Multi-Functional Privacy-Preserving Data Aggregation

with Malicious User Detection

C. M. Koster

TU Delft

Multi-Functional Privacy-Preserving Data Aggregation

with Malicious User Detection

by

C. M. Koster

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday May 28, 2021 at 13:00 AM.

Student number:	4371011	
Project duration:	September 7, 2020 – May 28, 2021	
Thesis committee:	Dr. Z. Erkin,	TU Delft, supervisor
	Dr. ir. C. C. S. Liem,	TU Delft
	Dr. K. Liang,	TU Delft

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

TU Delft

Abstract

In practice, many applications like traffic monitoring and smart grids rely on computing functions on privacy-sensitive data. In order to protect privacy-sensitive data and still keep the ability to compute any arbitrary function, multi-functional privacy-preserving data aggregation schemes have been created. These schemes, however, can be abused by malicious users, leading to incorrect results. Existing literature mostly provides aggregation schemes which are either multi-functional or support malicious user detection, but to the best of our knowledge, there is only one scheme that provides both. This scheme requires each user to send a number of ciphertexts linear in the size of the aggregation function's domain. Furthermore, that scheme is not collusion-resistant. In this thesis, we design a multi-functional privacy-preserving data aggregation scheme with malicious user detection. In contrast to existing schemes in literature, the amount of messages is independent of the size of the aggregation function's domain, it does not rely on a trusted authority and it is collusion-resistant as long as at least two users are honest-but-curious.

Preface

In this report I describe the protocol that I have designed for my master thesis. This protocol allows a party to aggregate private data without leaking any information of that private data. Any aggregation function is supported to be computed and if any user is acting maliciously, that user is detected. The combination of those characteristics and the resulting communication complexity results in a protocol which improves upon the state-of-the-art in this field.

The protocol is the result of a period of nine months of research. The first three months, I read up on a lot of privacy-preserving data aggregation schemes. At first, it was the challenge to understand the different techniques that are used in such schemes. Later on, I started recognizing those techniques and recognized patterns that are repeated in those schemes. The challenge became to search for differences with other schemes and how those differences are caused. Sometimes the difference was as small as a slightly better performance. During this reading period, I have encountered a lot of topics I found interesting to look into. Near the end of the reading period, I read a paper which allowed a party to compute any arbitrary aggregation function. The way this is achieved in that paper is very inefficient which made me interested to look into other papers that support any arbitrary aggregation function. After reading those papers, I was motivated to try to come up with a solution that improves upon them in terms of complexity. After the reading period, a solution already came up quite quickly. After working out this solution it seemed to fit and it also had the desired characteristics. However, there was something missing in my opinion. The users of the protocol were assumed to be honest-but-curious. As an extra characteristic, I wanted to detect a user when it is behaving maliciously. This has been the main part of the next period, the design period, of my thesis. During the design period, I designed the protocol and analyzed the properties of the resulting properties. Once everything was worked out, I started the next and last period of my thesis, the writing period. This period started with writing a paper for the ARES conference which is unfortunately rejected. After the paper, I started writing this thesis.

In the first place I would like to thank Zekeriya Erkin for all the support he gave me during my thesis. I am also thankful for all the technical and less technical insights he gave me. I really enjoyed having him as my supervisor. I would also like to thank Felix W. Dekker for all the help during the designing period of my protocol and especially during the writing period for all the feedback on my writings. I also thank Cynthia Liem and Kaitai Liang for being a member of my thesis committee. Next, I thank all the other master students and PhD'ers that are supervised by Zekeriya for all the technical and enjoyable less technical discussions. I really had a great time working with all of you. Finally, I want to thank my family and girlfriend who supported me in this period.

C. M. Koster
Delft, May 2021

Contents

1 Introduction

Data plays a vital role in our society. Data about for example locations and speed of people are used to detect whether there is a traffic jam. With this information, people are redirected to reach their destination faster. Also, driving less in traffic jam decreases air pollution and decreases health risks for drivers [43]. Another example that emphasizes the importance of data is found in flooding systems. A flooding system needs, among others, data about the water level and certain aspects of the weather such as the direction and the strength of the wind. With no or incorrect data, the consequences could be catastrophic such as the North Sea flood in 1953. In that time, there were no flooding systems which resulted in large parts of Zeeland, which is a province in the Netherlands, being flooded [64].

One source of data is smart devices. Smart devices contain sensors which measure external aspects of their environment such as a Fitbit for measuring the heart rate or a smart meter to measure the energy consumption. A device is smart when it is connected with a set of other devices and networks [61]. The amount of smart devices is increasing rapidly [37] and therefore the amount of data is also increasing. In 2020, the worldwide data volume is already 59 zettabytes [36].

Data might be privacy-sensitive such as in the case of the location of a person. Location data allows a person to be followed anywhere and information about his or her habits can be deduced. To preserve the data privacy, encryption schemes can be used. An encryption schemes encrypts private data so that other users, without the private key, cannot gain any knowledge about the private data. However, encryption schemes does not allow a user, without the private key, to get the result of a function computed over the private data. In other words, the utility is lost. An application is no longer able to use the encrypted data to detect for example a traffic jam. On the other hand, not encrypting anything gives a perfect utility, since anything can be done with the data, but the data privacy is not preserved. Therefore, there is a constant trade-off between utility and data privacy. Privacy-preserving data aggregation schemes achieve both. Those schemes preserve the data privacy and gives the utility provider the possibility to aggregate the private data.

1.1. Data aggregation

With data aggregation, any arbitrary function over a set of data is computed such as the sum, average or principal component analysis. Data aggregation is useful in many applications for different types of users like governments, companies and individuals. In this section we highlight examples of applications which illustrates the usefulness of data aggregation.

We start with an example of data aggregation which shows the usefulness for a government. A government has to make decisions all the time. A contemporary example of this is the COVID-19 pandemic. During the pandemic, data aggregation is applied to get for example the number of infections in a week, the amount of patients in the hospitals or intensive care departments and a so-called r-score which indicates how many people a single person infects on average. The government is now able to take action upon the information resulting from data aggregation. When the infections are increasing, for example, the government might need to introduce stricter measures [45]. Without data aggregation, it is much harder to get same level of information, such as the r-score, about the data. Data aggregation, therefore, helps a government to take appropriate and substantiated actions on time and reduced the amount of infected persons and deaths [39].

Data aggregation also comprehends predictions that are made. A government may know that new measures

have to be taken, but it must also be known which measures to take. Since there are no past research results about the consequences of a measure in a pandemic of this magnitude, the consequences need to be predicted. With the help of predictions, a government was also able to estimate the capacity that was necessary in hospitals and intensive-care departments [60]. The pro-active increasing of the capacities led to the fact that everyone had the opportunity to be treated rightfully, which again saved many lives [53].

Next to a government, also companies benefit from data aggregation. This benefit is two-fold. The first benefit for a company is to use data aggregation for its own purposes. An example of this is a supermarket. The stock is a high risk for a supermarket since it can, for example, expire or get stolen. Therefore, a supermarket wants to minimize the stock. On the other hand, a supermarket wants to have enough stock so that customers are able to buy everything they want. With the help of the data of previous months or years, data aggregation is used to predict which amount of which product a supermarket must buy for the next period in order to satisfy the needs of every client and not to have too much stock. As a result, supermarkets decreased the amount of food they had to throw away because it is expired [50].

Instead of using data aggregation for own purposes, a company can also use data aggregation as its core business. Since data aggregation is so useful for a large variety of entities, companies earn their money with data aggregation as their business. This results in a wider variety of data aggregation which offers new insights. Google Maps, for example, computed that amount of cars on the road increased during a later stage of the lockdown [2]. This new insight could again be used by the government to take new measures in order to prevent the spread of the COVID-19 virus.

Individuals also benefit from data aggregation. The smart grid [26] is one such an example. In a smart grid, the energy consumption in households is measured by smart meters. These smart meters send the data to the utility center. With the knowledge of the energy consumption per household on a frequent basis, the utility company better balances the provision of energy through the energy grid. The better balance prevents blackouts like the Northeast blackout [5].

Finally, individuals also benefit from data aggregation in their daily lives. Popular examples are the traffic jam notifications by Google Maps, an indication of how busy a supermarket is at which time or recommendations of restaurants or other places by TripAdvisor. All those examples are generated with data aggregation and turned into information which users use to optimize their activities. This information helps individuals to make decisions in one moment instead of going through lots of data.

1.2. Privacy

Data aggregation is valuable for everyone, for the government and business, but also for individuals. However, not all data can be used as it is for data aggregation. Privacy-sensitive data cannot be used as plain text in an aggregation protocol, since the party aggregating the data gains knowledge of the sensitive data. Many of the examples in the previous section process such privacy-sensitive data.

The data concerning infections of COVID-19 or patients staying in the hospital or the intensive care, for example, is privacy-sensitive. In general, data concerning any disease or other health related issues is privacy-sensitive. A patient does not always want others to know its health information. Disclosure of health data can lead to discrimination and infringements of fundamental rights [48]. An example of discrimination is that the costs of a health insurance for people with a known underlying disease will be higher in comparison to people where no disease is known, even though the people with no known disease might have the same underlying disease.

Smart grids also processes privacy-sensitive data. If the energy consumption is known for short consecutive periods, the aggregator is able to derive, among others, the presence of inhabitants, the presence of a visitor, the applications that are turned on and even customers' religions [70].

Another example processing privacy-sensitive data is Google Maps. Google Maps, as discussed above, notifies users of traffic jams. In order for Google Maps to be aware of a traffic jam, it has to keep track of users. Where many users are driving slowly a traffic jam has emerged. However, this tracking data is privacy-sensitive. With this tracking data, someone is able to get information about the whereabouts of any user. For example, where someone works, where someone lives or when someone is not at home.

Privacy-sensitive data includes different categories of data [44]. The first category is the personal data. Personal data includes all information that is directly related to an individual, such as address, license plate of a car and a name. The second category is special personal data. Data in this category is extra sensitive to affect the individual. Examples are race, sexual orientation, religion, criminal record and more [1]. Not all this data does directly link to an individual, but with the combination of other data it might be linkable. Finally, besides

the data of a natural person, privacy-sensitive data also includes sensitive data of legal persons [44].

Therefore, when storing and processing privacy-sensitive data, extra care has to be taken to preserve the privacy. The storage and processing of sensitive data is regulated with legislation which is done in multiple jurisdictions. The European Union, for example, regulates it with the General Data Protection Regulation (GDPR) [1], which is in effect since 2018. The GDPR also affects citizens and companies outside the European Union. The regulation states, among others, that there has to be a legal ground in order to store and process sensitive data. Examples of such legal ground are the explicit consent of the data subject, the person to who the data relates, and the urgency for national security. When a processor, the entity who processes the data, has this legal ground, it is still subject to limitations imposed on them by the GDPR. For example, the processor may only process the sensitive information for the purpose for which the data was acquired. Also, the data must be removed from the storage as soon as possible.

1.3. Privacy by design

Legislation alone is not enough to enforce the privacy of sensitive data. If a company legally stores the sensitive data, it is still susceptible to internal and external attacks. It is even possible that the company is acquired by an adversarial company. Therefore, the GDPR also states that organisational and technical measures should be taken as much as possible, taking into account the costs of implementation and the context and scope of the processing. These measures should already be determined while designing the process or application and not only during processing. Now the core goal of the process or application becomes to prevent privacy violations while maintaining the other functionality. This methodology is called privacy by design.

Privacy by design can make use of Privacy-Enhancing Technologies (PETs) [56]. There are numerous PETs designed in the past. Cha et al. [14] give an overview of a subset of those PETs with each their objective. One type of PETs are the anonymity-related PETs [68], which can have multiple guarantees. The first guarantee is pseudonymity, which guarantees that the identity of a user is not revealed. However, it can still be tracked. The second guarantee is anonymity, which guarantees that the identity is not revealed and cannot be tracked. An anonymity related PET can also guarantee the unobservability, which guarantees that one is unable to derive whether an application is used by a given user. Another guarantee is the unlinkability which guarantees that two steps in a process cannot be linked to the same user. Finally, the last possible guarantee discussed in [68] is deniability. Deniability gives users the possibility to deny a claim about them which someone else on its turn is unable to verify. Other types of PETs focus less on the identity, but more on the data itself. There are PETs which focus on the guarantee of confidentiality such as the AES [20] and Paillier [51] encryption schemes. Or a PET can provide a guarantee for the authenticity of the data for example with a signature.

A subset of PETs focus on providing extra functionality while keeping the guarantee of confidentiality. One example is searchable encryption. Searchable encryption allows someone to search within a database of encrypted items for a specific item. Searchable encryption has numerous characteristics such as the level of expressiveness of the search queries and the level of confidentiality [13]. Access control is another PET providing confidentiality and still providing utility. In these type of PETs, only parties with the right set of attributes are able to decrypt the data. Other parties are only able to see the encrypted data [47]. The previous two types of PETs, searchable encryption and access control, require a central database for the encrypted data. There are scenarios in which such a central place is not available or not wanted. Still users may want to compute a function while keeping the confidentiality of their data. A category of PETs guaranteeing this is multiparty computation [19]. In multiparty computation schemes, the users compute the result of any public function jointly while preserving the data privacy.

The type of PETs that is of interest for us is the privacy-preserving data aggregation. In such PETs, users send their data encrypted according to the specifics of the protocol. The encrypted data is sent to a party who is called the aggregator. The aggregator performs computations to compute any arbitrary function on the incoming data such as the sum, the minimum, the maximum or any polynomial function. Without gaining any knowledge of the data of individual users, the aggregator gains the result of a function it wants to be performed on the data. Such PETs guarantee the confidentiality of the data, but also allow the aggregator to compute functions on the data.

1.4. Existing solutions

A wide range of privacy-preserving data aggregation schemes exist. The schemes vary in functionality and trust assumptions. The scheme in [41] assumes the aggregator is trusted. The scheme gives the aggregator the possibility to decrypt every value of every user so that it aggregates on plain text data. Giving the aggregator

this power is not desirable in every scenario. The sensitive information is still available. Whether the aggregator is really trusted does not matter. It is possible that an adversary finds its way to this sensitive data. Another subset of aggregation schemes make use of a trusted authority. A trusted authority can be used for setting up the system. The trusted authority provides every party with its own secret key as done by Shi et al. [57]. In [54], the trusted authority gets all encrypted data, decrypts them and relays the aggregated result encrypted under a different key to the party who wants the result. Here, the trusted authority is actually acting like the aggregator which is fully trusted. A trusted authority is useful since trusted computations can be outsourced to it. However, a trusted authority might not be realistic to have in practice. As said before, even when a party is fully trusted, there is still a potential threat from external adversaries.

Another problem many aggregation schemes have in common is that they only allow the protocol to compute the sum. The support of computing the sum indirectly also includes other functions, such as average, to be computed since it additionally requires the knowledge of the amount of users which is equal to the amount of incoming messages. However, the aggregator might want to compute other functions such as minimum, maximum or another arbitrary function. The computations of these functions are for example not possible in the schemes from Shi et al. [57], Lu et al. [42], Erkin and Tsudik [25] and Guan et al. [34].

There are privacy-preserving data aggregation schemes which support the computation of any arbitrary function. This kind of schemes are called multi-functional. However, existing multi-functional privacy-preserving data schemes have major drawbacks. The schemes presented by Gong et al. [32] and Zhang et al. [72] do not scale well in the number of users and the input range. The message size of a user grows linearly with the amount of users and the size of the data. The schemes presented by Wang et al. [66, 67] produce approximate results which might not be desirable in every application.

The schemes presented by Shi et al. [58] and Han et al. [35] also support arbitrary functions where the message size for each user is independent on the amount of users and the results are exact. However, in order to compute the histogram, which on its turn allows the computation of any arbitrary function, the protocol requires interaction. The users have to respond for every range of values if it is in that range. So the amount of messages grows linearly with the amount of possible values resulting in a higher bandwidth usage.

Zhang et al. [71] also present a scheme which supports any arbitrary function to be computed and does not have any of the above problems. However, also this scheme has its own drawbacks. First of all, the decryption requires the aggregator to compute the discrete log which is an inefficient operation. Above all, with the protocol described to compute functions like minimum and maximum, the exact value of any user can be retrieved which results in a loss of data privacy.

Besides all the mentioned drawbacks of the previous schemes, none of them assumes that a user of the system is malicious. For example, in the scheme of Shi et al. [58], a user is able to maliciously change its value every round of interaction which results in a wrong result. Next to a wrong result, malicious behaviour may also result in no results at all. Having wrong or no results can have serious consequences. For example in the case of the smart grid, energy can be allocated to the wrong part of the energy grid resulting in a blackout. To the best of our knowledge, there is only one scheme which both supports multi-functionality and malicious user detection. This scheme is presented by Viejo et al. [65]. This scheme has two main drawbacks. The first drawback is that the scheme is not collusion-resistant. The aggregator retrieves the data of an individual user when it colludes with any node between the user and the aggregator. The second drawback concerns the bandwidth usage. Every user needs to send a ciphertext for every possible value a user is able to send.

1.5. Research goal

In this thesis we aim to design a privacy-preserving data aggregation scheme which must satisfy multiple criteria. The first criterion is multi-functionality. The scheme must support the computation of any arbitrary function. In literature, the majority of schemes does not satisfy this criterion. The schemes that are multi-functional, suffer other drawbacks such as communication which is linear to the product of the amount of values and users or a loss of data privacy. In order to achieve this criterion, our scheme must not rely on a trusted authority which is allowed to aggregate the plaintext data. Such a trusted authority might not be realistic in all applications.

The second criterion, our privacy-preserving data aggregation scheme has to satisfy is the ability to detect malicious users. All but one multi-functional privacy-preserving data aggregation schemes assume users to be honest-but-curious. In real-world applications this might not always be the case. The scheme that supports malicious user detection, however, is not collusion-resistant. Also, this scheme requires communication which is linear to the product of the amount of values and users.

The third criterion is that our privacy-preserving data aggregation scheme is collusion-resistant. This criterion is the most important criteria to improve upon the state-of-the-art. To the best of our knowledge there is no multi-functional privacy-preserving data aggregation scheme with malicious user detection which also is collusion-resistant.

All these criteria lead to the following research question.

> How to design a privacy-preserving data aggregation scheme which is multi-functional, collusion-resistant and able to detect malicious users without relying on a trusted authority?

Due to the fact that comparable privacy-preserving data aggregation schemes have a communication complexity which is at least linear to the product of the amount of values and users, the goal for our scheme is to improve on that. The term of the amount of users is inevitable since every user at least has to send its data. Therefore in this thesis we aim to remove the term for the amount of values or replace it with a smaller term.

1.6. Contributions

In this thesis we propose three protocols. The first protocol is an encoding scheme. The encoding scheme maps the value of a user to a coefficient. The coefficients are constructed in such a way, that the amount of users which have sent a certain coefficient is deduced from the sum of all the coefficients sent by the users. A coefficient is then mapped back to a value so that it is known which amount of users have sent a specific value. Any privacy-preserving data aggregation scheme which uses an additive homomorphic encryption scheme is turned into a multi-functional scheme with the usage of this encoding scheme.

The second protocol uses the encoding scheme to create a privacy-preserving data aggregation scheme which is multi-functional. In this protocol we assume the users to be honest-but-curious and we focus to improve on the computation and communication complexity of the state-of-the-art. Compared to comparable privacy-preserving data aggregation schemes our scheme improves with respect to the communication and computation complexity.

The last protocol proposed in this thesis is a privacy-preserving data aggregation scheme which satisfies all criteria that are stated in the research goal in Section 1.5. This scheme is multi-functional, collusion-resistant and it supports malicious user detection. The multi-functionality is due to the usage of the encoding scheme. The second protocol, which assumed honest-but-curious users, is modified and extended with non-interactive zero-knowledge proofs so that it supports the detection of malicious users. The manner how all the coefficients of the encoding scheme are encrypted results in the fact that this last protocol is collusion-resistant. To the best of our knowledge, we are the first ones who propose a privacy-preserving aggregation scheme which is multi-functional, collusion-resistant and supports malicious user detection.

Besides fulfilling the criteria that are stated in the research goal, the last protocol also improves upon the state-of-the-art with respect to the computation and communication complexity. Instead of depending on the amount of values, our protocol depends on the group size. The advantage of the group size is that it is more flexible. The amount of values is often fixed in a certain application. The group size can be set depending on the requirements. Finally, both privacy-preserving data aggregation schemes do not rely on a trusted authority as required by the research goal.

In summary, our contributions are as follows.

- To the best of our knowledge we propose the first multi-functional privacy-preserving aggregation scheme which is collusion-resistant, supports malicious user detection and does not rely on a trusted authority.
- We propose two privacy-preserving data aggregation schemes, with different user assumptions, which improve upon the state-of-the-art with regard to the computation complexity and communication complexity.
- We propose an encoding scheme which is used to turn any privacy-preserving data aggregation scheme, which uses an additive homomorphic encryption scheme, into a multi-functional scheme.

1.7. Outline

We proceed in Chapter 2 with the discussion of the preliminary knowledge needed to understand the remainder of this thesis. Chapter 3 continues with the discussion of the related work. Next, Chapter 4 presents an encoding

scheme which is used in the privacy-preserving data aggregation scheme. Chapter 5 discusses the first multi-functional privacy-preserving data aggregation scheme. It, however, assumes users to be honest-but-curious and serves as an intermediate step towards the protocol discussed in Chapter 6 which supports malicious user detection. Finally, we conclude this thesis in Chapter 7.

2

Preliminaries

The subjects that are discussed in this chapter are superincreasing sequences, secret sharing, homomorphic encryption, the Boneh-Boyen signature scheme and zero-knowledge proofs..

2.1. Superincreasing sequence

A superincreasing sequence is a sequence of real numbers $\{\alpha_0, \alpha_1, \ldots, \alpha_m\}$ for which holds that each number is higher than the sum of the previous numbers in the sequence. This is formalized in the equation

$$\alpha_i > \sum_{j=0}^{i-1} \alpha_{j-1}. \tag{2.1}$$

Another condition that should hold for a sequence to be superincreasing, is $\alpha_0 > 0$. Since the first number is now positive, all the other numbers of a superincreasing sequence also have to be positive due to Equation 2.1.

