/30/2022).

[69] Cong Peng et al. “An Efficient Privacy-Preserving Aggregation Scheme for Multidimensional Data in IoT”. In: *IEEE Internet of Things Journal* 9.1 (Jan. 2022), pp. 589–600. ISSN: 2327-4662. DOI: 10.1109/JIOT.2021.3083136.

[70] Keith S. Reid-Green. “The History of Census Tabulation”. In: *Scientific American* 260.2 (Feb. 1989), pp. 98–103. ISSN: 0036-8733. DOI: 10.1038/scientificamerican0289-98. URL: https://www.scientificamerican.com/article/the-history-of-census-tabulation (visited on 08/29/2022).

[71] Office for Civil Rights (OCR). *Health Information Privacy*. HHS.gov. URL: https://www.hhs.gov/hipaa/index.html (visited on 08/31/2022).

[72] Charles Riley. *Insurance Giant Anthem Hit by Massive Data Breach*. CNNMoney. Feb. 4, 2015. URL: https://money.cnn.com/2015/02/04/technology/anthem-insurance-hack-data-security/index.html (visited on 10/06/2022).

[73] Mark A. Rothstein. “Genetic Privacy and Confidentiality: Why They Are So Hard to Protect”. In: *Journal of Law, Medicine & Ethics* 26.3 (1998), pp. 198–204. ISSN: 1073-1105, 1748-720X. DOI: 10.1111/j.1748-720X.1998.tb01420.x. URL: https://www.cambridge.org/core/product/identifier/S107311050001305X/type/journal_article (visited on 08/31/2022).

[74] Yannis Rouselakis and Brent Waters. “Practical Constructions and New Proof Methods for Large Universe Attribute-Based Encryption”. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security - CCS ’13*. The 2013 ACM SIGSAC Conference. Berlin, Germany: ACM Press, 2013, pp. 463–474. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516672. URL: http://dl.acm.org/citation.cfm?doid=2508859.2516672 (visited on 07/14/2022).

[75] Yumi Sakemi et al. *Pairing-Friendly Curves*. Internet Draft draft-irtf-cfrg-pairing-friendly-curves-08. Internet Engineering Task Force, 2020. 54 pp. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-pairing-friendly-curves-08 (visited on 08/04/2022).

[76] Berry Schoenmakers. “Lecture Notes Cryptographic Protocols”. In: *Department of Mathematics and Computer Science, Technical University of Eindhoven, version* 1 (2022), p. 142. URL: https://www.win.tue.nl/~berry/CryptographicProtocols/LectureNotes.pdf.

[77] Kent Seamons. “Privacy-Enhancing Technologies”. In: *Modern Socio-Technical Perspectives on Privacy*. Ed. by Bart P. Knijnenburg et al. Cham: Springer International Publishing, 2022, pp. 149–170. ISBN: 978-3-030-82785-4 978-3-030-82786-1. DOI: 10.1007/978-3-030-82786-1_8. URL: https://link.springer.com/10.1007/978-3-030-82786-1_8 (visited on 09/19/2022).

[78] Adi Shamir. “How to Share a Secret”. In: *Communications of the ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/359168.359176. URL: https://dl.acm.org/doi/10.1145/359168.359176 (visited on 09/18/2022).

[79] Elaine Shi et al. “Privacy-Preserving Aggregation of Time-Series Data”. In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*. The Internet Society, 2011. URL: https://www.ndss-symposium.org/ndss2011/privacy-preserving-aggregation-of-time-series-data (visited on 10/10/2022).

[80] Katie Shilton. “Four Billion Little Brothers?: Privacy, Mobile Phones, and Ubiquitous Data Collection”. In: *Communications of the ACM* 52.11 (Nov. 2009), pp. 48–53. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/1592761.1592778. URL: https://dl.acm.org/doi/10.1145/1592761.1592778 (visited on 08/29/2022).

[81] Nigel P. Smart. *Cryptography Made Simple*. Information Security and Cryptography. Cham: Springer International Publishing, 2016. ISBN: 978-3-319-21935-6 978-3-319-21936-3. DOI: 10.1007/978-3-319-21936-3. URL: http://link.springer.com/10.1007/978-3-319-21936-3 (visited on 02/19/2021).

[82] *The Uses of Census Data: An Analytical Review*. URL: https://clintonwhitehouse4.archives.gov/WH/EOP/CEA/html/censusreview.html (visited on 08/29/2022).

[83] Sameer Tilak. “Real-World Deployments of Participatory Sensing Applications: Current Trends and Future Directions”. In: *ISRN Sensor Networks* 2013 (May 27, 2013), e583165. DOI: 10.1155/2013/583165. URL: https://www.hindawi.com/journals/isrn/2013/583165/ (visited on 08/29/2022).

[84] Ngoc Hieu Tran, Robert H. Deng, and HweeHwa Pang. “Privacy-Preserving and Verifiable Data Aggregation”. In: *Proceedings of the Singapore Cyber-Security Conference (SG-CRC) 2016 - Cyber-Security by Design, Singapore, January 14-15, 2016*. Ed. by Aditya Mathur and Abhik Roychoudhury. Vol. 14. Cryptology and Information Security Series. IOS Press, 2016, pp. 115–122. DOI: 10.3233/978-1-61499-617-0-115.

[85] Ata Ullah et al. “Secure Healthcare Data Aggregation and Transmission in IoT—A Survey”. In: *IEEE Access* 9 (2021), pp. 16849–16865. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3052850.

[86] Stephane Vincent. “Exploring Pairing Based Cryptography”. In: (Dec. 2018), p. 6. URL: https://www.sikoba.com/docs/SKOR_SV_Pairing_Based_Crypto.pdf.