Given a random subset of distinct elements $b = \{b_0, b_1, \ldots, b_n\} \subseteq \{\alpha_0, \alpha_1, \ldots, \alpha_m\}$, the sum function $f(b) = \sum_{i=0}^{n} b_i$ for such a set b is a bijective function. In other words, for every possible subset, f produces a unique outcome. Due to the fact that f is bijective, it is possible to compute the inverse of f, i.e. given $f(b)$, the sum of the subset elements, we are able to compute which elements are included in the subset. Starting at the largest element, one checks whether an element fits in the sum. If it does fit it is part of the sum and, in order to check the smaller elements, the current element is subtracted from the sum. If it does not fit it is not part of the sum. Formally, $f^{-1}(b)$ is computed as described in Algorithm 1, where u_i is 1 when element α_i is included in b and 0 otherwise.

Algorithm 1: Inverse sum of superincreasing sequence

Data: $f(b)$, $[\alpha_0, \alpha_1, \ldots, \alpha_m]$
Result: $[u_0, u_1, \ldots, u_m]$
$X_m = f(b)$;
for $i \leftarrow m$ **to** 0 **do**
    $X_{i-1} = X_i \bmod \alpha_i$;
    $u_i = \frac{X_i - X_{i-1}}{\alpha_i}$;

2.2. Secret sharing

There are scenarios where there is no party who is trusted to retrieve secret data on its own. That single party can easily turn malicious without any interference. Multiple parties together, however, are less likely to be corrupt at the same time. The secret data must be shared between the parties in such a way that only all parties together or a subset of parties is able to retrieve the secret. Note that a subset has a size bigger than 1. Otherwise sharing the secret is not needed in the first place. In order to behave maliciously, all parties, or the defined subset of parties, must collaborate which takes more effort for an adversary to accomplish. Any

amount of users less than required should not gain any information about the secret. A well-known example is about nuclear codes. A president on his/her own should not be able to retrieve the code to activate the nuclear weapons since all responsibility lies with the president. Instead, the nuclear code is retrieved when, for example, the president, vice-president and minister of defense collaborate. Now, if the president wants to retrieve the code with no good reason, it has to convince the others first.

There are multiple ways of achieving the sharing of a secret. The most naive way is by dividing the secret among n different parties. This way leaks information. A party receiving its share knows that the secret is at least bigger than its share. This reduces the brute-force space for an adversary significantly.

Another way is (t,n)-Shamir's secret sharing [55]. With (t,n)-Shamir's secret sharing, at least t out of n users are needed in order to recover the secret which is embedded in a polynomial. This polynomial has a degree of $t-1$. A polynomial with degree $t-1$ can be reconstructed from t distinct points of that polynomial. Any amount of points less than t is not sufficient to reconstruct the polynomial, since there is an infinite amount of polynomials with degree $t-1$ that go through these points.

To construct secret shares, the party which has the possession of the secret, creates the polynomial of degree $t-1$. Therefore, the polynomial is of the format

$$f(x) = \sum_{i=0}^{t-1} a_i \cdot x^i. \tag{2.2}$$

The coefficients a_i are chosen randomly. Except for a_0, this coefficient is set to the secret S. With this polynomial, shares of the secret are distributed to all n users. Each user i gets a random point of the polynomial $(x_i, f(x_i))$, where every user gets a distinct point. Note that the point $(0, f(0))$ is not assigned to a user since $f(0) = S$.

When all users retrieved their shares, t out of n users are able to recover the secret. The t users share their points of the polynomial. With the help of the Lagrange interpolation, as described in [69], the users reconstruct the polynomial f. Now the value $f(0)$ is computed by the users to retrieve the secret S.

The last secret sharing scheme we discuss, which is commonly used, is additive secret sharing. Where Shamir's secret sharing is (t,n)-secret sharing, additive secret sharing is (n,n)-secret sharing. In other words, with additive secret sharing, all users have to collaborate in order to retrieve the secret. Every user i gets a random share s_i from the party who possesses the secret. These shares are random numbers. Except for user 0, who gets $s_0 = S - \sum_{i=1}^{n-1} s_i$. When all users collaborate to retrieve the secret, the users sum all their shares resulting in

$$\sum_{i=0}^{n-1} s_i = S - \sum_{i=1}^{n-1} s_i + \sum_{i=1}^{n-1} s_i = S. \tag{2.3}$$

2.3. Homomorphic encryption

With a homomorphic cryptosystem, performing an operation on elements of the group of ciphertext messages corresponds to performing an operation on elements of the group of plaintext messages followed by an encryption. Specifically, this equals to

$$E(m_1) * E(m_2) = E(m_1 + m_2), \tag{2.4}$$

where E is the encryption function. Note that the operations $*$ and $+$ can be any operation. A cryptosystem is homomorphic with respect to the operations $*$ and $+$ if Equation (2.4) holds for any message m_1 and m_2.

One example of such a homomorphic cryptosystem is the Paillier cryptosystem [51]. We make use of this cryptosystem in our protocol discussed in Chapter 6. The Paillier cryptosystem is additively homomorphic and consists out of three phases, namely key generation, encryption and decryption.

The key generation is performed by the party who must be able to decrypt the ciphertexts. That party randomly chooses two large prime numbers p and q of the same size. Then, $n = pq$ is computed. The generating party also chooses a random $g \in \mathbb{Z}^*_{n^2}$. The values n and g form the public key. The private key contains two values, namely $\lambda = lcm(p-1, q-1)$ and $\mu = L(g^\lambda \bmod n^2)^{-1} \bmod n$. There are a few things to explain here. First, $lcm(x,y)$ computes the least common multiple of x and y. The value λ is private since other users do not know p and q and they cannot derive them from n, because factorization is assumed to be hard. Second, the function L is defined as $L(x) = \frac{x-1}{n}$. Finally, in order to compute μ, the modular inverse of $L(g^\lambda \bmod n^2)$ is computed modulo n. It is possible that this inverse does not exist. In this case, a new value is chosen for g after which λ and μ are computed again. The process continues until the modular inverse exists.

Once the keys are generated, a party encrypts a message $m \in \mathbb{Z}_n$ as

$$c = g^m \cdot n^r \bmod n^2, \tag{2.5}$$

where r is a random number from $\mathbb{Z}_n^*$. The random number has to be fresh for every ciphertext.

The last phase of decryption, is executed by the party who has the private keys. A ciphertext c is decrypted as

$$m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n. \tag{2.6}$$

The Paillier cryptosystem is additively homomorphic. When a scheme is additive homomorphic the operations in Equation (2.4) are multiplication and addition respectively. The Paillier cryptosystem is additive homomorphic since

$$E(m_1) \cdot E(m_2) = g^{m_1} \cdot r_1^n \cdot g^{m_2} \cdot r_2^n = g^{m_1+m_2} \cdot (r_1 \cdot r_2)^n = E(m_1 + m_2) \tag{2.7}$$

holds for any message m_1 and m_2.

The Paillier cryptosystem also allows the multiplication with a scalar by computing

$$E(m)^e = (g^m \cdot r^n)^e = g^{me} \cdot r^{ne}, \tag{2.8}$$

where e is some scalar. The result is still decryptable since r^{ne} can be rewritten to $(r^e)^n$ where $r^e \in \mathbb{Z}_n^*$ which is on its own a new random number.

In order to characterize the security of a cryptosystem, we use the terminology introduced by Smart [62]. A security game consists out of two parties, an adversary and a challenger. As the names suggest, the goal of the adversary is to win the security game and the challenger gives the adversary a challenge which need to be solved to win the game. When the best option for an adversary is to guess the output, the cryptosystem used in the game is said to be secure to this security game.

First, there is a difference between the one-way (OW) and the indistinguishability (IND) security games. An OW security game requires an adversary to return the secret message of a ciphertext chosen by the challenger. With an IND security game, the adversary sends two messages as challenge to an oracle. The oracle responds with the encryption of one of the messages. The task of an adversary in an IND security game is to return which of the two messages is encrypted. When a cryptosystem is IND secure, it is also semantic secure as proven in [62]. Semantic security means that an adversary with polynomial bounded computing power cannot learn any information from a ciphertext.

Both OW and IND security games have multiple levels of security. The lowest level of security is the passive attack (PASS). This level is the same as we described for the security games in the previous paragraph. The next level is the chosen-plaintext attack (CPA). Now, the adversary has to possibility to use an encryption oracle. This encryption oracle returns the ciphertext of any plaintext message of the adversary's choice. Any plaintext messages used for the challenge is not allowed to be encrypted by this oracle. Another level of security is CCA1. A security game in this level adds a decryption oracle. This oracle returns the plaintext message of a ciphertext of choice. In a CCA1 security game, the decryption oracle is only allowed to be executed before the challenge ciphertext is known. The last level is CCA2. This level has the same oracles as CCA1. Now, the decryption oracle is also allowed to be queried after the challenge ciphertext is known. Again, the challenge ciphertexts are not allowed to be decrypted by the decryption oracle.

Due to the malleability, the Paillier cryptosystem is not CCA2 secure. It is CPA and CCA1 secure for both the OW and IND security games assuming that the decisional composite residuosity assumption holds [4]. This assumption states it is hard to distinguish a random number from $r^n \bmod n^2$, where n is known and r is unknown.

2.4. Boneh-Boyen signature scheme

Another useful technique are signature schemes. Signature schemes are designed to prove the authenticity of a message. In other words, a signature schemes proves that a message belongs to the user it is supposed to come from. A signature scheme creates a signature which a user sends together with the ciphertext. The receiver verifies the signature. When the verification is successful, the ciphertext is sent by the correct user.

One signature scheme which we use in the protocol described in Chapter 6, is the Boneh-Boyen signature scheme [10] which nicely fits with the zero-knowledge proof in the verification part of our protocol described in Chapter 6. The signature scheme contains three phases, namely key generation, signing and verification.

In the key generation, two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are chosen. Both groups have a length of some prime number p. Next, a generator g_1 and g_2 for each group respectively are chosen. Now, the private key is any random element $x \in \mathbb{Z}_p^*$. The public key is formed by g_1, g_2 and $v = g_2^x$.

The party who has the private key signs a message $m \in \mathbb{Z}_p^*$ as

$$\sigma = g_1^{\frac{1}{x+m}}. \tag{2.9}$$

In the rare case of $x + m = 0$, the signature σ is set to 1.

Finally the receiving party receives a ciphertext and a signature. First the ciphertext is decrypted according to the cryptosystem that is used. The decryption results in a message m'. With the public key of the signature scheme, the message m' is correct when the equation

$$e(\sigma, v \cdot g_2^{m'}) = e(g_1, g_2) \tag{2.10}$$

holds. In the rare case that $\sigma = v \cdot g_2^{m'} = 1$, the signature is also valid. In all other cases, the signature is not valid and the received message is not sent by the correct user.

For signature schemes, it is important to be unforgeable. There are two types of unforgeability [62]. The first type is selective unforgeability. With a signature scheme of this type, an adversary does not have any advantage over guessing a valid signature of a challenge message. The second, and stronger, type is existential unforgeability. With a signature scheme of this type, an adversary does not have any advantage, over guessing, of producing a valid signature for any message the adversary chooses. The resources of the adversary can be extended in both types of unforgeability with a signing oracle. This is called the chosen message attack (CMA). The signing oracle computes the signature of any message of the adversary's choice. However, this message may not be equal to the challenge message or the message the adversary is going to output. Lastly, there are two types of CMA, namely the weak and the strong one. In the weak variant, the adversary is only able to call the signing oracle before the public key of the signature scheme is published. In the strong variant, on the other hand, the adversary is also allowed to call the signing oracle after the public key is published.

The signature scheme of Boneh-Boyen is existential unforgeable under a weak CMA assuming that the q-Strong Diffie-Hellman problem (q-SDH) assumption holds [10]. The q-SDH assumption states that it is hard to produce a pair $(y, g^{\frac{1}{x+y}})$, for any y, given $(g, g^x, g^{x^2}, \ldots, g^{x^q})$, where g is a generator of group $\mathbb{G}$.

2.5. Zero-knowledge proof systems

In this section, we first briefly introduce the topic of zero-knowledge proofs. After that we consider a specific proof which is used to prove plaintext equality.

2.5.1. A brief introduction to zero-knowledge proofs

A zero-knowledge proof is used by two parties, namely a prover and a verifier [31]. The prover needs to prove a certain statement about its private data to the verifier. However, the verifier may not gain any knowledge about the private data the user has except that the statements holds. Therefore, the proof has to be zero-knowledge. One way to create a zero-knowledge proof is to use a sigma protocol. A sigma protocol requires the prover to send a commitment to the verifier which replies with a challenge. The prover then sends a response which has incorporated the challenge. With the response, the verifier verifies whether the statement to be proven holds. The challenge is incorporated to prevent that a commitment and response are reused without having the correct data.

A zero-knowledge proof based on a sigma protocol can also be made non-interactive [15] with the help of the Fiat-Shamir heuristic [27]. A non-interactive proof does not require the verifier to send a challenge to the prover. Instead, the prover sends the commitment, the response and the output of a hash function all in once to the verifier. This hash function is computed over the input of the public parameters, the statement that needs to be proven and the commitment. The hash function replaces the challenge and prevents a prover to reuse a commitment and response without having the correct data.

In order to analyze the security of a zero-knowledge proof, three properties have to be proven.

1. **Completeness** ensures that a proof of a prover, with the knowledge of the data that satisfies the statement, is always correctly verified by the verifier.

2. **Soundness** ensures that a proof of a prover, with no knowledge of the data that satisfies the statement, is never correctly verified.

3. **Zero-knowledge** ensures that the zero-knowledge proof does not leak any information about the private data of the prover.

2.5.2. A non-interactive zero-knowledge proof of plaintext equality

As a part of the malicious user detection we use a non-interactive zero-knowledge proof in order to prove the plaintext equality of two ciphertexts that is presented by [22]. Assume we have two users i and j with Paillier public keys $pk_i = (n_i, g_i)$ and $pk_j = (n_j, g_j)$ respectively. One user generates two ciphertexts

$$c_i = g_i^m r_i^{n_i} \bmod n_i^2 \qquad c_j = g_j^{m'} r_j^{n_j} \bmod n_j^2, \tag{2.11}$$

where $r_i, r_j \in \mathbb{Z}_{n_i}^* \cap \mathbb{Z}_{n_j}^*$. Now, we want to prove that m is equal to m' without leaking any information about the message. The non-interactive zero-knowledge proof for this is given in Algorithm 2, where ℓ is the maximum amount of bits needed for either n_i or n_j and H is a secure cryptographic hash function.

The prover encrypts a random plaintext with the keys of both users. This random plaintext together with the hash of the resulting ciphertexts are used to mask the secret message. The prover finally computes two values v_i and v_j which are needed for the verifier to verify the plaintext equality. The prover outputs the masked secret message, the two ciphertexts of the random plaintext, v_i and v_j. The verifier, on its turn, computes the hash of the two public ciphertexts of the random plaintext. With this hash and the other public information, the verifier is able to verify the plaintext equality.

The completeness property holds since

$$\begin{aligned} u_i E_i(m)^e &= g_i^\rho s_i^{n_i} \cdot g_i^{m \cdot e} r_i^{n_i \cdot e} \\ &= g_i \rho + m \cdot e \cdot s_i^{n_i} r_i^{n_i \cdot e} \\ &= g_i^z \cdot v_i^{n_i}. \end{aligned}$$

By replacing the subscripts with j, the other check performed by the verifier is also proven to be correct. The completeness proof and the proofs of the two other properties, soundness and zero-knowledge, are given by Baudron et al. [6].

Algorithm 2: Non-interactive proof of plaintext equality

Prover:
Input: $\{m, r_i, r_j, g_i, g_j, n_i, n_j, H\}$
Choose a random $\rho \in [0, 2^\ell)$, $s_i \in \mathbb{Z}_{n_i}^*$ and $s_j \in \mathbb{Z}_{n_j}^*$
Compute $u_i = g_i^\rho s_i^{n_i} \bmod n_i^2$ and $u_j = g_j^\rho s_j^{n_j} \bmod n_j^2$
Compute $e = H(u_i, u_j)$
Compute $z = \rho + m \cdot e$
Compute $v_i = s_i r_i^e \bmod n_i$ and $v_j = s_j r_j^e \bmod n_j$
Output: $\{z, u_i, u_j, v_i, v_j\}$

Verifier:
Input: $\{E_i(m), E_j(m'), g_i, g_j, n_i, n_j, H, u_i, u_j, v_i, v_j\}$
Compute $e = H(u_i, u_j)$
Verify that $g_i^z v_i^{n_i} \bmod n_i^2 = u_i E_i(m)^e \bmod n_i^2$ and $g_j^z v_j^{n_j} \bmod n_j^2 = u_j E_j(m')^e \bmod n_j^2$

3

Related Work

The aim of this chapter is to explain the literature that relates to our research about privacy-preserving data aggregation schemes and the support of multi-functionality and malicious user detection. Section 3.1 discusses some state-of-the-art privacy-preserving data aggregation schemes. The second part focuses on privacy-preserving data aggregation schemes providing the support of a certain level of multi-functionality with each their advantages and disadvantages. The different levels of multi-functionality are described in more detail in the corresponding subsection. The last part discusses privacy-preserving data aggregation schemes which add the assumption of malicious user behaviour and how those schemes detect which user is acting maliciously.

During the discussions in this chapter, we assume the following network layout. There are u users or smart devices whose private data is aggregated by a party called the aggregator. The aggregator should never learn anything about the private data of an individual user. In some schemes, the aggregator is treated as the owner of the aggregated result. In those schemes, the aggregator is able to retrieve the aggregated result. In other schemes, the aggregator forwards the result of its operation to a control center. The control center is on its turn able to retrieve the aggregated result without gaining any knowledge about the data of individual users. Depending on the context, the owner of the aggregated result can share the aggregated result with others. Some schemes require an additional party, namely a trusted authority. The trusted party is used to perform trusted computations.

3.1. Privacy-preserving data aggregation

We start with some state-of-the-art privacy-preserving schemes which do not support the computation of any arbitrary function. Instead, they only support the computation of the sum. In addition, they assume all users to be honest-but-curious. Aspects of those schemes are used by many other privacy-preserving data aggregation schemes and therefore are important to take into account when designing a privacy-preserving data aggregation scheme.

Shi et al. The first scheme we discuss is from Shi et al. [57]. A trusted authority chooses a public random generator g of a cyclic group $\mathbb{G}$ with prime order p. The scheme makes use of additive secret sharing with the secret equal to zero as discussed in Section 2.2. The trusted authority initializes the system by giving each user i a secret share $s_i \in \mathbb{Z}_p$. The aggregator gets s_0 as its secret share for which holds that $s_0 + \sum_{i=0}^{u-1} s_i = 0$.

Each user i encrypts their message m_i by computing

$$c_i = g^{m_i} \cdot H(t)^{s_i}, \tag{3.1}$$

where H is a hash function and t is the current time point. After receiving all the ciphertexts from the users, the aggregator aggregates the ciphertexts as

$$c = H(t)^{s_0} \prod_{i=0}^{u-1} c_i = g^{\sum_{i=0}^{u-1} m_i}. \tag{3.2}$$

The sum of all the data is retrieved by computing the discrete log of c with base g.

The scheme also considers differential attacks, for example when the data of users is constant over time and a new user joins, the difference of the sum before and after the new user joined is the data of that new user. To deal with differential attacks, and therefore providing differential privacy [23], this scheme states that each user adds a random number as noise to its data. The random number is picked from a certain distribution which is described in the paper. Such a distribution has to make a trade-off between maximizing the differential privacy and minimizing the error of the aggregate. Finally, the sum of the data and the random number is encrypted and sent to the aggregator which decrypts the noisy sum.

This scheme, however, is not fault-tolerant since all secret shares has to be summed in order to result in zero. If one secret share is missing, summing the remaining secret shares results in a random number. Now, the aggregate message cannot be retrieved. Therefore, Chan et al. [16] present an extension to make the aggregation scheme fault-tolerant. The users are now organized in a tree structure. When one user is faulty, other branches of the tree can still be used for computing the aggregate. However, with only the knowledge of s_0, the aggregator is not able to decrypt the aggregate of the remaining users. The trusted authority therefore provides the aggregator with keys which can decrypt subsets of the tree. So the aggregator aggregates the ciphertexts of a subset of users of which it has a decryption key and is still able to retrieve the aggregate of the remaining users.

Garcia and Jacobs The privacy-preserving data aggregation scheme discussed by Garcia and Jacobs [29] removes the reliance on the trusted authority. Every user i splits its message x_i into random shares. One share x_{ij} for every other user j such that

$$x_i = \sum_{j=0}^{u-1} x_{ij}. \tag{3.3}$$

User i sends share x_{ij} to the aggregator encrypted under the public key of user j and keeps share x_{ii} to itself. After receiving all the shares, the aggregator relays the shares to the right users. Every user i then decrypts the shares and sum all the shares together resulting in

$$\sum_{j=0}^{u-1} x_{ji}. \tag{3.4}$$

This sum is sent to the aggregator without any encryption. The aggregator, finally, sums the responses of all users to retrieve the sum of the original data, i.e.

$$\sum_{i=0}^{u-1}\sum_{j=0}^{u-1} x_{ji} = \sum_{i=0}^{u-1}\sum_{j=0}^{u-1} x_{ij} = \sum_{i=0}^{u-1} x_i. \tag{3.5}$$

The work of Finster and Baumgart [28] does in essence the same as Garcia and Jacobs. Users in this protocol also create shares of the data and send them to other users. However, other steps are taken in order to make the scheme more robust. One such step is to split all the users in sub groups such that if one group fails, the others can continue with the protocol. Also the communication complexity is decreased since users only send shares to other users in the same group instead of all other users.

Erkin and Tsudik The scheme discussed by Erkin and Tsudik [25] also makes use of additive secret sharing. First the aggregator sets up the Paillier cryptosystem [51] with (n, g) as public parameters. Every user i secretly sends a random number r_{ij} to every other user j. Now, each user i encrypts its message m_i as

$$c_i = g^{m_i} \cdot H(p)^{n+\sum_{j=0,j\neq i}^{u-1} r_{ij} - \sum_{j=0,j\neq i}^{u-1} r_{ji}}, \tag{3.6}$$

where H is a secure hash function and p is the time point.

The aggregator then computes

$$\prod_{i=0}^{u-1} c_i = g^{\sum_{i=0}^{u-1} m_i} \cdot H(p)^{\sum_{i=0}^{u-1} n}. \tag{3.7}$$

Since the random shares cancel each other out, the result is decryptable following the Paillier cryptosystem.

In the paper, Erkin and Tsudik focus on the context of a smart grid. The protocol just described computes the spatial consumption, so the consumption of a set of users. Additionally, they also discuss a protocol to compute temporal and spatio-temporal. The temporal protocol computes the consumption of one user over a longer period. The spatio-temporal protocol combines the spatial and temporal protocol.

Kursawe et al. The last scheme with additive secret sharing we discuss in this section is from Kursawe et al. [40]. This scheme selects a subset P of the set of all users which are made leaders. Each user i then chooses a random number r_{ij} for each leader j. The random numbers are each encrypted under the public key of the corresponding leader and are sent to the aggregator. The aggregator relays the encrypted random numbers to the correct leader. A leader j computes its key r_j such that $r_j + \sum_{i \notin P} r_{ij} = 0$. Every other user i computes their key r_i such that $r_i = \sum_{j \in P} r_{ij}$.

A user i, which is either a leader or a non-leader, with message m_i adds its key r_i which forms the ciphertext, i.e. $c_i = m_i + r_i$. Since the sum of the keys of all leaders cancel out the sum of the keys of all other users, the aggregator sums all the ciphertexts which results in the sum of the data. As for all other schemes discussed so far, this scheme is not fault-tolerant since the share of every user must be included in order for the randomness to be removed.

Efthymiou and Kalogridis The last scheme of this section is completely different from the schemes discussed so far. Efthymiou and Kalogridis [24] do not focus on encrypting the data, but on making the data anonymous. In this scheme, a trusted authority creates anonymous certificates for all users. Those users first authenticate themselves to that trusted authority with their personal certificates. Once the users receive their anonymous certificates, they send data in combination with the anonymous certificate. The aggregator verifies these anonymous certificates without gaining any knowledge which user has sent it.

There are more schemes which have elaborated on this concept of anonymization, such as [9, 38, 46, 52]. Some remove the reliance on the trusted authority and others improve the efficiency of the authentication protocol.

3.2. Multi-functional privacy-preserving data aggregation

In this section we discuss the privacy-preserving data aggregation schemes which put their focus on providing a certain level of multi-functionality. The different levels of multi-functionality are additive aggregation, non-additive aggregation and arbitrary aggregation. The definition of each level is discussed in the corresponding subsection. The levels are discussed in order of increasing functionality.

3.2.1. Additive aggregation

The first level of multi-functionality we discuss is the additive aggregation. Additive aggregation only takes functions into account which rely on any form of addition. The addition is possibly done over weighted input. Examples of additive aggregation are the sum, the average and the variance. Additive aggregation does not give the support for any arbitrary function, but it does give more support than a privacy-preserving data aggregation scheme which only supports the summation.

Note that privacy-preserving data aggregation schemes, which only support the summation in literature like the ones discussed in Section 3.1, can be modified so that they also support other functions which are part of the additive aggregation level. In order to achieve this, users can encrypt a different value. Users can for example encrypt the square of their private data. Together with the sum of the original values, the sum of these squares is needed in order to compute the variance. Instead, in this section we discuss privacy-preserving data aggregation schemes which put their focus on providing additive multi-functionality and have the advantage that a user does not have to encrypt the result of a function, that can vary every round, performed on its value. Instead, a user sends the encryption of the value itself. Now, a user performs the same operation every round instead of depending on the function the aggregator or the control center wants to compute which on its turn requires an extra message to be sent to all the users to indicate which function needs to be computed.

Every privacy-preserving data aggregation scheme in this subsection has a different set of computations the aggregator has to perform for every function the aggregator or the control center wants to compute. For example, when the aggregator wants the average, it performs a different set of computations in comparison to computing the variance. This subsection, therefore, does not explain all the computations for every possible function in detail. Instead, we explain the detailed computations of one function which all the schemes have in common, namely the variance. We did not choose for the average, because the computations of this function resembles with the schemes with no multi-functionality.

Chen et al. The first privacy-preserving data aggregation scheme with additive aggregation is called MuDA by Chen et al. [18]. The trusted authority starts the protocol by setting up all the parameters. It creates a bilinear map tuple $(p, q, \mathbb{G}, \mathbb{G}_1, \epsilon)$ based on [12], the parameters for the Boneh-Goh-Nissim cryptosystem [11]

$(N, \mathbb{G}, \mathbb{G}_1, e, g, h)$ which is used for the encryption in the scheme and a secure cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}$.

Once everything is initialized by the trusted authority, each user i sends the ciphertext of its message $m_{i,t}$ for period t. First, the user computes $g_t = H(t)$. Then the user computes the ciphertext $c_{i,t}$ as

$$c_{i,t} = g_t^{m_{i,t}} \cdot h^{r_{i,t}}, \tag{3.8}$$

where $r_{i,t}$ is picked randomly from $\mathbb{Z}_N^*$. Once the encryption is completed, the user sends $c_{i,t}$ to the aggregator.

The next steps of the protocol depend on the function the control center wants to compute. There are three different functions considered in MuDA, namely average, variance and one-way ANOVA. For the variance, the aggregator sends two encrypted messages to the control center. The first is the encryption of $(\sum_i m_{i,t})^2$ which is retrieved by the computation

$$C_{t,1} = e(\prod_{i=0}^{u-1} c_{i,t}, \prod_{i=0}^{u-1} c_{i,t}), \tag{3.9}$$

where e denotes a bilinear pairing. The second is the encryption of $\sum_i m_{i,t}^2$ which is calculated as

$$C_{t,2} = \prod_{i=0}^{u-1} e(c_{i,t}, c_{i,t}). \tag{3.10}$$

The control center, on its turn, decrypts both values as is done in the Boneh-Goh-Nissim cryptosystem with its private key p. Raising $C_{t,1}$ or $C_{t,2}$ to the power p results in $(e(g,g)^p)^m$, where m is the underlying message of the ciphertext. Computing the discrete log with a base of $e(g,g)^p$ gives the underlying message m. The variance is then be calculated by

$$\sigma^2 = (1/u) \cdot D(C_{t,2}) - (1/u^2) \cdot D(C_{t,1}), \tag{3.11}$$

where the function D means the decryption of a ciphertext. For the correctness of these operations and the details of the other functions, we refer the reader to the original paper [18].

One disadvantage of MuDA is the reliance on a trusted authority which might not always be realistic in practice. Another disadvantage is that the control center is able to decrypt $c_{i,t}$ for any user i at round t. So when the control center intercepts the ciphertext transmitted to the aggregator, the control center gains the private information. The control center could also collude with the aggregator in order to gain possession of the ciphertext.

Ge et al. Another additive aggregation scheme we discuss is called FGDA by Ge et al. [30]. FGDA supports the average, variance and skewness functions and makes use of a trusted authority which bootstraps the system. First, the trusted authority chooses a large prime number p and a secure hash function H which are set as public parameters. Then, the trusted authority picks a random number x_i as private key for every user i. The control center gets x_0 as its private key for which should hold that

$$x_0 \prod_{i=0}^{u-1} x_i \mod p^4 = 1. \tag{3.12}$$

Now, the users are able to send a ciphertext of their data. First, each user i computes

$$h_i = x_i^{H(t)} \mod p^4, \tag{3.13}$$

where t is the time point. Next, each user i with message m_i computes its ciphertext as

$$c_i = (1 + m_i \cdot p) \cdot h_i \mod p^4. \tag{3.14}$$

Each user sends its ciphertext to the aggregator which relays the product C of all received ciphertexts to the control center.

The control center first computes $\sum_{i=0}^{u-1} m_i$ by computing

$$S_1 = \frac{(C \cdot x_0^{H(t)} \mod p^2) - 1}{p}. \tag{3.15}$$

Next, the control center needs to compute $\sum_{i=0}^{u-1} m_i^2$ which is done by computing

$$S_2 = S_1^2 - 2 \cdot \frac{(C \cdot x_0^{H(t)} \mod p^3) - 1 - p \cdot S_1}{p^2}. \tag{3.16}$$

Finally, the control center has the necessary data in order to compute the variance. Again, for the correctness of these operations, we refer the reader to the original paper [30].

One disadvantage of FGDA is that it relies on a trusted authority which, again, may not be realistic in practice. Another disadvantage is that it is not fault tolerant since x_i of every user i must be included in C in order to be decryptable.

3.2.2. Non-additive aggregation

All schemes of the previous subsection provide the support to compute a certain set of functions. However, in practice, one might want to compute other functions like minimum or maximum. Therefore, this subsection discusses some privacy-preserving data aggregation schemes which put their focus on the non-additive aggregation. As the name suggests, the non-additive level of multi-functionality includes all functions which do not belong to the additive aggregation discussed in the previous subsection. Examples of non-additive aggregation are the minimum, maximum and median. One other important function is the computation of histograms. If the bins of the histogram are set to exactly one value, every other arbitrary function can be computed from there on. The privacy-preserving data aggregation schemes discussed in this subsection do not focus on providing arbitrary functions. Therefore, they are discussed in this separate subsection within the level of non-additive aggregation.

Shi et al. PriSense, the scheme discussed by Shi et al. [58], makes use of the additive secret sharing. Each user i splits its message m_i into different slices such that all the slices sum up to m_i. User i sends to each other user j a share $m_{i,j}$ and keeps $m_{i,i}$ to itself. Every user i then computes the sum M_i of all the incoming shares $m_{j,i}$ and the share it kept to itself $m_{i,i}$. The result is sent to the aggregator. The aggregator retrieves the sum of all data (i.e. $\sum_{i=0}^{u-1} m_i$) by computing the sum of all M_i.

In order to achieve non-additive aggregation, PriSense is based on count "queries". Such a query sends a particular range to each user. The users then respond with an encrypted 1 if its value is in the range and an encrypted 0 otherwise. In other words, the message m_i of any user i is set to 0 or 1 depending on the requested range.

The paper discusses multiple non-additive functions, namely minimum, maximum, median, percentile and histogram. The first functions are the minimum and maximum. Since the protocol for those functions is intuitively the same, we only discuss the maximum function. Assume that the range of possible values goes from *min* to *max*. Now, *mid* is set to be the element which divides the entire range in two halves. First, the control center sends a count query for the range $[mid, max]$. The users respond with an encrypted 1 or 0 as explained before. The control center is able to get sum of those values. If the sum is at least 1, the maximum value is in the half from *mid* to *max*. Now we set *min* to *mid* and *mid* to be the element that divides the range between the new *min* and *max* into two halves. Otherwise, if the sum of the responses is 0, the maximum value is in the range $[min, mid]$. Now we set *max* to *mid* and *mid* to be the element that divides the range between *min* and the new *max*. The process continues until the range that is left is of size one. The resulting value is the maximum value.

For the median, it is necessary to think of *max* as a power of two. So assume that *max* is equal to 2^m. If the amount of users is not known, a count query is made for the entire range. First, the case of an uneven amount of users is considered. The position of the median, in an ordered list of elements, is equal to the amount of users plus one divided by two. The control center sends a count query for the range $[min, 2^{m-1}]$. If the sum of the responses is larger than the position of the median, it means that the median is lower. In this case, the next count query is for the range $[min, 2^{m-2}]$, If, on the other hand, the sum of the responses is lower than the position of the median, the median must be higher. The new range is $[min, 2^{m-1}2^{m-2}]$ in this case. The process continues until the sum of responses is exactly the position of the median. Now, the exact value for the median has to be found. We know that the highest value that falls in the resulting range is the median of the entire range, since the resulting range contains all positions up to and including the position of median. So performing the maximum protocol on this resulting range gives the median.

When the amount of users is even, the middle two elements have to be found. Instead of seeking for the value at the position equal to the amount of users plus one divided by two, we want to find the value at the

position equal to half the amount of users and at the position equal to half the amount of users plus one. The protocol described for the uneven case can be reused for these positions. Running the protocol with the two positions results in two values. The average of both values is the median in the case of an even amount of users.

The protocol for the median can be modified in such a way that it is also applicable for any percentile. Instead of using the position of half the amount of users as a condition, σ times the amount of users is used, where σ is the percentile which is divided by 100.

The last non-additive function discussed is the histogram. For a histogram you want to know the counts of each bin of the histogram. Therefore, computing a histogram is as simple as sending a count query for every such sub-range.

As for the additive aggregation part of PriSense, the non-additive aggregation is not fault-tolerant since each response relies on shares that are sent to other users. For the computation of any arbitrary function, the bins of the histogram need to be set to one. This, however, requires the aggregator to send a count query to every user for every value. Each user, therefore, also responds to a count query for every value. Above all, the response of a user is an encryption which shares the private data, so 0 or 1, with every other user in the same group. So, also this process is done for every value for every user.

Zhang et al. The scheme in [71] has two versions to aggregate data. One version is the same as the PriSense scheme which shares slices of the data to other users. The second relies on a shared secret k_i between each user i and the control center. Every round the control center sends a request to the users for sending their data. This request contains a random nonce r. Each user i with message m_i now sends

$$c_i = H_1(k_i||r) + m_i, \tag{3.17}$$

where H_1 is a secure hash function. The aggregator sums all ciphertexts and relays it to the control center. The control center then computes the sum $\sum_{i=0}^{u-1} m_i$ by computing

$$\sum_{i=0}^{u-1} m_i = \sum_{i=0}^{u-1} c_i - \sum_{i=0}^{u-1} H_1(k_i||r). \tag{3.18}$$

In order to achieve non-additive aggregation, this protocol uses the same structure with count queries as discussed with PriSense. The inefficiency of the communication complexity and the fault-intolerance of PriSense can be traded for giving the control center the ability to decrypt any individual ciphertext. Still, the aggregator sends a count query to every user for every value, but now the encryption of the response is more efficient.

Han et al. The scheme PPM-HDA from Han et al. [35] also makes use of the same count queries for non-additive functions. However, the implementation of the count queries is different. PPM-HDA makes use of a prefix membership verification in order to test whether a user has data in a specific range. First, the trusted authority initializes the system by creating a bilinear map tuple $(p, q, \mathbb{G}, \mathbb{G}_1, \epsilon)$ based on [12], the parameters for the Boneh-Goh-Nissim cryptosystem [11] $(N, \mathbb{G}, \mathbb{G}_1, \epsilon, g, h)$ which is used for the encryption in the scheme and a secure cryptographic hash function $H : \{0,1\}^* \rightarrow \mathbb{G}$.

After initialization, a user computes the prefix family F of its value, for example $F(12) = F(1100) = \{1100, 110*, 11**, 1***, ****\}$. Then, the user applies prefix numericalization Γ. Prefix numericalization has as input a prefix of the form $b_1 b_2 \dots b_k * \dots *$ and outputs $b_1 b_2 \dots b_k$ followed by a 1 and every $*$ is replaced by 0, for example, $\Gamma(F(12)) = \{11001, 11010, 11100, 11000, 10000\}$. Each user i with message m_i then sends the ciphertext

$$c_i = g^{\Gamma(F(m_i))} h^{H(t) \cdot r_i} \tag{3.19}$$

to the aggregator, where t is the time point and r_i is a random number picked by the user from $\mathbb{Z}_N$.

Upon receiving a ciphertext c_i from a user, the aggregator gets $(g^p)^{\Gamma(F(m_i))}$ by computing c_i^p. Assume that the aggregator wants to check that the value is in the range $[min, max]$, the aggregator first computes the minimum set of prefixes R of that range, for example $R([9,14]) = \{1001, 101*, 110*, 1110\}$. Now, the aggregator computes $(g^p)^{\Gamma(R([min,max]))}$. If the message m_i is a member of the range $[min, max]$, the intersection of $(g^p)^{\Gamma(F(m_i))}$ and $(g^p)^{\Gamma(R([min,max))}$ must not be empty. So, if the intersection is empty the comparison returns 0 and otherwise 1. When the aggregator does the comparison for all users, it sums up the outcomes which is the result of the count query. When the aggregator performs the same set of count queries as explained for PriSense, it is able to compute the non-additive functions such as minimum, median and histogram.

The advantage of PPM-HDA with respect to the previous schemes is that it requires less communication. Users send their data once and the aggregator performs all the count queries itself. However, the PPM-HDA relies on a trusted authority which may not be realistic in practice. Above all, the aggregator is able to determine the private data of a user in a binary search fashion. The aggregator divides the entire data range in two halves and checks if the value is in the lowest half. If it is, the aggregator divides the lowest half in two halves. Otherwise the highest half is divided in two halves. This process continues until the resulting range is of size one which is the private value.

3.2.3. Arbitrary aggregation

After explaining some schemes that focus on providing additive and/or non-additive aggregation, this subsection discusses the privacy-preserving data aggregation schemes that are on the level of arbitrary aggregation. The schemes on the arbitrary aggregation level support and focus on the calculation of every arbitrary function.

Chen et al. The first scheme we discuss is called RCDA by Chen et al. [17]. In RCDA, every user gets its own "bucket"of bits. Every bucket has the size of the amount of bits l needed for the maximum value that is possible. The position of a user's bucket determines the amount of zeros it should append. A user at position i appends $i \cdot l$ zeros. So, for example, the maximum value fits in 4 bits. The user with a bucket at position 0 just uses its value as message. The user with a bucket at position 1 appends its value with 4 zeros. The next appends 8 zeros, etc. The position a user gets is based on it ID. The exact details of assigning an ID is not explained in the paper.

The appended message is then encrypted under a scheme which satisfies the additive homomorphic property. We do not go into detail of the encryption scheme since any encryption scheme can be used as long as it has the additive homomorphic property. Now the aggregator computes the product of all the ciphertexts and relays it to the control center. The control centers then decrypts it. The data points are easily retrieved since they are in fixed buckets. So bits 0 to $l-1$ is a value, bits l to $2l-1$ is a value etc. From these values, any arbitrary function can be computed. Note that the message of the users is still private, since it is not known which value belongs to which user.

The disadvantage of this scheme is the needed amount of communication. In the worst case, a user sends l bits for every user. Another disadvantage is the unknown generation of an ID for a user.

Zhang et al. The scheme discussed in [72] does approximately the same as RCDA. Instead of using the ID for the order, it uses a private sequence number. The paper states two options for determining the sequence number. One is by communication between users, but it is not said how. The other option is by letting a trusted authority determine these sequence numbers. Another difference is that next to appending zeros for the buckets that come after it, this scheme also prepends zeros for the buckets that come before it. So every user sends $l \cdot u$ bits, where l is the amount of bits needed for the maximum value.

Gong et al. Also the scheme from Gong et al. [32] behaves in the same way. Every user appends and prepends the correct amount of zeroes. The difference is that this protocol picks private sequence numbers without reliance on a trusted authority. Every user i shares a random seed $r_{i,j}$ for a pseudo random number generator with every other user j. The seeds are shared through a Diffie-Hellman key exchange [21]. Every round t, a random number $r_{i,j,t}$ is retrieved from seed $r_{i,j}$. Additionally, the function $H_i(t)$ is defined as

$$H_i(t) = \sum_{k=0,k\neq i}^{u-1} r_{i,k,t} - \sum_{k=0,k\neq i}^{u-1} r_{k,i,t}. \tag{3.20}$$

The aggregator then sends a list of subintervals of the possible range of values a user can choose. Each user i picks a random number s_i and creates a vector V_i, where each subinterval is represented. A subinterval gets value 0 in the vector when s_i is no member of it. The subinterval which contains s_i gets value 1. The vector is encrypted by adding the value $H_i(j)$ to the value at every index j. The result is then sent to the aggregator.

The aggregator on its turn adds the vectors of all users together. Since the output of all H functions cancel out, the aggregator retrieves a vector of counts per subinterval. When a subinterval contains more than 1 user, it is split into subintervals again. For this new set of subintervals, the previous steps are repeated. This process continues until every subinterval only contains one user. At the end, the aggregator sends an ordered list of the subintervals with count 1 to the users. Each user now determines its private sequence number by getting the index of the subinterval its value is part of. Note that it is possible, with a very small probability, that two or

more users have picked the same value s_i. In this case the protocol is restarted and every user i picks a new value s_i.

When every user knows its private sequence number, the data is aggregated. Again a vector is created with n entries. Every entry gets the value $H_i(j+t)$ for user i at index j and round t. At the position in the vector that equals the private sequence number of a user, the data of that user is added. The resulting vector is sent to the aggregator which retrieves all values by summing all vectors. Since the private sequence numbers are private, the aggregator does not know which user has which value.

Also this scheme requires each user to send a message for every other user. In addition, this scheme is also not fault-tolerant since the output of $H_i(j)$ for every i needs to be included.

Bell et al. Another way of computing arbitrary functions is presented by Bell et al. [7]. For their scheme, they use invertible Bloom lookup tables [33]. In an invertible Bloom lookup table, users insert a key value pair. The value is set to the message they want to send. The key is an arbitrary string. The arbitrary string has to be big enough such that the probability of a collision with other users is minimal. The tables that are created locally are added together by the aggregator which results in a table which contains a union of all the elements. From this resulting table, the control center recovers the values of every user which can be used for the computation of any arbitrary function. Note that the control center does not know which user has sent which value. The recovery process is also a probabilistic one. So there are cases, with a very low probability, depending on the parameters, that the retrieval gives incorrect results. The messages that need to be sent by every user have a size of $2u \cdot \lceil \log_2(|P|) + \log_2(m) + \log_2(u) \rceil$, where m is the maximum value and P the maximum key used for inserting the value.

Wang et al., 2015 A different approach is described by Wang et al. [66]. The data in this scheme is sent in plain text. However, enough noise is added so that the exact value cannot be retrieved. The noise is picked in such a way that when the noised values are aggregated in a histogram, the noise cancels out as much as possible. But still, there is always noise left. So the result is not exact which might not be practical in every application. This idea is also known as differential privacy.