[87] *Your Phone Is Now More Powerful than Your PC*. URL: https://insights.samsung.com/2021/08/19/your-phone-is-now-more-powerful-than-your-pc-3/ (visited on 10/06/2022).

[88] Lei Zhang and Jing Zhang. “Publicly Verifiable Spatial and Temporal Aggregation Scheme Against Malicious Aggregator in Smart Grid”. In: *Applied Sciences* 9.3 (Jan. 31, 2019), p. 490. ISSN: 2076-3417. DOI: 10.3390/app9030490. URL: http://www.mdpi.com/2076-3417/9/3/490 (visited on 02/17/2022).

[89] Gaoqiang Zhuo et al. “Privacy-Preserving Verifiable Data Aggregation and Analysis for Cloud-Assisted Mobile Crowdsourcing”. In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. IEEE INFOCOM 2016 - IEEE Conference on Computer Communications. San Francisco, CA, USA: IEEE, Apr. 2016, pp. 1–9. ISBN: 978-1-4673-9953-1. DOI: 10.1109/INFOCOM.2016.7524547. URL: http://ieeexplore.ieee.org/document/7524547/ (visited on 05/05/2022).

A

mPVAS+ Group Probabilities

In this appendix, we collect some graphs showing the probabilities of obtaining a group where all users are malicious for various group sizes, ranging from 2 to 16, and different percentages of malicious users, $0.05\%, 0.10\%, 0.20\%, 0.30\%$. Each graph shows the simulated probability distribution for six settings in which the number of users is, respectively, $50, 100, 250, 500$, and 1000. The probabilities were computed using a simulation with $100,000$ trials over which the average number of times at least one group was fully malicious was computed.

The graphs are presented in Figure A.1.

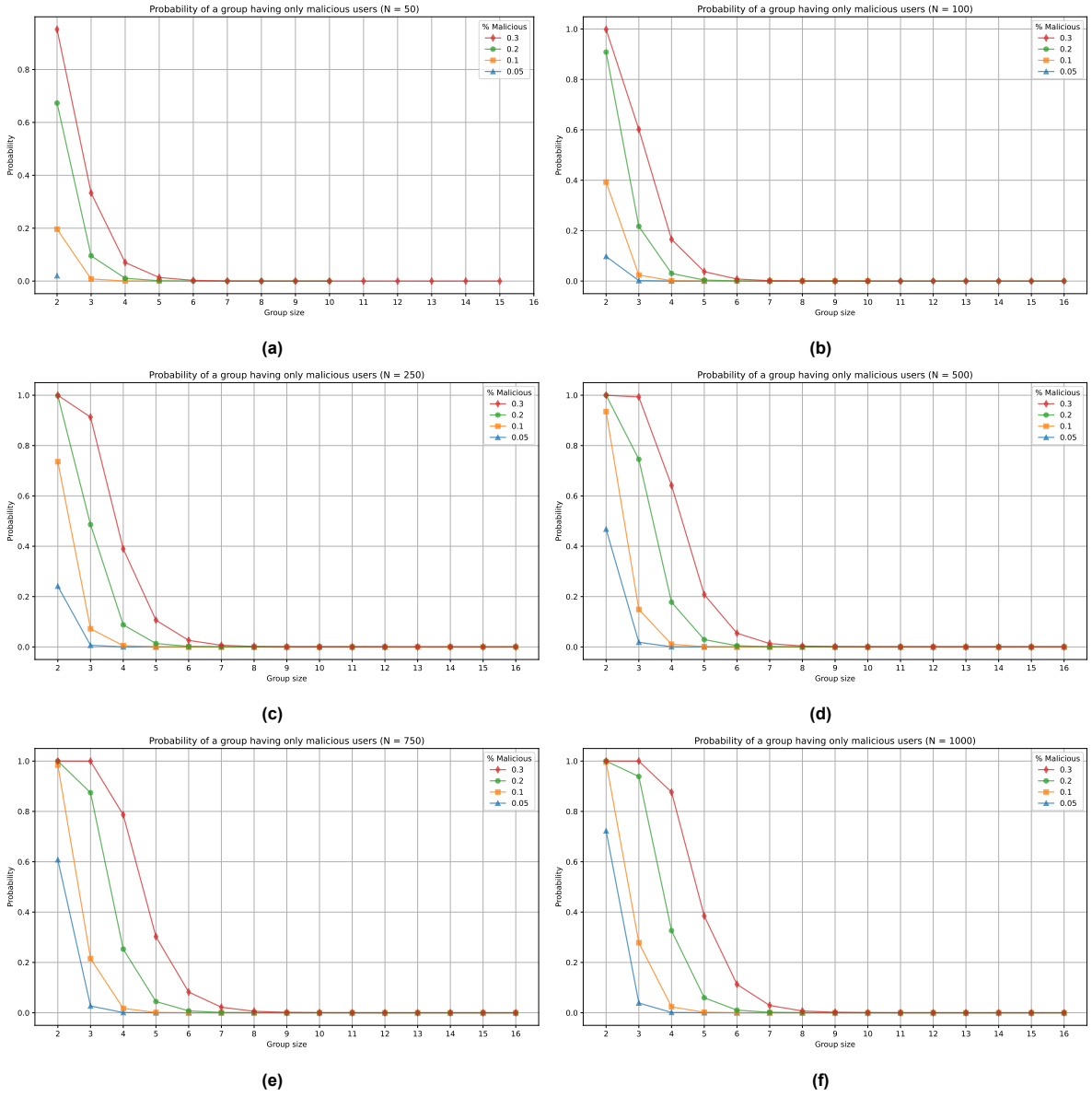


Figure A.1: Simulated distributions of the probability of obtaining at least one fully malicious group for different numbers of users, group sizes, and percentages of malicious users.