The scheme implements the noise generation with a bit flipping matrix. Every possible value has a corresponding bit value in a vector. The bit value corresponding to the value of the user has a probability greater than 0.5 to be 1. All the other values have a probability smaller than 0.5 to be 1. Each choice of probability is a trade-off between accuracy and privacy. When the chance is high that only the correct value is set to 1, the privacy is lost. When the chance is high that there are a lot of values set to 1, it impossible to retrieve the correct value.

The vectors of all the users are combined by the control center in such a way that they try to minimize the noise and turn it in a histogram. The histogram can then be used to compute any arbitrary function. However, since the histogram is noisy, so is the outcome of any function.

Wang et al., 2016 In a later work by Wang et al. [67], the concept is extended to weighted histograms which means that every bucket of the histogram has a specific weight attached to it. The focus here is to reserve more privacy guarantees for the high weighted values in comparison to the low weighted values. However, attention must be paid that the result does not leak information whether someones value is a high- or low weighted value.

In short, the algorithm they describe is as follows. If the value of a user is a high weighted one, it sets the bit value of a random high weighted value to 1. The real value has a different probability than the other values. There is also a probability that none of those items is set to 1. Next, each low weighted item has a probability to be set to 1. When the value of the user is a low weighted one, it also sets the bit value of a random high weighted value to 1. Now every value has the same probability, but again it is possible to set none of these items to 1. Then it sets the bit value corresponding to the real value to 1 with a certain probability and all the other low weight values with a different probability. For the characteristics and the proofs of this scheme we refer the reader to the paper [67]. Also this scheme does not produce exact results.

Viejo et al. Also the scheme discussed in Viejo et al. [65], supports the computation of any arbitrary function. The control center initializes the system by setting up the parameters (x, g, h, k, p, e) of the Okamoto-Uchiyama cryptosystem [49]. Those parameters are kept secret by the control center. The control center also creates a public one-way hash function H. Besides that, every user i shares a secret key K_i with the control center and receives $w_i = gh^{ID_i}$ from the control center.

At the start of a new round of aggregation, the control sends a random number v and an array I of random ciphertexts to each user. An element $I[s]$ contains $g^{\alpha_s}h^{\delta_s}$, where α_s and δ_s are random. Each user i now computes $I_i[s] = I[s]^{a_i} = g^{\alpha_s \cdot a_i}h^{\delta_s \cdot a_i}$ for every element s, where $a_i = H(v||K_i)$. For the element m, the user wants to send, it computes

$$I_i[m] = I_i[m] \cdot w_i = g^{\alpha_s \cdot a_i + 1}h^{\delta_s \cdot a_i + ID_i}. \tag{3.21}$$

The user finally sends the array I_i to the aggregator.

The aggregator computes per element the product of all received arrays, i.e. $\gamma_s = \prod_{i=0}^{u-1} I_i[s]$ for every s, which is relayed to the control center. By decrypting γ_i according to the Okamoto-Uchiyama cryptosystem, the control center receives $\alpha_s \cdot \sum_{i=0}^{u-1} a_i + \sum_{i \in S} 1$, where S is the set of users that have value s. The control center knows each a_i and therefore computes $\sum_{i \in S} 1$ which is the amount of users that have sent this value s. If this amount of users is computed for every element, any arbitrary function can be computed over the data of the users.

This scheme requires every user to send a ciphertext for every possible. Above all, the control center is able to decrypt an individual ciphertext which is retrieved by either intercepting the ciphertext in transmission to the aggregator or by colluding with the aggregator.

Bianchi et al. The last scheme we discuss in this section is a scheme from Bianchi et al. [8]. This scheme is not focused on the multi-functionality, but on packing different signals together in one encryption and how to unpack them also in their encrypted form. This scheme can also be used in the context of multi-functional privacy-preserving data aggregation which makes it interesting for us. The scheme uses super-increasing sequences. Coefficients are created which satisfy the conditions of a super-increasing sequence. However, there is one difference. It should hold for a coefficient α_i that

$$\alpha_i > \sum_{j=0}^{i-1} \alpha_j \cdot \Delta d, \tag{3.22}$$

where Δd is the maximum signal a user can send. Each user gets one such coefficient and multiplies its data with that coefficient. This encoding is then encrypted. The ciphertexts of multiple users can be added together due to the additive homomorphic property which the encryption scheme must satisfy.

In this paper, there are scenarios where you want to unpack the values again, but without decrypting it since each user has its own coefficient. When the aggregated ciphertext is decrypted, the value of each user can be retrieved. The operations needed for this, which are stated as follows, are translated to the ciphertext domain. First the summed values are taken modulo the highest coefficient. Take the difference of the value before and after the modulo operation and divide by the highest coefficient. The result is the value of the user with the highest coefficient, but since these operations are done in the ciphertext domain this value is also encrypted. The process is repeated for the second highest coefficient with as input the resulting aggregate that is left after the modulo operation of the highest coefficient. The entire process is repeated for all coefficients from high to low.

3.3. Malicious user privacy-preserving data aggregation

Users of a privacy-preserving data aggregation scheme can behave maliciously. In this section, we discuss the schemes which assume that a user can be malicious and which detects those malicious users.

Viejo et al. The first scheme with malicious user detection that we discuss is a scheme of Viejo et al. [65]. We already discussed this scheme in the previous section since it also provides the support for the computation of any arbitrary function.

There are three checks for the control center to perform to check whether a user is behaving maliciously. First, every user i included its ID in the element m of array I_i which corresponds to the user its value. Multiplying the ciphertexts of all elements, therefore gives a result in the following format

$$g^m h^{\sum_{i \in M} ID_i}, \tag{3.23}$$

for some m where M is the set of all user that have value m. The control center computes whether all ID's have been included in some element. If not, a user is behaving maliciously. Another check the control center performs is to check the count of each element. The counts per element should not exceed the amount of

users participating in the system. The last check is to sum up those counts per element which should add up to the amount of users.

With these three checks, the control center moves down the tree of users to detect which user, or users, is behaving maliciously and possibly remove that user, or users. The aggregator in this scheme actually consists out of a set of intermediary nodes. All users are leafs of a tree and the control center is the root. In between the users and the control center are all intermediary nodes. Those nodes aggregate the ciphertexts of their children and relay it to their parent. With the malicious user check, the control center checks per sub-tree whether the aggregate is valid until it reaches the leaves and detects the malicious user.

Again, the main disadvantage of this scheme is that control center is able to decrypt an individual ciphertext. With the malicious user detection, the control center has the autonomy to check the individual ciphertexts and therefore does not need to collude with an intermediary node or to intercept a ciphertext.

Sun et al. Another scheme which supports malicious user detection is APED [63]. In this scheme, users are organised in pairs. For each pair of two users i and j, the trusted authority generates a private key k_i for user i and k_j for user j. The trusted authority then sends to the control center a key k_{ij} for each pair such that $k_i + k_j + k_{ij} = 0 \mod p$, where p is a random large prime number. The control center computes K which is the sum of k_{ij} of every pair.

In first instance, the control center tries to decrypt the aggregate of all users using the key K. When this is successfully, there is no malicious user. Otherwise, there is, and the control center has to find this malicious user(s). The control center decrypts the aggregate of every pair with the corresponding key k_{ij}. In this way, the control center finds the pairs which contains at least one malicious user. The next round a different pairing is applied. By intersecting the malicious pairs, the malicious users are found. Note that it is assumed the a malicious user behaves maliciously every round.

Shi et al. With a different encryption scheme, for efficiency reasons, the scheme described in [59] has the same error detection protocol as APED [63]. One difference is that APED is described to work on pairs where the scheme of Shi et al. is described in such a way that it can be extended to bigger groups.

Dimitriou and Awad Where the previous schemes detect malicious users after checking the aggregate, the scheme discussed in [22] checks every individual ciphertext beforehand without losing privacy. In this scheme every user i secretly sends a random number r_{ij} to every other user j. During the encryption of message m_i of user i, the outgoing random numbers are added and the incoming random shares are subtracted, i.e.

$$c_i = m_i + \sum_{j=0, j\neq i}^{u-1} r_{ij} - \sum_{j=0, j\neq i}^{u-1} r_{ji}. \tag{3.24}$$

When the aggregator sums all the ciphertexts, the random numbers cancel each other out and the sum of the messages remain.

However, when a user maliciously sends the wrong random numbers, the result cannot be retrieved anymore. To prevent this, some checks are added in the protocol. Each user i now publishes a ciphertext of its message encrypted under its own public key, i.e. $E_i(m_i)$. Every user i sends $E_i(r_{ij})$ and $E_j(r_{ij})$ for every other user j to the aggregator. With the help of a non-interactive zero knowledge proof of plaintext equality as described in 2.5, the aggregator verifies that both ciphertexts have the same plaintext. The aggregator relays $E_j(r_{ij})$ to user j who decrypts it to get r_{ij}. Once user i received the random shares from all the other users it computes

$$E_i(c_i) = E_i(m)\frac{E_i(r_{ij})}{E_i(r_{ji})}, \tag{3.25}$$

where every encryption used is the ciphertext that is published beforehand so that the aggregator can compute $E_i(c_i)$ in exactly the same way. Each user i also encrypts c_i under the public key of the aggregator, i.e. $E_A(c_i)$, where A is the aggregator. Both $E_i(c_i)$ and $E_A(c_i)$ are sent to the aggregator together with a proof of plaintext equality. The aggregator checks whether the version of $E_i(c_i)$ which the user sent is the same as the version the aggregator computed. Again with the help of the proof of plaintext equality, the aggregator checks whether c_i is the same in both ciphertexts. Finally, the aggregator decrypts $E_A(c_i)$ with which, the aggregator computes the aggregate as before.

With all those checks, a malicious user is forced to follow the protocol, otherwise it is detected by one of the checks. However, a disadvantage of this scheme is that is does not support the computation of any arbitrary function.

4

Encoding for Multi-Functionality

In this chapter, we describe two encoding schemes for providing the multi-functionality. Note that these encoding schemes are not privacy-preserving on their own. They are used in the privacy-preserving data aggregation schemes presented in Chapters 5 and 6. The first encoding scheme has maximum certainty which means that the output after decoding is always correct. However, the range of values that are supported in this encoding scheme are limited. The second encoding scheme therefore extends the range at the cost of having some probability that the output after decoding is incorrect.

4.1. Encoding scheme with maximum certainty

The concept of a super increasing sequence is used for the encoding scheme to give the aggregation scheme its multi-functionality. In a nutshell, each user sends a coefficient corresponding to their value to the aggregator. The amount of users sending a particular coefficient can be retrieved from the sum of all those coefficients. This results in a histogram of the values, giving the aggregator the opportunity to perform any arbitrary function. In this section we give a detailed overview of the encoding scheme with maximum certainty. Also, we provide a proof of the correctness of the encoding scheme and show what range of values it supports.

4.1.1. Initialization

As was said above, the users send a coefficient corresponding to their values. Those coefficients, however, cannot be any random values. If the coefficients are chosen incorrectly, the aggregate does not have a unique solution and the sum of coefficients cannot be decoded. For determining the coefficients, we use the idea behind the super increasing sequences. Modifications are made to fit it in our context of an aggregation scheme. The coefficients have to satisfy three conditions. The first condition is related to the lowest coefficient. The definition of a super increasing sequence states that every number must be positive. Therefore also every coefficient must be positive which leads to condition 1:

$$\alpha_0 > 0, \tag{4.1}$$

where the character α is used for coefficients and the subscript 0 indicates that it is the first and lowest coefficient.

It is possible that every user sends the same coefficient, since every user can have the same value. The coefficients must take this into account by providing enough space between consecutive coefficients such that the histogram can be uniquely determined. In our protocol, the users can only send one coefficient. So there is no need to keep space for the possibility of each user sending multiple the same coefficients. Providing the space between consecutive coefficients results in condition 2:

$$\alpha_i > \alpha_{i-1} \cdot u, \tag{4.2}$$

where u is the total amount of users participating in the protocol. The second condition relates to Equation 2.1 of the super-increasing sequence which reserved space for any combination of previous numbers.

Finally, we deal with the message space. With only the previous conditions, coefficients can be created up to infinity. However, it has to fit in a message which is going to be sent to the aggregator, i.e the highest

coefficient has to have enough space to be sent by all users. Condition 3 is defined as

$$n > \alpha_m \cdot u, \tag{4.3}$$

where n is the size of the message space and the subscript m indicates the total amount of possible values minus one, since we start counting at zero, and therefore also the amount of coefficients that can be sent by the users minus one.

The aggregator provides a list of all possible values $V = \{v_0, v_1, \ldots, v_m\}$ and a list of all possible coefficients $A = \{\alpha_0, \alpha_1, \ldots, \alpha_m\}$ to every user, such that coefficient α_i corresponds to value v_i for every i.

4.1.2. Encoding

After setting up all the coefficients, the users send the coefficient corresponding to their value. Each value therefore is mapped to a coefficient. A value can be anything like a string, a character or a number. Upfront, it has to be known what the possible values are. In the case of numbers the possible values can be a range. This range can be either continuous, where every number in the range is possible, or non-continuous, where not every number in the range is possible. All the possible values are received from the aggregator in list V. The correct coefficient from A is found at the same index as the value in V.

When a user's value is mapped to a coefficient, it sends its coefficient to the aggregator which sums up all the coefficients received resulting in X_m.

4.1.3. Decoding

Finally, we can decode the computed sum of the coefficients X_m and receive the histogram of the values of all users. Note that the summation now seems useless since you can map the coefficients received from the users individually to a value and add it to the histogram. However, when making the protocol privacy-preserving, the coefficient send by a user is not known to the aggregator. The only things the aggregator knows are the sum of the coefficients that are sent and which coefficients are possible for a user to send.

It is useful to see X_m not just as one number but in the following representation:

$$X_m = u_0 \cdot \alpha_0 + u_1 \cdot \alpha_1 + \ldots + u_m \cdot \alpha_m, \tag{4.4}$$

where u_i means the number of users that have sent coefficient α_i. The first thing the aggregator calculates is u_m. It first checks how many times α_m fits in X_m. The result is u_m. The product of α_m and u_m is subtracted from X_m resulting in X_{m-1}. With X_{m-1}, the aggregator can repeat the process for retrieving u_{m-1}. The whole algorithm is formally summarized in Algorithm 3. Note that X_{-1} is defined to be 0 when a valid encoding is used.

Algorithm 3: Decoding

Data: X_m
Result: $[u_0, u_1, \ldots, u_m]$
for $i \leftarrow m$ **to** 0 **do**
  $X_{i-1} = X_i \bmod \alpha_i$;
  $u_i = \frac{X_i - X_{i-1}}{\alpha_i}$;

The resulting histogram of coefficients now are mapped to the histogram of values by mapping the coefficients back to the corresponding values. The mapping is done in the same was as done in Subsection 4.1.2, but in the reversed way. So a coefficient at index i in list A maps to the value at index i in list V.

Finally, with the histogram of values, the aggregator can perform any arbitrary function on the data such as sum, minimum, maximum and any polynomial function.

4.1.4. Correctness

In order to prove the correctness, we have to prove that $\frac{X_j - X_{j-1}}{\alpha_j} = u_j$ holds for any j. The correctness of the encoding scheme with maximum certainty follows from

$$X_m = u_0 \cdot \alpha_0 + u_1 \cdot \alpha_1 + \ldots + u_m \cdot \alpha_m. \tag{4.5}$$

Since users may only send one coefficient, we have $\sum_{i=0}^{j} u_i \leq u$ for any j and therefore

$$u_0 \cdot \alpha_0 + u_1 \cdot \alpha_1 + \ldots + u_{m-1} \cdot \alpha_{m-1} \leq u \cdot \alpha_{m-1}. \tag{4.6}$$

By definition, this is strictly smaller than α_m. Therefore

$$X_{m-1} = X_m \mod \alpha_m = u_0 \cdot \alpha_0 + u_1 \cdot \alpha_1 + \ldots + u_{m-1} \cdot \alpha_{m-1}, \tag{4.7}$$

$$\frac{X_m - X_{m-1}}{\alpha_m} = \frac{u_m \cdot \alpha_m}{\alpha_m} = u_m. \tag{4.8}$$

With the same reasoning we can prove $\frac{X_j - X_{j-1}}{\alpha_j} = u_j$ for any $j \in [0, 1, \ldots, m-1]$.

4.1.5. Space complexity

As already said, the coefficients have to fit within the message space according to condition three defined by Equation 4.3. The question of this subsection is how many coefficients, and therefore values, can be used in this encoding scheme. For retrieving the amount of coefficients that fit in the message space, the conditions of the coefficients are combined. First we can rewrite condition one defined in Equation 4.1 as

$$\alpha_0 \geq 1. \tag{4.9}$$

Using condition two defined in Equation 4.2, we get

$$\alpha_1 \geq \alpha_0 \cdot u + 1 \geq u + 1 \tag{4.10}$$

and

$$\alpha_2 \geq \alpha_1 \cdot u + 1 \geq (u+1) \cdot u + 1 = u^2 + u + 1. \tag{4.11}$$

This continues until α_m

$$\alpha_m \geq \alpha_{m-1} \cdot u + 1 = \sum_{i=0}^{m-1} u^i. \tag{4.12}$$

Finally we can use condition three as

$$n \geq \alpha_m \cdot u + 1 = \sum_{i=0}^{m} u^i. \tag{4.13}$$

These calculations show that the amount of values possible in the protocol depends on the number of users in the protocol. The more users, the less values are possible and vice versa. The dependence between the amount of users and values is illustrated in Figure 4.1. For this figure we used a message space of 2048 bits since that is the standard for a modest security level for the Paillier cryptosystem which we use in the privacy-preserving data aggregation scheme presented in Chapter 5

Figure 4.1 also shows that the amount of possible values decreases more slowly when the amount of users gets higher. Even with 10 million users, each user is still able to choose a value from a set of 88 values.

In some applications, the amount of values supported by the encoding scheme for a message size of 2048 bits is not enough. One option is to extend the message size. The other option is discussed in Section 4.2. In order to analyze this first option of extending the message size, we compute the dependence between the message size and the amount of values used in the encoding scheme and the amount of users participating in the protocol. In Equation 4.13 we stated an upper bound for the coefficients expressed in the amount of values and users. This equation can be interpreted in such a way that the message size should be at least as big as $\log_2(\sum_{i=0}^{m} u^i)$, where the log operation is needed to transform the message size to bits. In other words, the message size in bits is $O(\log_2(u^m))$.

4.2. Encoding scheme with extended range

The aim of this section is to present an encoding scheme which extends the range of possible values compared to the encoding scheme with maximum certainty, without increasing the size of the messages that are sent by the users.

The problem with the encoding scheme with maximum certainty is that every coefficient has to have enough space such that all users can send it in the same round. This was formulated as condition two in Equation (4.2) in Section 4.1. Most of the space is never used since it might be unlikely that all users have the

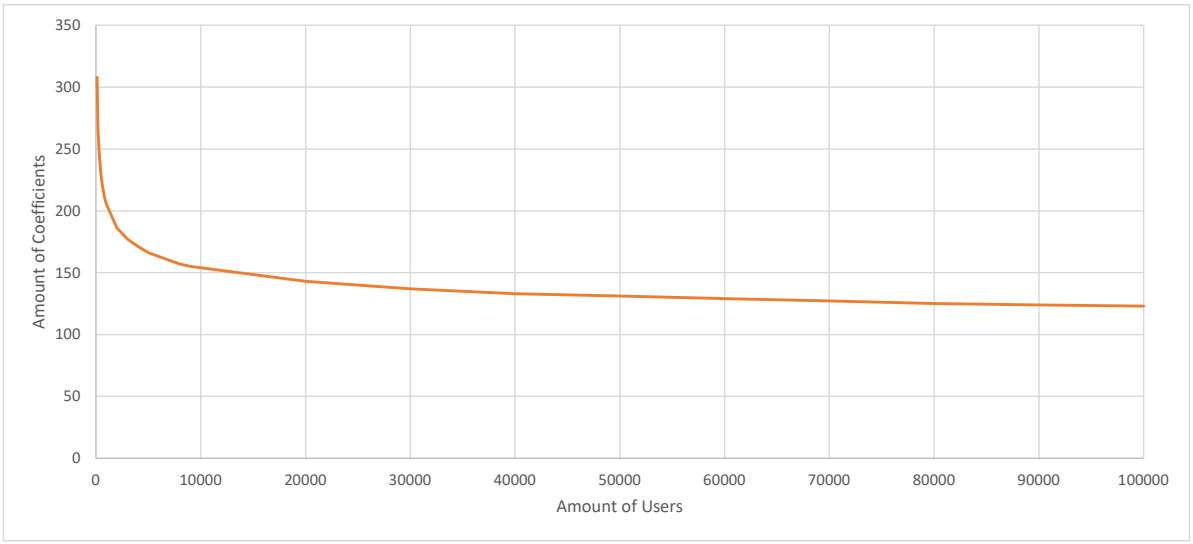

Figure 4.1: This figure illustrates the relation between the amount of users participating in the protocol and the amount of values that fit in a message space of 2048 bits for the encoding scheme with maximum certainty.

same value. Also, some values are sent more often than others. If the probability distribution of the values is known beforehand, it is possible to determine how likely it is that a certain amount of users send a certain coefficient. With the help of these probabilities, you can decrease the space of a coefficient from all users to an amount which is smaller, but still gives you a certain probability that it is big enough. This section goes through two probability distributions to illustrate the details, namely a uniform distribution and a normal distribution. After those details, some key points are summarized to take into account when dealing with any arbitrary distribution. We conclude this section by providing a proof of correctness and a comparison of the supported range of values.

4.2.1. Uniform distribution

In a uniform distribution every coefficient has the same probability to be used by a user. Therefore the probability for a coefficient α_x to be used by a user is set to

$$p_x = 1/m. \tag{4.14}$$

The probability for exactly i users using coefficient α_x is therefore

$$p_{i,x} = \binom{n}{i} \cdot p_x^i \cdot (1-p_x)^{n-i}. \tag{4.15}$$

If the amount of users expected for a coefficient is set to a specific value i, it is needed to compute the probability that i users is not enough. This probability comes down to the probability that at least $i+1$ users use this coefficient,

$$p'_{i+1,x} = \sum_{j=i+1}^{n} p_{j,x}. \tag{4.16}$$

The goal is to find a value i which is as small as possible, but for which $p'_{i+1,x}$ is smaller than a pre-determined threshold T. Let U_x denote the value i which achieves this for coefficient α_x. It means the amount of users that are taken into account for that coefficient.

When rewriting the conditions for the coefficients for this encoding scheme, the u in the second condition in Section 4.1, however, cannot be replaced by the U_x value of the corresponding coefficient. In the encoding scheme with maximum certainty, it was not possible that multiple coefficients were sent by all u users. At most one coefficient could be sent by u users. Now, it is possible that multiple coefficients are sent by the amount of users U_x that are taken into account for those coefficients. So instead of taking only the previous coefficient into account, now all previous coefficients are taken into account. The renewed version of condition two is

now stated as,

$$\alpha_i > \sum_{j=0}^{i-1} \alpha_j \cdot U_j. \tag{4.17}$$

The same also holds for the third condition which states that

$$n > \sum_{i=0}^{m-1} \alpha_i \cdot U_i. \tag{4.18}$$

After determining the coefficients, the encoding and decoding is executed as discussed in Section 4.1.

4.2.2. Normal distribution

In a normal distribution, not every coefficient has the same probability as was the case with a uniform distribution. The normal distribution therefore requires some extra attention when defining the values p_x.

First of all, the normal distribution is continuous and the coefficients are discrete. Therefore, when computing the probability of a coefficient, a range on the continuous spectrum is assigned to that coefficient. A coefficient α_x gets the range from $x-0.5$ to $x+0.5$. The probability of coefficient α_x occurring is equal to the area under the curve within that range which can be calculated by

$$P(X \leq x+0.5) - P(X \leq x-0.5), \tag{4.19}$$

where X is a random variable representing the coefficient chosen by a user. However, these probabilities are not same for every normal distribution. It differs due to different standard deviations. To incorporate a standard deviation σ, there are so called z-scores. The z-scores for a coefficient α_x are:

$$z_1 = (x-0.5)/\sigma, \quad z_2 = (x+0.5)/\sigma. \tag{4.20}$$

These z-scores are incorporated into Equation 4.19, resulting in

$$p_x = P(X \leq z_2) - P(X \leq z_1). \tag{4.21}$$

These probabilities over z-scores are the same for every normal distribution and can be looked up in a z-score table. The result is the probability we are looking for, namely p_x.

The computation of values $p_{i,x}$ and $p'_{i,x}$ are the same as for the uniform distribution. However, when determining the value of U_x, special attention has to be paid to coefficients that are far from the mean of the normal distribution. At some point the probability that at least 1 user uses a coefficient is lower than the threshold. According to the protocol, U_x is set to zero. This comes down to no coefficient at all for that value. Since we want to extend the range, ignoring coefficients does not contribute. Users do not have the possibility to send any of those coefficients and therefore the range is not be extended. Therefore, U_x of those coefficients are set to be one. Now every user has the opportunity to send this coefficient and therefore the range is extended. Note that the probability for those coefficients to have too less space is much smaller than the pre-determined threshold.

When the value U_x is determined for every coefficient α_x, the coefficients are determined with the same conditions as for the uniform distribution. With the correct coefficients, the encoding and decoding is again executed as discussed in Section 4.1.

4.2.3. Arbitrary distribution

The last two subsections explained the encoding scheme with extended range using two frequently occurring distributions. However, you could use any distribution. You can assign random probabilities to each coefficient under the condition that they all have to sum up to one. From these probabilities you can again compute the U_x values, so for how many users you reserve space for coefficient α_x. Finally, when knowing the U_x values, you can compute the coefficients with the conditions stated for the uniform distribution and execute the encoding and decoding as described in Section 4.1.

There is one key thing to take into account which is also discussed in the case of the normal distribution. All coefficients α_x should at least have a U_x value of 1, otherwise the users do not have the possibility to use the coefficient and the range is not extended.

4.2.4. Correctness

The decoding for the encoding scheme with extended range is equal to the decoding of the encoding scheme with maximum certainty. Therefore, in order to proof the correctness we again have to prove that $\frac{X_j - X_{j-1}}{\alpha_j} = u_j$ holds for any j. The correctness of the encoding scheme with extended range follows from

$$X_m = u_0 \cdot \alpha_0 + u_1 \cdot \alpha_1 + \ldots + u_m \cdot \alpha_m. \tag{4.22}$$

Since $u_j \leq U_j$ holds with probability $1 - T$, for any j, we have

$$u_0 \cdot \alpha_0 + u_1 \cdot \alpha_1 + \ldots + u_{m-1} \cdot \alpha_{m-1} \leq U_0 \cdot \alpha_0 + U_1 \cdot \alpha_1 + \ldots + U_{m-1} \cdot \alpha_{m-1}$$

By definition, this is strictly smaller than α_m. Therefore

$$X_{m-1} = X_m \mod \alpha_m = u_0 \cdot \alpha_0 + u_1 \cdot \alpha_1 + \ldots + u_{m-1} \cdot \alpha_{m-1}, \tag{4.23}$$

$$\frac{X_m - X_{m-1}}{\alpha_m} = \frac{u_m \cdot \alpha_m}{\alpha_m} = u_m. \tag{4.24}$$

Again, with the same reasoning we can prove $\frac{X_j - X_{j-1}}{\alpha_j} = u_j$ for any $j \in [0, 1, \ldots, m-1]$. Note that with a probability of at most T the encoding scheme with extended range does not produce correct results.

4.2.5. Space complexity

If we choose every coefficient to be the smallest as possible, as we did for the encoding scheme with maximum certainty, we get

$$\alpha_i = \sum_{j=0}^{i-1} \alpha_j \cdot U_j + 1, \tag{4.25}$$

for every i, for the encoding scheme with extended range. Now, the maximum amount of coefficients is the biggest m for which holds that

$$2^{2048} \geq \sum_{j=0}^{m} \alpha_j \cdot U_j. \tag{4.26}$$

Again we use a message space of 2048 bits since it is the standard for a modest level of security for the Paillier cryptosystem which we use in Chapter 5.

Figure 4.2 shows the causation between the amount of coefficients and the amount of users for both encoding schemes. For the encoding scheme with extended range, different probability distributions and different thresholds T are used.

From Figure 4.2 we derive that the encoding scheme with extended range indeed extends the range of values a user can send. We also see that a normal distribution with a low standard deviation accepts a larger range than a uniform distribution or a normal distribution with a high standard deviation.

When the threshold T is increased, coefficients can be lower, since the probability to exceed the expected amount of users is allowed to be higher. Therefore, the amount of coefficients also increases. If, on the other hand, T decreases, the amount of coefficients also decreases. This dependence on T is illustrated in the graph. We also see that the difference in the amount of coefficients between different thresholds become smaller when the standard deviation decreases.

4.3. Computation complexity

The computation complexity of both encoding schemes are summarized in Table 4.1. The initialization of both encoding schemes, performed by the aggregator, is only executed once and computes the coefficients. In the encoding scheme with maximum certainty this requires $O(m)$ additions and multiplications. Every coefficient is computed as $\alpha_{i-1} \cdot u + 1$.

For the encoding scheme with extended range, the expected amount of users U_x needs to be computed for every coefficient α_x. In order to find this amount, the minimal amount of users should be found such that Equation 4.16 is smaller than T which relies on Equation 4.15. Once we computed a probability $p_{i,x}$ for some i, it can be reused later on for the same coefficient α_x. Therefore we compute Equation 4.15 at most $O(u)$ times for every coefficient. One computation of Equation 4.15 requires $O(u)$ multiplications and $O(1)$ exponentiations, subtractions and divisions. Therefore, for all coefficients, Equation 4.16 requires $O(m \cdot u^2)$

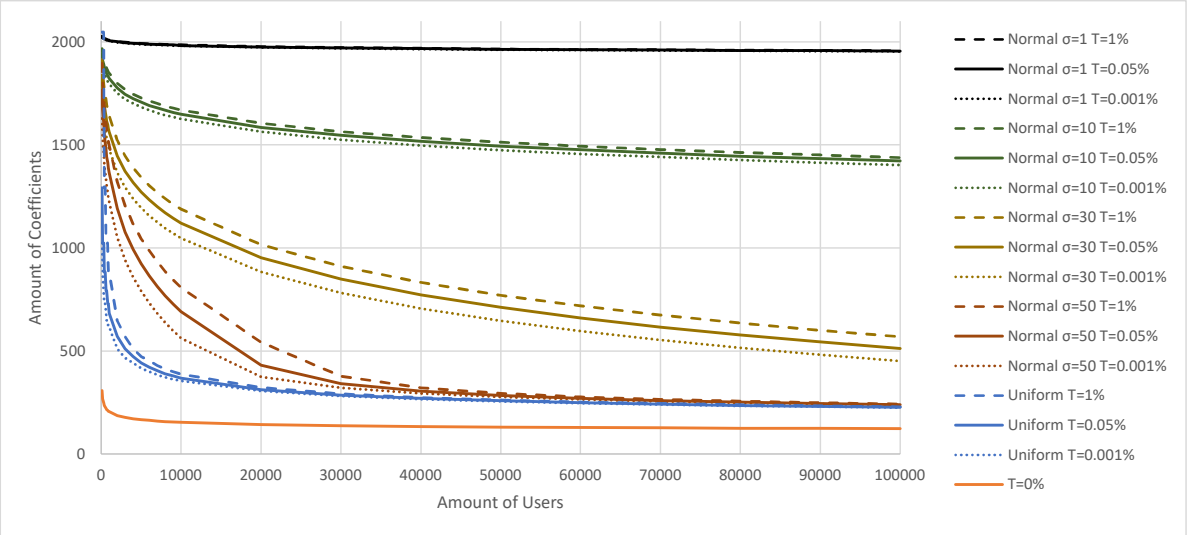

Figure 4.2: This figure illustrates how many coefficients are supported in a 2048 bit message for a certain amount of users participating in the protocol. The encoding scheme with maximum certainty corresponds to a threshold T of 0%. For the encoding scheme with extended range, we took a uniform distribution and several normal distributions as examples, each with different thresholds T. Note that the mean of the normal distributions does not matter in our context.

Table 4.1: Computation complexity of the encoding scheme with maximum certainty and with extended range, where m is the amount of values and u the amount of users.

	Initialization		
	Encoding with maximum certainty	Encoding with extended range	Decoding
Addition	$O(m)$	$O(m \cdot u)$	-
Multiplication	$O(m)$	$O(m + u^2)$	-
Subtraction	-	$O(m \cdot u)$	$O(m)$
Exponentiation	-	$O(m \cdot u)$	-
Division	-	$O(u)$	$O(m)$
Modulo	-	-	$O(m)$

multiplications, $O(m \cdot u)$ exponentiations, subtractions, divisions and additions. Since $\binom{n}{i}$ for some i is the same for every coefficient, it only needs to be computed once. This results in a final computation complexity of $O(m + u^2)$ multiplications, $O(u)$ divisions and $O(m \cdot u)$ subtractions, exponentiations and additions.

After initialization, the user performs encodes some value v_i resulting in the corresponding coefficient α_i. The aggregator decodes the sum of received coefficients as described in Algorithm 3. This decoding requires $O(m)$ subtractions, divisions and modulos.

5

Multi-Functional Privacy-Preserving Data Aggregation

In this chapter we present a privacy-preserving data aggregation scheme which includes the encoding scheme of Chapter 4 resulting in a multi-functional privacy-preserving data aggregation scheme. In this scheme, we assume that the integrity and the authenticity of any message is preserved during transmission. Also, we assume both the aggregator and the users to be honest-but-curious, i.e. both the aggregator and the users follow the protocol. The assumption for an aggregator to be honest-but-curious is realistic since in the real-world, the reputation of the aggregator is at stake. Also, when the aggregator behaves maliciously and if it is detected, the aggregator has to pay a fine according to the GDPR [1]. Existing literature, such as [25, 57], also assume an honest-but-curious aggregator.

The protocol discussed in this chapter consists out of three phases, namely initialization, encryption and decryption. After the explanation of those phases, we analyze the protocol in terms of security, computation complexity and communication complexity.

5.1. Initialization

The initialization is only performed once and is executed by the users, except for the initialization that is needed for an encoding scheme presented in Chapter 4. Each user i generates an asymmetric key pair consisting out of a public key pk_i and a secret key sk_i. Such a key pair can, for example, be generated with the Paillier cryptosystem discussed in Section 2.3. The users send their public key to the aggregator who relays it to the other users. All users now have a public key of all other users.

In order to give the encryption in the next phase its randomness, we use the protocol as described by Erkin and Tsudik [25]. Their protocol uses additive secret sharing with 0 as the secret as discussed in Section 2.2. All users have access to a pseudo random number generator. A pseudo random number generator produces a sequence of random numbers based on an initial value, a seed. Two sequences of random numbers are the same when the seed of both sequences is the same. In our protocol, the pseudo random number generator produces random numbers that have at least the same amount of bits as the maximum coefficient α_m plus some extra bits determined by a security parameter.

Every user chooses a random seed for every other user. We denote the random seed that some user i has chosen for some other user j as $r_{i,j}$. Since the random seed may only be known to users i and j, it is encrypted with the public key of user j. The resulting ciphertext is sent directly to user j who decrypts it and retrieves $r_{i,j}$. The ciphertext does not have to go via the aggregator since we assume the users to be honest-but-curious. Now each user i shares two random seeds with every other user j, namely $r_{i,j}$ and $r_{j,i}$.

The users can be divide into groups of size k in order to reduce the bandwidth usage. Now, users only share a random seed with all other users in the same group. These groups can be created by the aggregator or by the users themselves. The aggregator can, for example, randomly divide the users into groups. The users are notified by the aggregator with a list of other users in the same group. Another set-up can be that users share a group with the users that are closest them in order to prevent a long transmission. Such a group can, for example, be all the users in the same neighborhood. Other methods and how this is done exactly is out of the scope of this thesis.

5.2. Encryption

The encryption phase is executed every round. With all the random seeds being shared among all users, the users create their secret shares. A secret share for any user i for round t is computed as

$$s_{i,t} = \sum_{j=0, j\neq i}^{u-1} r_{i,j,t} - \sum_{j=0, j\neq i}^{u-1} r_{j,i,t}, \tag{5.1}$$

where $r_{i,j,t}$ is the element at index t of the random number sequence generated by the pseudo random number generator with random seed $r_{i,j}$.

These shares are valid shares for additive secret sharing with the secret equal to 0 since all shares are random and since

$$\sum_{i=0}^{u-1} s_{i,t} = \sum_{i=0}^{u-1} \left(\sum_{j=0, j\neq i}^{u-1} r_{i,j,t} - \sum_{j=0, j\neq i}^{u-1} r_{j,i,t} \right) = \sum_{i=0}^{u-1} \sum_{j=0, j\neq i}^{u-1} r_{i,j,t} - \sum_{i=0}^{u-1} \sum_{j=0, j\neq i}^{u-1} r_{j,i,t} = 0, \tag{5.2}$$

i.e. all shares summed up equal to 0. When the users are split in groups the secret shares only consist out of random numbers shared with users in the same group. All shares of the users in the same group sum in this case to 0.

Once some user i has chosen a coefficient $\alpha_{i,t}$ for round t, that user encrypts that coefficient. The ciphertext for some user i for round t is computed as

$$c_{i,t} = \alpha_{i,t} + s_{i,t}. \tag{5.3}$$

The ciphertext $c_{i,t}$ is sent to the aggregator.

5.3. Decryption

The aggregator is not able to decrypt the ciphertext of an individual user, but it can compute the sum of all coefficients which is needed for decoding as discussed in Chapter 4. In order to retrieve this sum, the aggregator sums all the ciphertexts, i.e.

$$\sum_{i=0}^{u-1} c_{i,t} = \sum_{i=0}^{u-1} \alpha_{i,t} + \sum_{i=0}^{u-1} s_{i,t} = \sum_{i=0}^{u-1} \alpha_{i,t}. \tag{5.4}$$

Now the aggregator decodes the sum to retrieve the amount of user that have sent a certain coefficient, and therefore the corresponding value. With these amounts, the aggregator can compute any arbitrary function.

5.4. Security

In order to analyze the security of the multi-functional privacy-preserving data aggregation scheme, we view the protocol in three different perspectives, namely the perspective of any amount of users,the aggregator and any amount of users colluding with the aggregator.

Any amount of users less than $u-1$, or in the case of groups $k-1$, only have the possession of random numbers that are needed for their own secret share. With at least two honest users i and j, the secret share of an honest user contains two random numbers $r_{i,j,t}$ and $r_{j,i,t}$ which are not known to the colluding users. Therefore, the secret share of an honest user is completely random. When, however, $u-1$, or in the case of groups $k-1$ users collude, they know the secret share of the one remaining honest user. The users are assumed to be honest-but-curious and therefore are assumed to not intercept any messages. These colluding users therefore cannot use the secret share to decrypt the ciphertext. The result of the data aggregation is also only known to the aggregator which can also not be used by the colluding users to derive the value of the honest user.

The aggregator does not know any random seed or random number. Also the aggregator is assumed to be honest-but-curious and therefore does not intercept the ciphertexts containing the random seeds. The aggregator, therefore, cannot derive any information about the random seeds and the secret shares. The only information the aggregator can get is the sum of all coefficients by summing all received encrypted coefficients. From this information, the aggregator cannot derive which user has sent which value.

When $u-1$ users collude with the aggregator, those colluding parties can derive the secret share and therefore the coefficient of the one remaining honest user from the result which is retrieved by the aggregator. When the users are split in groups of size k, $k-1$ users in the same group have to collude with the aggregator

Table 5.1: Computation complexity comparison of the aggregator of our own scheme and that of others.

	Aggregator					
	Initialization			Every round		
	[32]	[58]	Ours	[32]	[58]	Ours
Exponentiation	$O(1)$	-	$O(m \cdot u)$	-	-	-
Addition	$O(u^2)$	-	$O(m \cdot u)$	$O(u^2)$	$O(m \cdot u)$	$O(u)$
Multiplication	-	-	$O(m + u^2)$	-	-	-
Modular inverse	$O(1)$	-	$O(1)$	-	-	-
Subtraction	$O(1)$	-	$O(m \cdot u)$	-	-	$O(m)$
Division	-	-	$O(u)$	-	-	$O(m)$
Modulo	$O(1)$	-	$O(1)$	-	-	$O(m)$

Table 5.2: Computation complexity comparison of a user of our own scheme and that of others, where k means the group size in which users share a random seed, m means the amount of possible values and u means the amount of users participating in the protocol.

	User					
	Initialization			Every round		
	[32]	[58]	Ours	[32]	[58]	Ours
Exponentiation	$O(k)$	$O(1)$	$O(k)$	-	$O(m \cdot k)$	-
Addition	$O(k^2)$	-	-	$O(k^2)$	$O(m \cdot k)$	$O(k)$
Multiplication	-	-	-	-	$O(m \cdot k)$	-
Modular inverse	-	-	-	-	$O(m \cdot k)$	-
Subtraction	$O(k)$	-	-	$O(k)$	-	$O(k)$
Division	-	-	-	-	-	-
Modulo	$O(k)$	$O(1)$	$O(k)$	-	-	-

in order to derive information about the remaining honest user in that group. Any amount of colluding users smaller than $u-1$, or $k-1$, is not enough to derive any information of the honest users. The colluding users do not know all random numbers contained in the secret share of an honest user. The secret share is completely random in the perspective of those colluding users. Therefore, unless $u-1$, or $k-1$ users collude with the aggregator, the colluding users cannot derive which user has sent which value.

5.5. Computation complexity

In this analysis, we use the key that is shared between two users and produced by a Diffie-Hellman key exchange [21] as a random seed. We use this technique since Gong et al. [32] also make use of this technique which makes our scheme more comparable to theirs.

The initialization is only executed once. Each user performs a Diffie-Hellman key exchange with every other user in the same group which requires $O(k)$ exponentiations and modulo operations. After the key exchange, all users have their random seeds which are used in the remainder of the protocol. In addition, the encoding scheme needs to be initialized by the aggregator.

During encryption a user computes its secret share which requires $O(k)$ additions and subtractions. Computing the ciphertext only requires 1 addition. During decryption, the aggregator must sum all ciphertext resulting in $O(u)$ additions. Finally the decoding algorithm is executed as discussed in Chapter 4.

Tables 5.1 and 5.2 compare the computation complexity with that of other comparable multi-functional privacy-preserving data aggregation schemes. Other schemes are not included since they have a characteristic which makes them incomparable. Examples of such characteristics are the reliance on a trusted authority and approximate results instead of exact ones.

The scheme of Gong et al. [32] requires an interactive protocol during initialization in order to determine the private sequence number of a user. Due to this protocol, the initialization for a user is more expensive in comparison to the initialization for a user in our scheme. For the aggregator, the initialization of our scheme is more expensive. This is due to the fact that the aggregator creates the coefficients during initialization. During a round the computation complexity of both schemes are not significantly different. There is one important thing to notice which contributes to an extra factor of k for a user during a round in the scheme of [32]. The scheme of [32] requires each user to send a ciphertext for every other user in the same group.

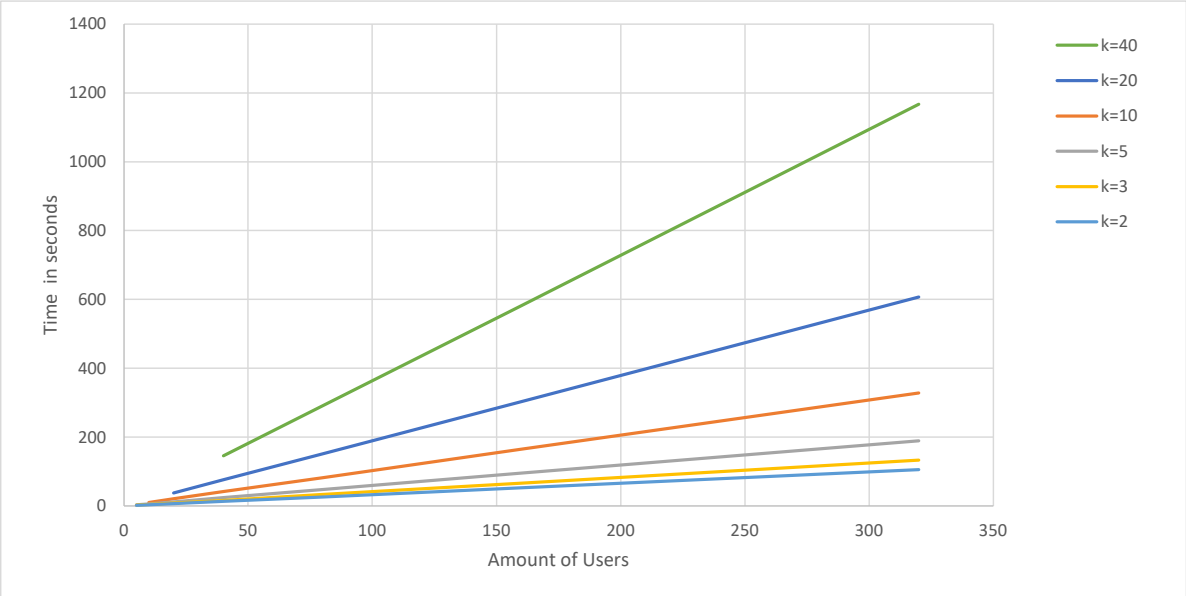

Figure 5.1: This figure illustrates the relation between the amount of users participating in the protocol and the time it takes to execute the initialization. The time is taken in seconds and is the average of 10 experiments. Different lines in the graph represent different group sizes.

Not every ciphertext can include the same secret share since this would leak the private data. That scheme, therefore, requires a different secret share to be computed for every ciphertext which results in an extra factor of k. Our scheme, on the other hand, requires a user to send only one message and therefore does not have this problem.

For PriSense presented by Shi et al. [58] we assume the use of the ElGamal cryptosystem. Their paper states that any public cryptosystem can be used for sharing the shares. We assume the ElGamal cryptosystem since it is a public cryptosystem which resembles the most with the Diffie-Hellman key exchange used in the other schemes. PriSense requires each user to send shares of their data of a round to every other user in the same group. This results in the fact that the costs for creating randomness for encryption are taken every round instead of during initialization as is the case with our scheme. Additionally, in order to retrieve the multi-functionality, this process is repeated for every possible value which results in an extra factor of m for both the aggregator and the user. The initialization for as well the aggregator and the users of the PriSense scheme barely requires any computation.

On overall, the computation complexity of our scheme, compared to PriSense, is better considering every round. This is due to the fact that our scheme does not require to repeat the process of aggregating data for every possible value. During initialization, PriSense performs better since the initialization is moved to be performed in every round. Compared to [32], our scheme performs similar with some small differences. With respect to the computation complexity, our protocol does not significantly outperforms all comparable state-of-the-art multi-functional privacy-preserving data aggregation schemes.

5.6. Experimental run-time

In this section we discuss the results of the experiments we performed to measure the run-time. First we measured the run-time of the initialization and second the run-time of a round after initialization. All experiments are executed with an Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz processor and with 16GB RAM memory on Windows 10 Home. The experiments are implemented in Java 14 with built-in functions and are executed 10 times of which the average is taken.

Figure 5.1 illustrates the dependence of the run-time of the initialization on the amount of users. During the experiments we used the encoding scheme with maximum certainty to initialise the coefficients. Different lines in the graph represent different group sizes. Note that the group size is not allowed to be higher than the amount of user participating in the protocol. Therefore, not all lines start at the same x-coordinate.

The graph shows that the run-time is linear with respect to both the amount of users and the group size. Additional experiments showed that the amount of values used in the protocol does not influence the run-time

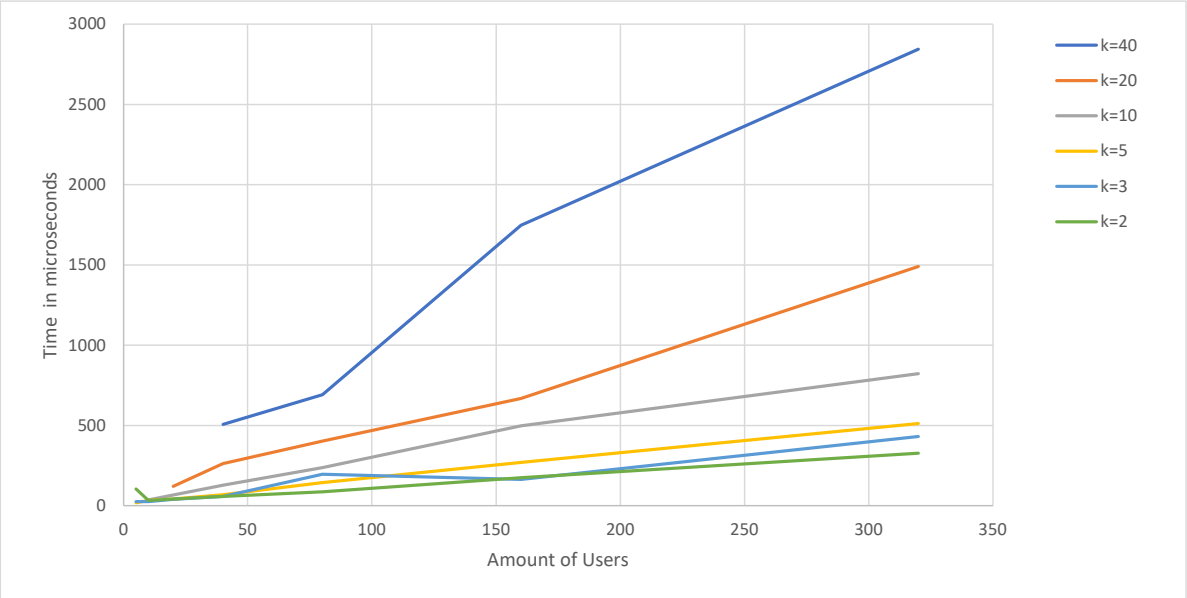

Figure 5.2: This figure illustrates the relation between the amount of users participating in the protocol and the time it takes to execute a round of our protocol after initialization. The time is taken in microseconds and is the average of 10 experiments. Different lines in the graph represent different group sizes.

Table 5.3: Communication complexity comparison of our own scheme and that of others, where b means the amount of bits needed for the largest message.

Initialization			Every round		
[32]	[58]	Ours	[32]	[58]	Ours
$O(u \cdot k \cdot b)$	$O(u \cdot b)$	$O(u \cdot k \cdot b)$	$O(u \cdot k \cdot b)$	$O(m \cdot u \cdot k \cdot b)$	$O(u \cdot b)$

of the initialization. This is due to the fact that the amount of values is limited because of the encoding scheme.

Figure 5.2 illustrates the dependence of the run-time of a round after initialization on the amount of users. Different lines in the graph represent different group sizes. Note that the group size is not allowed to be higher than the amount of users participating in the protocol. Therefore not all lines start at the same x-coordinate. Also, the run-time does not depend on the choice of encoding scheme since the computational difference of both encoding schemes is during initialization. In a round after initialization, both encoding schemes perform exactly the same.

The graph shows that the run-time grows linearly with the amount of users and the group size. Due to the fact that the run-time for a smaller amount of users is only a few microseconds, those results have a relatively higher standard deviation. The results of such fast execution are influenced more by background processes on a computer. Additional experiments showed that the amount of values used in the protocol does not influence the run-time of a round after initialization. This is due to the fact that the amount of values is limited because of the encoding scheme.

5.7. Communication complexity

During initialization, every user sends a message with a random seed to every other user in the group resulting in $O(u \cdot k \cdot b)$ bits. With b we denote the maximum amount of bits needed for the message space. In a round, every user sends its message resulting in $O(u \cdot b)$ bits in total.

Figure 5.3 summarizes the communication in our protocol. We assume that lists V and A are already pre-loaded on the device of the user beforehand.

Table 5.3 presents the comparison of the communication complexity with other scheme. PriSense shares its data every round with all other users in the same group for every possible value resulting in an extra factor of $m \cdot k$. During initialization, the users only have to publish their public key which requires a factor k less than our scheme. The scheme of Gong et al. [32] has an extra factor k in a round since every user must send a message for every other user in the same group. During initialization, our scheme performs similar to [32].

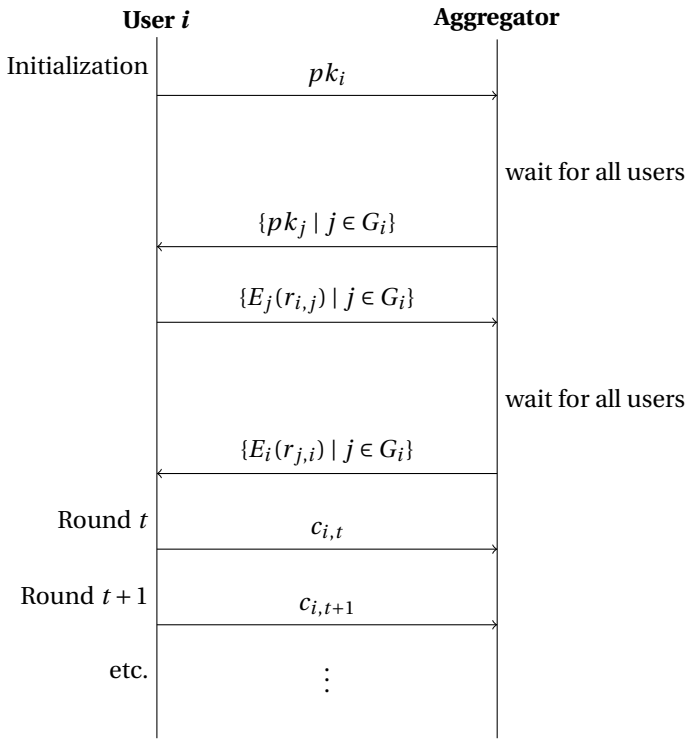

Figure 5.3: An overview of the communication in our protocol where we denote G_i as the set of all other users that are in the same group as user i.

We assume that all schemes use the same level of security and therefore the same amount of bits for the encryption.

On overall, the communication complexity of our scheme is a factor k better than [32] and a factor $m \cdot k$ better than [58]. That means that our protocol requires significant less bandwidth than comparable state-of-the-art multi-functional privacy-preserving aggregation schemes. This improvement comes from the fact that in our scheme each user only sends one message per round. In addition, the size of such a message is independent of any other variable.

6

Multi-Functional Privacy-Preserving Data Aggregation with Malicious User Detection

Chapter 5 presented a multi-functional privacy-preserving data aggregation scheme which relies on the assumption that users are honest-but-curious. However, this assumption may not always be realistic. Whether this assumption is realistic, depends on the application. In an application where only certified users participate, the assumption is more realistic in comparison to an application where every random user can participate. Therefore, we present a multi-functional privacy-preserving data aggregation scheme in this chapter which assumes that only the aggregator is honest-but-curious. The protocol assumes there are y adaptive and malicious users while the other $u - y$ users are honest-but-curious, where u is again the total amount of users participating in the protocol. The goal of the protocol is to detect the malicious users in order to take appropriate action upon them. The protocol consists out of four phases. The protocol starts with a one-time initialization. Next, the submission which is executed every round. After that, the ciphertext of every user needs to be verified during the verification phase. Last, when all users are verified, the aggregator decrypts the result.

6.1. Initialization

First, the aggregator performs the key generation of the Paillier cryptosystem which is explained in Section 2.3. The aggregator keeps the secret key $sk_A = (\lambda_A, \mu_A)$ secret and publishes the public key $pk_A = (n_A, g_A)$ in addition with h_A and H to all the users. The value h_A is chosen randomly from $\mathbb{Z}^*_{n_A}$ and H is a secure cryptographic hash function with as input $\{0,1\}^*$ and as output a value from $\mathbb{Z}_{n_A}$. The aggregator also performs the initialization of an encoding scheme presented in Chapter 4 which gives a list of values V and a list of coefficients A. The list of coefficients is updated later during initialization in order to accommodate the signatures as we discuss in a bit. The upper limit n of the coefficients defined in Equation 4.3 is hereby set to n_A since the aggregate message must be in Z_{n_A}.

Also each user i performs the key generation of the Paillier cryptosystem which gives them a public key $pk_i = (g_i, n_i)$ and a secret key $sk_i = (\lambda_i, \mu_i)$. Each user sends its public key to the aggregator who relays it to all other users. In this way, all users have the public key of every other user. Note that the aggregator is assumed to be honest-but-curious and therefore can be trusted to relay the correct public keys to all the users.

As a part of the malicious user detection, the aggregator creates a signature for each coefficient of the encoding scheme. For these signatures, we use the Boneh-Boyen signature scheme [10]. We make one modification in the signature scheme in order to combine the signatures with the Paillier cryptosystem that we use for encryption. Instead for the signature scheme to operate in a group of order p, our protocol requires it to operate in a group of order n_A^2, where n_A is the same as the n_A in the public key generated by the aggregator. We also only need one group $\mathbb{G}_1$ since we only have to create signatures. The verification of the signatures is done with a zero-knowledge proof which does not require the usage of the public key.

Next, the aggregator creates a generator g_S for the group $\mathbb{G}_1$ of order n_A^2 and a random integer $x \in \mathbb{Z}_{n_A}$ which is the secret key of the signature scheme. The aggregator now computes the signature for each coefficient α_i as

$$\sigma_i = g_S^{\frac{1}{x+\alpha_i}} \mod n_A^2. \tag{6.1}$$

Computing $\frac{1}{x+\alpha_i}$ is equal to computing the modular inverse of $x+\alpha_i$ modulo n_A. However this modular inverse does not exist when $x+\alpha_i$ is not coprime to n_A. If the modular inverse does not exist, the coefficient α_i is incremented with steps of one until the modular inverse does exist. When $x+\alpha_i$ has a modular inverse, all subsequent coefficients are updated accordingly to satisfy the conditions stated in Chapter 4. The list of coefficients A is changed in order to contain the updated coefficients.

The lists of all values V, updated coefficients A and signatures σ are distributed to all users, where coefficient α_i again corresponds to value v_i for every i and where signature σ_i corresponds to coefficient α_i for every i.

6.2. Submission

As we did in the protocol for honest-but-curious users in Chapter 5, the users share a random number. However, since we deal with malicious users, we have to make sure that each user includes the correct random shares in the eventual ciphertext. Therefore, the users share a random number for every encryption instead of one random seed during the initialization.

Each user i chooses a random number for every other user j in round t. This random number is denoted by $r_{i,j,t}$. Again, note that we can split the users in groups to decrease the amount of communication. Now, each user only chooses a random number for every other user in the same group. User i then sends $E_i(r_{i,j,t})$ and $E_j(r_{i,j,t})$ to the aggregator, where $E_k(m)$ is the Paillier encryption of a message m under the public key of user k.

In order to check that both, $E_i(r_{i,j,t})$ and $E_j(r_{i,j,t})$, for some users i and j, contain the same random number, an additional non-interactive zero knowledge proof is sent by user i. We use the non-interactive zero knowledge proof of plaintext equality described by Dimitriou and Awad [22] as we discussed in Section 2.5.

Once the aggregator has verified that $E_i(r_{i,j,t})$ and $E_j(r_{i,j,t})$ contain the same random numbers, it stores both ciphertexts, which are needed for the verification discussed in the next section, and relays $E_j(r_{i,j,t})$ to user j who decrypts it. When the verification of the plaintext equality fails, user i is marked as malicious and corresponding action can be taken, such as removal or inspection.

When every user has sent and received random shares from all other users and the aggregator verified the plaintext equality of every share, each user i creates its secret share $s_{i,t}$ for round t. In contrast to the protocol with honest-but-curious users, in this protocol we use additive secret sharing with the secret equal to $u \cdot n_A$. The secret share of a user i is computed as

$$s_{i,t} = n_A + \sum_{j=0, j\neq i}^{u-1} r_{i,j,t} - \sum_{j=0, j\neq i}^{u-1} r_{j,i,t}. \tag{6.2}$$

The summation of all secret shares for some round t return $u \cdot n_A$ since

$$\sum_{i=0}^{u-1} s_i = \sum_{i=0}^{u-1} n_A + \sum_{i=0}^{u-1}\sum_{j=0, j\neq i}^{u-1} r_{i,j,t} - \sum_{i=0}^{u-1}\sum_{j=0, j\neq i}^{u-1} r_{j,i,t} = \sum_{i=0}^{u-1} n_A = u \cdot n_A. \tag{6.3}$$

Each user i now encrypts its coefficient $\alpha_{i,t}$ as

$$c_{i,t} = g_A^{\alpha_{i,t}} \cdot h_A^{s_{i,t}} \mod n_A^2 \tag{6.4}$$

and sends it to the aggregator. Each user i also sends $E_i(n_A)$ and $E_A(n_A)$ together with a proof of plaintext equality to the aggregator which is needed in the verification phase discussed next.

6.3. Verification

There are three ways a malicious user can behave in order for the aggregator to be unable to retrieve any result or correct results. The first way is that some user initially sends random numbers to other users so that they include them in their secret share, but that the malicious user does not send its encrypted coefficient. Therefore, that secret share is not included and the aggregate of the other ciphertexts is not decryptable anymore. This malicious behaviour is easy to detect since the aggregator can check who has sent random numbers and who did not send a ciphertext.

Another way a malicious user can behave maliciously is by sending a coefficient which is not in the list of valid coefficients A or by sending more than one coefficient. As a consequence, the aggregator is not able to retrieve correct results. Therefore, the aggregator checks whether $\alpha_{i,t}$ for some user i at round t is a valid

coefficient. In order to verify this, we need another non-interactive zero knowledge proof. The proof we use is based on the proof presented by Arfaoui et al. [3]. There are, however, some changes. The groups we work with are different since we combine it with the Paillier cryptosystem. We also do not need the aggregator to send a challenge since we want it non-interactive which brings some changes along with it. The values that are outputted by the prover differs. Also the hash function is computed over a different set over values. The modified proof is described in Algorithm 4, where we denote the signature corresponding to coefficient $\alpha_{i,t}$ as $\sigma_{i,t}$. In this proof the values of s_1, s_2 and s_3 mask the real coefficient, secret share and the value for ℓ respectively. This last value, ℓ, is used to randomize the signature so that the aggregator cannot link the signature back to the coefficient. The values D and D_1 are used in order for the verifier to check that the used signature corresponds to the coefficient that is included in s_1. The ciphertext $c_{i,t}$ and the value $c_{i,t,1}$ are then used to check whether the coefficient in s_1 is the same as the coefficient used in the ciphertext.

Algorithm 4: Non-interactive proof of set membership

Prover (user i):
Input: $\{c_{i,t}, \alpha_{i,t}, \sigma_{i,t}, s_{i,t}, g_A, g_S, h_A, n_A, H, \sum_{j=0}^{u-1} r_{i,j,t}\}$
Choose random $\ell, \ell', \alpha' \in \mathbb{Z}_{n_A}$
Compute $s' = \sum_{j=0}^{u-1} r_{i,j,t}$
Compute $B = \sigma_{i,t}^{\ell} \mod n_A^2$
Compute $B_1 = B^{-1} \mod n_A^2$
Compute $D = B_1^{\alpha_{i,t}} g_S^{\ell} \mod n_A^2$
Compute $c_{i,t,1} = g_A^{\alpha'} h_A^{s'} \mod n_A^2$
Compute $D_1 = B_1^{\alpha'} g_S^{\ell'} \mod n_A^2$
Compute $h = H(c_{i,t}, B, D)$
Compute $s_1 = \alpha' + h \cdot \alpha_{i,t} \mod n_A$
Compute $s_2 = s' + h \cdot s_{i,t}$
Compute $s_3 = \ell' + h \cdot \ell \mod n_A$
Output: $\{B, D, c_{i,t,1}, D_1, s_1, s_2, s_3\}$

Verifier (aggregator):
Input: $\{c_{i,t}, B, D, c_{i,t,1}, D_1, s_1, s_2, s_3, g_A, g_S, h_A, n_A, x, H\}$
Compute $D' = B^x$
Compute $h = H(c_{i,t}, B, D')$
Compute $c'_{i,t,1} = g_A^{s_1} h_A^{s_2} c_{i,t}^{-h} \mod n_A^2$
Compute $D'_1 = B_1^{s_1} g_S^{s_3} D'^{-h} \mod n_A^2$
Verify that $D = D', c_{i,t,1} = c'_{i,t,1}$ and $D_1 = D'_1$

If the verification fails, that user is marked as malicious and corresponding action can be taken. Otherwise, the user has encrypted a valid coefficient.

There is one way left for a malicious user to act maliciously. A malicious user can include a wrong secret share which causes the aggregate ciphertext to not be decryptable. In Algorithm 4, there is already a check that the secret share included in s_2 must be equal to the secret share in the ciphertext. So if we verify that s_2 contains the correct secret share, it is verified that the ciphertext does.

The aggregator already has knowledge of $E_i(r_{i,j,t})$ and $E_i(r_{i,j,t})$ for every user i and j. Each user i also sent $E_i(n_A)$ and $E_A(n_A)$ to the aggregator. The aggregator verifies whether the plaintext of both ciphertexts is equal and whether the plaintext is equal to n_A by decrypting $E_A(n_A)$. Now, the aggregator has enough knowledge to gain the encrypted secret share $s_{i,t}$ for each user i by computing

$$E_i(s_{i,t}) = E_i(n_A) \cdot \frac{\prod_{j=0, j\neq i}^{u-1} E_i(r_{i,j,t})}{\prod_{j=0, j\neq i}^{u-1} E_i(r_{j,i,t})}. \tag{6.5}$$

Now, the aggregator also computes the encryption of s_2 as

$$E_i(s_2) = \prod_{j=0, j\neq i}^{u-1} E_i(r_{i,j,t}) \cdot E_i(s_{i,t})^h. \tag{6.6}$$

User i creates $E_i(s_2)$ in the same way as the aggregator, with the public ciphertexts of n_A and the random numbers. In this way, $E_i(s_2)$ computed by user i and the aggregator are exactly the same, including the randomness. User i also computes $E_A(s_2)$ which is the normal encryption again with the public key of the aggregator. The user sends both $E_i(s_2)$ and $E_A(s_2)$ to aggregator together with a proof of plaintext equality according to Algorithm 2.

Finally, the aggregator performs three checks. The first check is whether the plaintext of $E_i(s_2)$ and $E_A(s_2)$ is equal. The second check is whether $E_i(s_2)$ received by the user is equal to the one the aggregator computed. The final check is whether the plaintext is equal to s_2 which is checked by decrypting $E_A(s_2)$. Again, if one of these checks fails, it means that the user is maliciously sending the wrong secret share and is marked as malicious. Note that a user can also use a value different than g_A and h_A, however, this will cause the proof of Algorithm 4 to fail and is, therefore, also detected.

6.4. Decryption

When all users are verified to behave according to the protocol, the aggregator aggregates the ciphertexts and decrypts the sum of the coefficients. The aggregator aggregates the ciphertexts by computing

$$c_t = \prod_{i=0}^{u-1} c_{i,t} = \prod_{i=0}^{u-1} g_A^{\alpha_{i,t}} \cdot h_A^{s_{i,t}} = g_A^{\sum_{i=0}^{u-1} \alpha_{i,t}} \cdot h_A^{\sum_{i=0}^{u-1} s_{i,t}} = g_A^{\sum_{i=0}^{u-1} \alpha_{i,t}} \cdot (h_A^u)^{n_A}. \tag{6.7}$$

In order to decrypt a ciphertext with the Paillier cryptosystem, the ciphertext must be in the form $g_A^m r^{n_A}$, where $m \in \mathbb{Z}_n$ and $r \in \mathbb{Z}_n^*$. The aggregate ciphertext c_i has this form if we set $m = \sum_{i=0}^{u-1} \alpha_{i,t}$ and $r = h_A^u$. Therefore the aggregator can decrypt c_i with its secret key and obtains $\sum_{i=0}^{u-1} \alpha_{i,t}$. This sum of coefficients is decoded according to the encoding schemes presented in Chapter 4 resulting in a histogram of values and the ability to compute any arbitrary function.

6.5. Security

In order to prove the security of the protocol, we prove that various subsets of parties cannot gain any information about the private data of honest users which are outside that subset. The subsets of parties we discuss are 1. Any amount of users 2. The aggregator 3. Multiple users colluding with the aggregator. When we discuss any amount of users, we also prove, under certain assumptions, that a user cannot act maliciously without being detected by the aggregator. Finally, we also prove the correctness of the protocol where necessary.

6.5.1. Users

In this subsection we prove three theorems regarding any amount of users. The first one concerns honest users which may not be marked as malicious during the verification phase. The second one concerns the privacy of the private data of other users and the last one concerns the detection when a user acts maliciously. The first theorem we prove is stated in Theorem 1.

Theorem 1. *Honest users pass the verification phase without being marked as malicious.*

The proof of this theorem is based on two checks which are performed during verification. The first one is the set membership verification of the coefficient that is included in the ciphertext. The second one is the verification of the secret share used in the ciphertext. When both checks pass, a user is not marked as malicious. The correctness of the first check is stated in Lemma 2. This check only contains the zero-knowledge proof and therefore, proving the completeness property of this proof is sufficient.

Lemma 2. *A user with a correct coefficient in its ciphertext passes the verification of the set membership proof and is not marked as malicious.*

Proof. In order to proof this we prove the completeness property of the zero-knowledge proof of set membership which means that every user with a correct coefficient has to pass the verification procedure. We prove three equalities in order to prove the completeness property, namely $D = D'$, $c_{i,t,1} = c'_{i,t,1}$ and $D_1 = D'_1$.

First, the equation $D = D'$ is proved by

$$\begin{aligned} D &= B_1^{\alpha_i} g_S^{\ell} = B^{-\alpha_i} g_S^{\ell} = \sigma_i^{-\alpha_i \cdot l} g_S^{\ell} \\ &= g_S^{\frac{-\alpha_i \cdot \ell}{\alpha_i + x}} g_S^{\ell} = g_S^{\ell - \frac{\alpha_i \cdot \ell}{\alpha_i + x}} = g_S^{\frac{x \cdot \ell}{\alpha_i + x}} \\ &= \sigma_i^{x \cdot \ell} = B^x = D'. \end{aligned}$$

Next, the equation $c_{i,t,1} = c'_{i,t,1}$ follows from

$$\begin{aligned} c'_{i,t,1} &= g_A^{s_1} h_A^{s_2} c_{i,t}^{-h} \\ &= g_A^{\alpha'+h\cdot\alpha_i} h_A^{s'+h\cdot s_{i,t}} g_A^{-h\cdot\alpha_i} h_A^{-h\cdot s_{i,t}} \\ &= g_A^{\alpha'} h_A^{s'} = c_{i,t,1}. \end{aligned}$$

Finally, the last equation $D_1 = D'_1$ follows from

$$\begin{aligned} D'_1 &= B_1^{s_1} g_S^{s_3} D'^{-h} = \sigma_i^{-\ell\cdot s_1} g_S^{s_3} \sigma_i^{-h\cdot x\cdot\ell} \\ &= \sigma_i^{-\ell\cdot\alpha'-\ell\cdot h\cdot\alpha_i} g_S^{\ell'+h\cdot\ell} \sigma_i^{-h\cdot x\cdot\ell} \\ &= g_S^{\frac{-\ell\cdot\alpha'-\ell\cdot h\cdot\alpha_i}{\alpha_i+x}} g_S^{\ell'+h\cdot\ell} g_S^{\frac{-h\cdot x\cdot\ell}{\alpha_i+x}} \\ &= g_S^{\frac{-\ell\cdot\alpha'-\ell\cdot h\cdot\alpha_i}{\alpha_i+x}} g_S^{\frac{\alpha_i\cdot\ell'+\alpha_i\cdot h\cdot\ell+x\cdot\ell'+x\cdot h\cdot\ell}{\alpha_i+x}} g_S^{\frac{-h\cdot x\cdot\ell}{\alpha_i+x}} \\ &= g_S^{\frac{-\ell\cdot\alpha'+x\cdot\ell'+\alpha_i\cdot\ell'}{\alpha_i+x}} = g_S^{\frac{-\ell\cdot\alpha'}{\alpha_i+x}} g_S^{\ell'} = B_1^{\alpha'} g_S^{\ell'} = D_1. \end{aligned}$$

Therefore, the zero-knowledge proof satisfies the completeness property and consequently a user which acts honestly passes the verification of the zero-knowledge proof of set membership. □

The other check which is performed in the verification phase concerns the verification of the secret share which leads to Lemma 3

Lemma 3. *A user which includes the correct secret share in its ciphertext passes the verification of the secret share and is not marked as malicious.*

Proof. The part of this check starts when the users share a random number with other users. An honest user sends a random number encrypted two times under two different keys, its own and that of the recipient. The user also includes a proof of plaintext equality which the aggregator verifies. This proof satisfies the completeness property as proven in [6]. The same holds for the plaintext equality of n_A encrypted two times under two different keys, the user its own key and that of the aggregator.

Now, a user i and the aggregator have the same ciphertexts of all the random numbers $E_i(r_{i,j,t})$ and $E_i(r_{j,i,t}$ for every j and both have the same ciphertext $E_i(n_A)$. Both can now compute the same ciphertext of the secret share of user i since

$$E_i(n_A) \cdot \frac{\prod_{j=0,j\neq i}^{u-1} E_i(r_{i,j,t})}{\prod_{j=0,j\neq i}^{u-1} E_i(r_{j,i,t})} = E_i(n_A) \cdot \frac{E_i(\sum_{j=0,j\neq i}^{u-1} r_{i,j,t})}{E_i(\sum_{j=0,j\neq i}^{u-1} r_{j,t,t})} = E_i(n_A + \sum_{j=0,j\neq i}^{u-1} r_{i,j,t} - \sum_{j=0,j\neq i}^{u-1} r_{j,i,t}) = E_i(s_{i,t}). \quad (6.8)$$

From the zero-knowledge proof for the set membership, the aggregator knows the values of s_2 and h. With this information, the verification of the secret share can be finished. Both user i and the aggregator compute the same ciphertext of s_2 since

$$\prod_{j=0,j\neq i}^{u-1} E_i(r_{i,j,t}) \cdot E_i(s_{i,t})^h = E_i(s' + h \cdot s_{i,t}) = E_i(s_2). \quad (6.9)$$

The user also honestly computes $E_A(s_2)$ and a proof of plaintext equality. Again, due to the completeness property, the plaintext equality is verified. The aggregator decrypts $E_A(s_2)$ which results in the correct s_2 as used in the zero-knowledge proof of set membership and finishes the check of a secret share. Therefore, an honest user passes the secret share verification. □

Both checks that are performed in the verification phase do not mark an honest user as malicious and therefore, Theorem 1 is also proved to hold.

The next theorem we prove concerns the privacy of private data of other honest users and is stated in Theorem 4.

Theorem 4. *With at least two honest users, any amount of colluding users with polynomial bounded computing power cannot gain any information about the private data of an honest user except for the distribution of the data of all honest users assuming that the decisional composite residuosity assumption holds.*

There are three types of messages that are sent in the network, namely ciphertexts, proof of set membership and proof of plaintext equality. For each of these types we prove that they leak no information which is done in the following two lemmas. The fact that the proof of plaintext equality does not leak any information is already proved in [6].

Lemma 5. *Any number of colluding users of at most $u-2$, or in the case of groups $k-2$, with polynomial bounded computing power cannot gain any information from a ciphertext assuming that the decisional composite residuosity assumption holds.*

Proof. There are two types of Paillier ciphertexts that we user in our protocol. The first type are the default ciphertexts which are computed as described by the Paillier cryptosystem. As discussed in Section 2.3, the Paillier cryptosystem is semantic secure under CCA1 and CPA assuming that the decisional composite residuosity assumption holds. This means that a user with polynomial bounded computing power cannot gain any information from theses ciphertexts

The other type of ciphertexts are the encrypted coefficients of users. Despite the modification we made for this encryption, the semantic security is still ensured. The Paillier cryptosystem uses a random number $r \in \mathbb{Z}^*_{n_A}$ and raises that to the power n_A. We replace this r with h_A which is also in $\mathbb{Z}^*_{n_A}$. To give the cipher its randomness we raise h_A to the power of a secret share. In the eyes of the colluding users, the secret share of an honest user is completely random. An honest user i shares random numbers $r_{i,j}$ and $r_{j,i}$ with at least one other honest user j. These random numbers are not known to the colluding users since they are encrypted according to the Paillier cryptosystem and, as discussed before, do not leak any information. The secret share of an honest user is therefore random since it contains unknown random numbers. The encrypted message of an honest user is, therefore, uniformly distributed to the ciphertext space in the eyes of the colluding users. Since the message in our protocol is encrypted in the same way as in the Paillier cryptosystem and the source of randomness in our protocol corresponds to the source of randomness in the Paillier cryptosystem, this type of ciphertext also falls under the semantic security of the Paillier cryptosystem.

Therefore, any number of colluding users of at most $u-2$, or in the case of groups $k-2$, with polynomial bounded computing power cannot gain any information from a ciphertext assuming that the decisional composite residuosity assumption holds. □

Lemma 6. *The zero-knowledge proof of set membership does not leak any information.*

Proof. In order to proof this, we proof the zero knowledge property of the zero-knowledge proof which means that the zero-knowledge proof does not leak any information. In order to prove this zero knowledge property, we replace the proof of the user, also the prover, V with a simulated proof S. The simulated proof is generated without any knowledge of the secret α_j of the user. The zero-knowledge proof has the zero knowledge property if S and V are indistinguishable.

The simulated proof, where we ignore the subscript t, can be constructed as follows

1. Choose a random coefficient α_i with its corresponding signature A_i.
2. Choose a random $\ell, s_1, s_2, s_3 \in \mathbb{Z}_n$.
3. Choose a random $c_i \in \mathbb{Z}^*_{n^2}$.
4. Compute $B = A_i^\ell$, $B_1 = B^{-1}$ and $D = B_1^{\alpha_i} g_S^\ell$.
5. Compute $h = H(c_i, B, D)$
6. Compute $c_{i,1} = g_A^{s_1} h_A^{s_2} c_i^{-h}$ and $D_1 = B_1^{s_1} g_S^{s_3} D^{-h}$.
7. Send $S = \{c_i, B, D, c_{i,1}, D_1, s_1, s_2, s_3\}$.

The simulated proof S is indistinguishable from the proof V of a user in the eyes of the aggregator. In both proofs s_1, s_2 and s_3 are random in $\mathbb{Z}_n$ and in both proofs $c_i, B, D, c_{i,1}$ and D_1 are random in $\mathbb{Z}^*_{n^2}$.

The verification of this simulated proof also succeeds. The value D, in S, is equal to B^x computed by the aggregator according to the same reasoning as discussed in the proof of Lemma 2. Also, $c_{i,1}$ and D_1 of S are equal to, respectively, $c'_{i,1}$ and D'_1 computed by the aggregator according to their definitions. Therefore, the zero-knowledge proof does not leak information. □

Now we have proven that every type of message that is sent in the network does not leak any information to any number of colluding users of at most $u-2$, or $k-2$ when working with groups, with polynomial bounded computing power assuming that the decisional composite residuosity assumption holds. The last thing to prove is that any combination of messages of these types do not leak information. The proofs combined of multiple users do not leak any information since the randomness is independent of each other. The randomness in a ciphertext of a coefficient is dependent on the randomness used by other users. When all ciphertexts of a group of users are multiplied, the secret shares and therefore the randomness cancels out. The result is $g_A^{\alpha_t} \cdot (h_A^u)^{n_A}$, where α_t is the sum of coefficients in that group of users at round t. The term $(h_A^u)^{n_A}$ can be removed since this is public knowledge. When the colluding users are able to compute α_t from $g_A^{\alpha_t}$, Algorithm 3 can be used to compute the histogram of coefficients. However, there is no link from the a specific coefficient to a specific user. Therefore the colluding users cannot gain any information about the private data of an honest user except for the distribution of the data of all honest users which proves Theorem 4.

The final theorem which we prove in this section concerns the detection of a malicious user which is stated in Theorem 7.

Theorem 7. *A user acting maliciously is detected assuming that the q-SDH assumption holds.*

There are multiple ways a user can act maliciously. The first is that a user can share random numbers, so that other users include those random numbers in their secret share, without sending any ciphertext during the submission phase. Now, the secret shares do not cancel out and the result is undecryptable. The aggregator can detect this user since it knows which users sent random numbers to other users and did not send an encrypted coefficient.

There are two other ways a user can act maliciously, namely using a wrong coefficient which results in wrong results and using a wrong secret share which results in an undecryptable result. The detection of these two action are proved in the following two lemmas. Note that a user can also use wrong public parameters during encryption, but if a user does this, the zero-knowledge proof of set membership fails and the user is detected.

Lemma 8. *The zero-knowledge proof of set membership always detects a wrong coefficient assuming that the q-SDH assumption holds.*

Proof. For this lemma we prove the soundness property of the zero-knowledge proof which means that a user with a wrong coefficient cannot pass the verification. Suppose a user has a wrong coefficient α_x in its ciphertext $c_{i,t} = g_A^{\alpha_x} h_A^{s_{i,t}}$. Due to the check of the verifier that $c_{i,t,1}$ must be equal to $c'_{i,t,1}$, s_1 must include this wrong coefficient α_x as well. If s_1 would contain a valid coefficient α_v, this check fails since

$$\begin{aligned} c'_{i,t,1} &= g_A^{s_1} h_A^{s_2} c_{i,t}^{-h} \\ &= g_A^{\alpha' + h\cdot\alpha_v} h_A^{s' + h\cdot s_{i,t}} g_A^{-h\cdot\alpha_x} h_A^{-h\cdot s_{i,t}} \\ &= g_A^{\alpha' + h\cdot\alpha_v - h\cdot\alpha_x} h_A^{s'} \neq c_{i,t,1}. \end{aligned}$$

The other check of the verifier, that D'_1 must be equal to D_1, requires the coefficient in s_1 to be equal to the coefficient in the provided signature. Otherwise this check fails since

$$\begin{aligned} D'_1 &= B_1^{s_1} g_S^{s_3} D'^{-h} = \sigma_i^{-\ell\cdot s_1} g_S^{s_3} \sigma_i^{-h\cdot x\cdot\ell} \\ &= \sigma_i^{-\ell\cdot\alpha' - \ell\cdot h\cdot\alpha_x} g_S^{\ell' + h\cdot\ell} \sigma_i^{-h\cdot x\cdot\ell} \\ &= g_S^{\frac{-\ell\cdot\alpha' - \ell\cdot h\cdot\alpha_x}{\alpha_v + x}} g_S^{\ell' + h\cdot\ell} g_S^{\frac{-h\cdot x\cdot\ell}{\alpha_v + x}} \\ &= g_S^{\frac{-\ell\cdot\alpha' - \ell\cdot h\cdot\alpha_x}{\alpha_v + x}} g_S^{\frac{\alpha_v\cdot\ell' + \alpha_v\cdot h\cdot\ell + x\cdot\ell' + x\cdot h\cdot\ell}{\alpha_v + x}} g_S^{\frac{-h\cdot x\cdot\ell}{\alpha_v + x}} \\ &= g_S^{\frac{-\ell\cdot\alpha' - \ell\cdot h\cdot\alpha_x + x\cdot\ell' + \alpha_v\cdot\ell' + \alpha_v\cdot h\cdot\ell}{\alpha_i + x}} \neq D_1, \end{aligned}$$

where, again, α_x is an invalid coefficient and α_v is a valid coefficient. Since the coefficient in s_1 must be equal to α_x, this check forces the prover to provide a valid signature for a wrong coefficient α_x. Since Boneh-Boyen signatures are unforgeable based on the hardness of the q-SDH problem, the prover cannot pass this check. Therefore, every user with a wrong coefficient is detected by this zero-knowledge proof assuming that the q-SDH assumption holds. □

Lemma 9. *A user which uses the wrong secret share in its ciphertext is detected by the aggregator.*

Proof. The secret share in the ciphertext must be equal to the secret share in s_2 from the zero-knowledge proof of set membership. The aggregator knows the encryption under the public key of user i of all random numbers and other relevant values for user i. The user therefore has to send the correct $E_i(s_2)$. A different ciphertext is detected by the aggregator. Due to the proof of plaintext equality, which satisfies the soundness property as proved by [6], the user must send the correct $E_A(s_2)$. Therefore, s_2 and, as a consequence, the ciphertext of the coefficient must contain the correct secret share. □

Therefore, any possible action a malicious user can do is detected assuming that the q-SDH assumption holds which proves Theorem 7.

6.5.2. The aggregator

We prove two theorems regarding the aggregator. The first theorem concerns the correct retrieval of the sum of the coefficients of all users. The second theorem concerns the information that the aggregator learns about the private data of a user. Note that the aggregator is assumed to be honest-but-curious and therefore we do not have to prove that any malicious action is detected as we did with the users.

Theorem 10. *The aggregator is able to correctly retrieve the sum of the coefficients of all users, assuming that all users passed the verification phase.*

Proof. In order to decrypt a ciphertext, according to the Paillier cryptosystem, a ciphertext must have the form $g_A^m \cdot r^{n_A}$. If the product is taken over all coefficient ciphertexts of all users, the secret shares cancel out and the result is $g_A^{\alpha_t} \cdot (h_A^u)^{n_A}$, where α_t is the sum of all the coefficients at round t. If we replace α_t with m and h_A^u with r, we see that this is in the correct form for the aggregator to decrypt. The aggregator is now able to retrieve the sum of the coefficients of all users correctly. □

Theorem 11. *The aggregator with polynomial bounded computation power does not learn any information of the private data of a user except for the distribution of the data of all users assuming that the decisional composite residuosity assumption holds.*

Proof. As we have proven in Lemma 6, the zero-knowledge proof of the set membership does not leak any information neither to a user nor to the aggregator. The same holds for the zero-knowledge proof of the plaintext equality which is proven by [6]. In the remainder of the verification of a secret share the aggregator only receives ciphertexts which are encrypted as defined by the Paillier cryptosystem. These ciphertexts, therefore, does not leak information to a party with polynomial bounded computation power assuming that the decisional composite residuosity assumption holds. The only ciphertexts in this process which the aggregator is able to decrypt are $E_A(n_A)$ and $E_A(s_2)$ which both are public information anyway.

We are left with proving that the aggregator does not learn any information from the encrypted coefficients. According to the Paillier cryptosystem, a ciphertext must have the form $g_A^m \cdot r^{n_A}$ to be decrypted. An encrypted coefficient cannot be rewritten in this form without having the knowledge of either the coefficient or the secret share. The aggregator does not know any random number that is part of a secret share and therefore the secret share is completely random as is the coefficient. Therefore, the aggregator cannot decrypt the ciphertext. The ciphertext is uniformly distributed in the ciphertext space and consequently, the aggregator cannot gain any knowledge about the underlying coefficient.

The aggregator can decrypt the product of all ciphertexts as proved in Theorem 10 which results in the distribution of the coefficients of all users. The aggregator is not able to link a value in the distribution to a specific user. Therefore, the aggregator with polynomial bound computing power cannot gain any information except for the distribution of the coefficients of all users assuming that the decisional composite residuosity assumption holds. □

6.5.3. Colluding users and aggregator

As a last part we prove that, when there are at least two honest users, all other colluding users and aggregator cannot gain any information from the honest users.

Theorem 12. *Assuming there are at least two honest users, a set of colluding users which are colluding with the aggregator cannot gain any information of the private data of an honest user, except for the distribution of the data of all honest users assuming that the decisional composite residuosity assumption holds.*

Proof. When there are two honest users i and j, those users share two random numbers $r_{i,j}$ and $r_{j,i}$ which are unknown to the other users and the aggregator. During transmission, the random numbers are encrypted with Paillier. These ciphertexts do not leak any information assuming that the decisional composite residuosity assumption holds. The secret share of an honest user therefore contains two random numbers which are completely random to the colluding users and the aggregator. Therefore, the secret share is random and so is the ciphertext $c_{i,t}$ of an honest user i at round t which is based on the secret share. The data for the zero-knowledge proofs of both the set membership and the plaintext equality also do not leak any information as proved in Lemma 6 and [6] respectively.

The colluding users and aggregator can only compute that product of all ciphertexts which the aggregator is able to decrypt. When the colluding users share their private data, the distribution of the values of the honest users retrieved. Therefore colluding users which are colluding with the aggregator cannot gain any information of the private data of an honest user when there are at least two honest users except for the distribution of the data of all honest users assuming that the decisional composite residuosity assumption holds. □

6.6. Complexity

In this section, we first discuss the computation complexity followed by a discussion about the experimental run-time. Finally, the communication complexity of our protocol is discussed.

6.6.1. Computation complexity

Table 6.1 gives an overview of the computation complexity of our protocol and of Viejo et al. [65]. We only compare our protocol to that of Viejo et al., because this is, to the best of our knowledge, the only protocol providing both multi-functionality and malicious user detection. The comparison to other schemes therefore does not provide any useful insights. We can see that the computation complexity of the aggregator of Viejo et al. depends on the amount of malicious users y since they check which user is malicious after receiving an invalid aggregate. The aggregator has to go through a different path in the tree of nodes for every malicious user. The length of such a path is $\log_d(u)$, where d is the amount of children of each node in the tree and u is the total amount of users. For each node on this path, the ciphertext of every possible value must be verified.

Our scheme, on the other hand, detects malicious users already as part of the submission which removes the reliance on the amount of malicious users y. The most computational expensive task of the verification of our scheme is the computation of $E_i(s_2)$ for every user i. The computation of one such value relies on $O(k)$ random numbers. For all users this results in $O(u \cdot k)$ operations. Additionally, every random number shared between two users must be verified resulting in a total of $O(u \cdot k)$ proof verifications.

During initialization, which is only executed once, the computational complexity of our protocol largely depends on the initialization of the encoding scheme. In Table 6.1, the encoding scheme with extended range is used. Note that the encoding scheme with maximum certainty is more efficient. When it is impossible to create a signature of a coefficient due to the lack of a modular inverse, all subsequent coefficients have to be updated. However, when the existence of a modular inverse is checked before creating the subsequent coefficients, only the current coefficient is updated. During creation of the subsequent coefficients, the updated current coefficient is taken into account.

The modular inverse modulo $n = p \cdot q$ does not exist only when $\alpha_i + x$ is not coprime to n. When this occurs, there are three possibilities, namely $\alpha_i + x$ is not coprime to p, q or both. In the latter case, $(\alpha_i + 1) + x$ is coprime to n and therefore has a modular inverse modulo n. If, on the other hand, $\alpha_i + x$ is not coprime to p, but it is to q, $(\alpha_i + 1) + x$ is coprime to p. However, it is possible that $(\alpha_i + 1) + x$ is not coprime to q anymore. Therefore, $\alpha_i + 1$ is incremented again with one. Assuming that p is not equal to 2, $(\alpha_i + 2) + x$ is coprime to both p and q and therefore also to n. The same reasoning holds when $\alpha_i + x$ is not coprime to q. Every coefficient, therefore, is incremented at most two times, which does not increment the computation complexity with an order of magnitude.

The initialization for the users in the scheme of Viejo et al. require no computation, where our scheme only requires some constant number of operations in order to initialize the Paillier cryptosystem. During a round, the computation complexity of the users of the scheme of Viejo et al. depends on m, where our scheme depends on k. The advantage of depending on k is the freedom of choosing such value, where m is fixed in most applications. The value k can be decreased resulting in better complexity, but less users need to collude in order to gain knowledge of an honest user. Vice versa the same holds.

Table 6.1: Computation complexity of our own scheme with the encoding scheme with extended range and the scheme of Viejo et al. [65], where d means the degree of the tree used in [65], y means the amount of malicious users and k the group size in which users share random numbers.

	Aggregator				User			
	Initialization		Every round		Initialization		Every round	
	[65]	Ours	[65]	Ours	[65]	Ours	[65]	Own
Exponentiation	$O(u)$	$O(u \cdot m)$	$O(m + ym \cdot log_d(u))$	$O(uk)$	-	$O(1)$	$O(m)$	$O(k)$
Addition	-	$O(u \cdot m)$	$O(m + u + yd \cdot log_d(u))$	-	-	-	-	$O(k)$
Multiplication	$O(u)$	$O(u^2 + m)$	$O(um + ydm \cdot log_d(u))$	$O(uk)$	-	$O(1)$	$O(1)$	$O(k)$
Modular inverse	-	$O(m)$	-	$O(u)$	-	$O(1)$	-	$O(1)$
Subtraction	$O(1)$	$O(u \cdot m)$	$O(m + ym \cdot log_d(u))$	$O(m)$	-	$O(1)$	-	$O(1)$
Division	-	$O(u)$	$O(m + ym \cdot log_d(u))$	$O(u + m)$	-	$O(1)$	-	-
Modulo	$O(1)$	$O(m)$	$O(m + ym \cdot log_d(u))$	$O(uk + m)$	-	$O(1)$	-	$O(k)$
Hash	-	-	$O(u)$	$O(uk)$	-	-	$O(m)$	$O(k)$
Equality	-	-	$O(m + ym \cdot log_d(u))$	$O(u)$	-	-	-	-

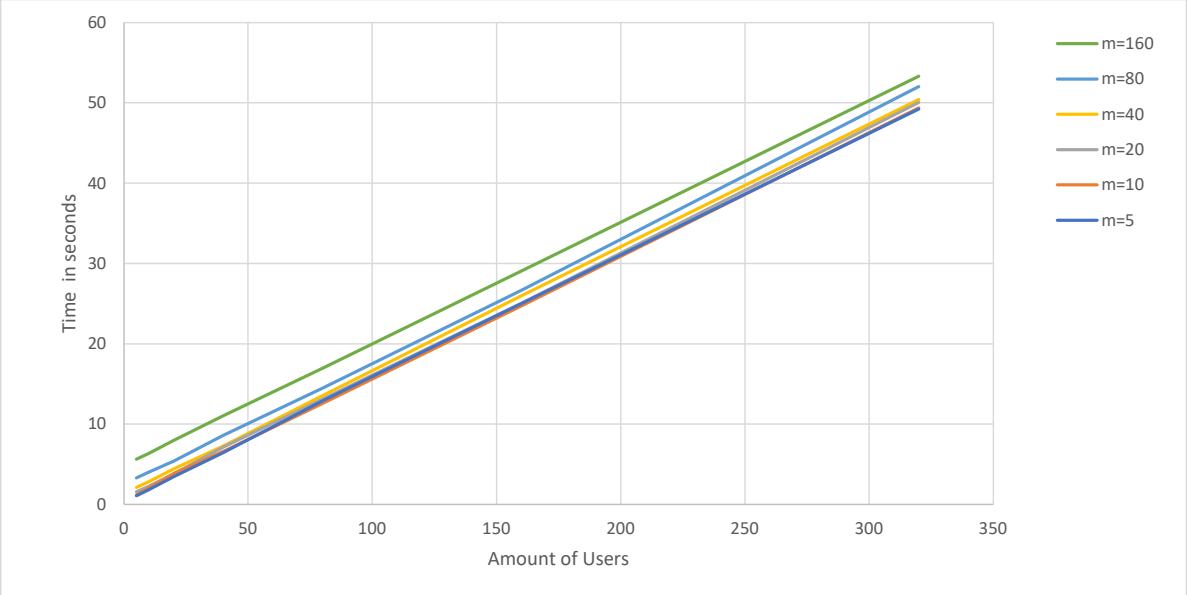

Figure 6.1: This figure illustrates the relation between the amount of users participating in the protocol and the time it takes to execute the initialization of our protocol based on the encoding scheme with maximum certainty. The time is taken in seconds and is the average of 10 experiments. Different lines in the graph represent different amounts of values.

6.6.2. Experimental run-time

We performed two types of experiments to measure the run-time of our protocol. The first type of experiments are used to measure the run-time of the initialization and the second type of experiments are used to measure the run-time of a round after initialization. All experiments are executed with an Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz processor and with 16GB RAM memory on Windows 10 Home. The experiments are implemented in Java 14 with built-in functions and are executed 10 times of which the average is taken.

Figure 6.1 illustrates the results of the first type of experiments, namely the run-time of the initialization of our protocol. During these experiments, we used the encoding scheme with maximum certainty for initializing the coefficients. The graph also plots different lines for different amounts of values.

The graph shows us that the run-time grows linearly with an increase of the amount of users participating in the protocol. The amount of values that are used does not make a significant difference. The difference between 5 and 160 values is between 4 and 5 seconds. Additional experiments showed that a possible group size does not affect the run-time of the initialization.

Figure 6.2 illustrates the results of the second type of experiments. The graph shows the dependence of the amount of users participating in the protocol on the run-time of a round after the initialization. In those experiments, it does not matter which encoding scheme is used. The only computational difference between

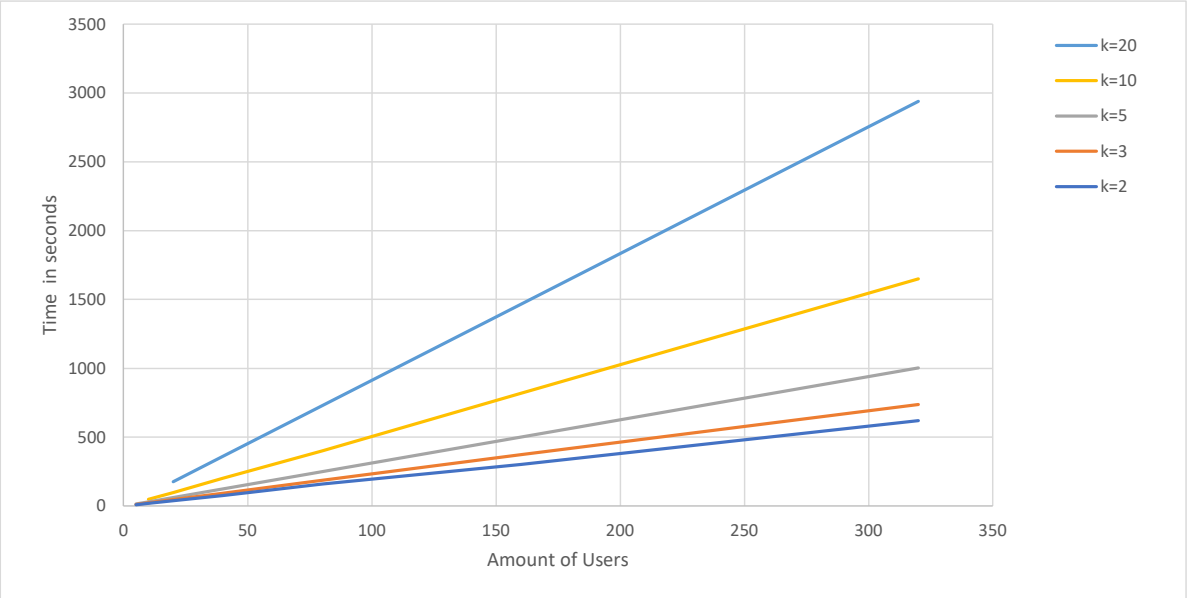

Figure 6.2: This figure illustrates the relation between the amount of users participating in the protocol and the time it takes to execute a round of our protocol after initialization. The time is taken in seconds and is the average of 10 experiments. Different lines in the graph represent different group sizes.

the encoding schemes is during the initialization. The experiments are done for different group sizes. Note that the group size is not allowed to be higher than the amount of users participating in the protocol. Therefore, not all lines start at the same x-coordinate.

The graph shows us that, also in a round after initialization, the run-time grows linearly with the amount of users participating in the protocol. The run-time also grows linearly with the group size. It is important to note that in these experiments, the computations of the users are performed in sequential order. In practice, when every user has a separate device, computations are performed in parallel which decreases the run-time. Additional experiments showed that the amount of values used in the protocol does not influence the run-time of a round after initialization. This is due to the fact that the amount of values is limited because of the encoding scheme.

6.6.3. Communication complexity

For the communication complexity, we use b as the amount of bits needed for n. We assume that the users are split in groups of size k. A Paillier ciphertext is $2b$ bits. Every user sends 2 ciphertexts with a random number for a round for every other user in the same group. The aggregator relays one of these ciphertexts. In total, this are $3(k-1)u$ ciphertexts of $2b$ bits. The zero-knowledge proof of plain text equality requires a user i to additionally send four messages of $2b$ bits, and one message of b bits, for every other user in the same group. In total, this results in $15u \cdot b \cdot (k-1)$ bits.

For the zero knowledge proof of the set membership, each user sends h, s_1, s_2 and s_3 which are b bits and B which is $2b$ bits. Also, in order to verify the secret shares, each user i sends $E_i(n_A)$, $E_A(n_A)$, $E_i(s_2)$, $E_A(s_2)$ and two proof of plaintext equality resulting in $12u$ messages of $2b$ bits and $2u$ messages of b bits.

In total, including the $2b$ bits ciphertexts $c_{i,t}$ for each user i at round t, each round of aggregation requires $2b \cdot (15u + 7(k-1)u) + b \cdot (6u + (k-1)u) = bu \cdot (21 + 15k) = O(buk)$ bits.

During initialization, every user receives lists of m values, m coefficients and m signatures. Each signature is $2b$ bits, each coefficient must be smaller than n and therefore requires at most b bits and the values depend on the application. We assume these lists to be pre-loaded in the devices of the users. Additionally, every user sends its public key during initialization to the aggregator which sends it to all other users.

All communication is summarized in Figure 6.3.

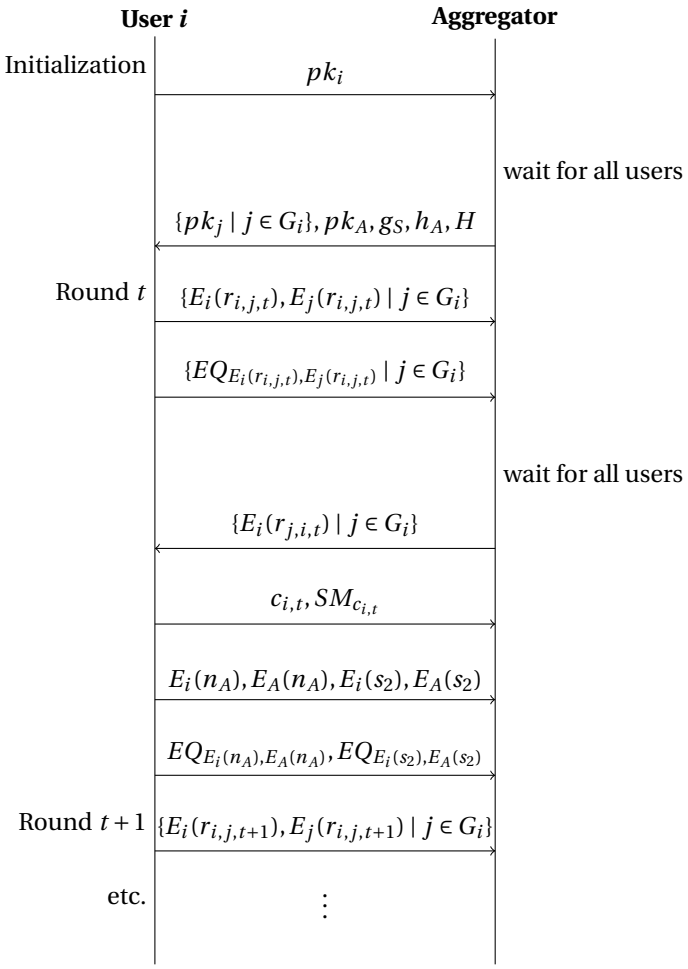

Figure 6.3: An overview of the communication in our protocol where we denote G_i as the set of all other users that are in the same group as user i. We also denote the proof of plaintext equality between two ciphertexts c_i and c_j as EQ_{c_i,c_j} and the proof of set membership of ciphertext c as SM_c.

6.7. Comparison

We compare our protocol with the scheme presented by Viejo et al. [65]. This scheme is chosen since it supports both the computation of any arbitrary function and malicious user detection. Other schemes, to the best of our knowledge, only support one of these which makes comparison impractical.

As we have already seen during the computation complexity analysis, the scheme of Viejo et al. differs from our scheme since it detects malicious users after aggregating instead of before. The aggregator of the scheme of Viejo et al. therefore has to perform extra computations dependent on the amount of malicious users. Additionally, where the computation complexity of a user of Viejo et al. depends on m which is in most applications fixed, our scheme depends on k which can be chosen and adapted to the context. However, when k cannot be set smaller than m due to security reasons, the scheme of Viejo et al. is more efficient with regard to the computation complexity. The same holds for the communication complexity. Our scheme requires $O(buk)$ bits, where the scheme of Viejo et al. requires $O(log_2(u) \cdot um)$ bits.

The main difference is with regard to the security. The aggregator in the scheme of Viejo et al. is able to decrypt the message of an individual user. Therefore, the aggregator is able to see what value the user has sent if it intercepts the message sent by a user. Our protocol makes sure that only the combination of all ciphertexts of a group is decryptable by the aggregator.

Secondly, the scheme of Viejo et al. requires all users to be in a tree structure where the leafs are users. This setup requires $O(u)$ extra nodes to function as intermediary nodes. Furthermore, these nodes can collude with the aggregator to relay the ciphertext of a user which the aggregator can decrypt. Our protocol does not rely on intermediary nodes and is therefore not susceptible to collusion between intermediary nodes and the aggregator.

Above all, the malicious user detection phase of Viejo et al. requires the aggregator to check the branches of the tree with a malformed ciphertext up to the leaves, which gives the aggregator the access to individual ciphertexts and therefore autonomy to decrypt such an individual ciphertext.

7

Discussion and Future Work

Data is used in many applications to make things easier for their users. Applications may improve the life of individuals or even save lives. Despite the advantages of data, it can also be misused, since data can be privacy-sensitive. When the privacy of this data is not guaranteed, the consequences can be enormous. Privacy-enhancing technologies are designed to both preserve the privacy of privacy-sensitive data and to give the applications their utility. One of those privacy-enhancing technologies is privacy-preserving data aggregation which preserves the data privacy while giving an application the opportunity to aggregate the privacy-sensitive data.

Every application has different circumstances it must operate in. An application might, for example, have a different computation it wants to perform on the private data in comparison to other applications. A desired characteristic of a privacy-preserving data aggregation scheme is, therefore, multi-functionality. Applications may run in an environment where the users are assumed honest-but-curious. Other applications, on the other hand, do not run in such an environment. It is therefore desired for a privacy-preserving data aggregation scheme to allow an application to use an honest-but-curious version, or a malicious user version where every message of each user is verified to be correct. Our research question is whether it is possible to design a privacy-preserving data aggregation scheme which is multi-functional, collusion-resistant and is able to detect malicious users without relying on a trusted authority. We presented and analyzed three protocols where one of those protocols satisfies the criteria of the research question. This chapter discusses our results, future work and concludes with concluding remarks.

7.1. Discussion

In this thesis we presented three protocols. We discuss each protocol separately. We start with the discussion of the encoding scheme followed by the two privacy-preserving data aggregation schemes.

7.1.1. Encoding scheme

The first protocol we presented is the encoding scheme. The encoding has two versions, one which maximizes the certainty of a correct result and one which extends the supported range of values. The encoding scheme with maximum certainty guarantees that every valid encoding is decoded correctly. However, it only supports a limited amount of values which might be impractical for certain applications. This amount of values depends on the message size and the amount of users that are participating in the protocol. The encoding scheme with extended range partially mitigates this problem by, as the name suggests, extending the range. In order to use this scheme, the probability distribution of all coefficients to be sent by a user must be known.

Both encoding schemes allow any data type to be used as values such as strings, intervals and numbers. Also, the encoding schemes are very flexible. They are both compatible with any privacy-preserving data aggregation scheme, which uses an additive homomorphic encryption scheme, to make it multi-functional.

When the probability distribution is not known beforehand and the amount of values supported by the encoding scheme is not enough, there are three possible solutions with which the encoding scheme can still be used. The first solution is by giving up on the exactness of the encoding scheme and use intervals as values. Now users send a coefficient corresponding to the interval their value is part of. The approximation error of the result depends on the size of the intervals. Note that intervals do not have to have the same size. Intervals

of values which are chosen less often may, for example, be bigger than intervals of values which are chosen more often.

Another solution is approximating the probability distribution with the help of a few rounds of the encoding scheme with maximum certainty with intervals as values instead of single values as in the previous solution. From the frequencies of users that have sent a certain interval, a probability distribution is approximated. This process may be extended to extra rounds of sub-intervals for a more detailed probability distribution. The risk with this solution is that the probability distribution is approximated and therefore it is more likely to happen that an unexpected amount of users send a specific value causing the decoding process to return incorrect results.

The last solution requires extra communication. The encoding scheme with maximum certainty is used multiple rounds where it encodes a different range of values in every round. In each round, a coefficient corresponds to a single values to preserve the exactness. The number of rounds therefore depend on the amount of values supported by the scheme and the amount of values the application requires to be supported. When a user has a value which is not in the range of a certain round, it sends the value zero as coefficient. The combination of all rounds result in an exact result for the entire range of values.

Therefore, despite the limitations of the encoding schemes, there are solutions so that the encoding schemes are still useful. In combination with the flexibility of privacy-preserving data aggregation schemes it can be used in and the support of different data types, the encoding schemes are suited for numerous real-world applications.

7.1.2. Multi-functional privacy-preserving data aggregation scheme

The first multi-functional privacy-preserving data aggregation scheme we presented assumed users to be honest-but-curious. Due to this user assumption, the simplest encryption scheme could be used as long as it is additive homomorphic in order to use the encoding scheme and as long as the privacy of the data is preserved. For this reason, the encryption and decryption only requires additions and subtractions. Except for the initialization, the computation complexity of our scheme is therefore better compared to comparable schemes. Also the communication complexity of our scheme is better compared to the state-of-the-art.

The second multi-functional privacy-preserving data aggregation scheme we presented removed the assumption of honest-but-curious users. The removal of this assumption required to include malicious user detection. Due to this extra requirement, the simple encryption and decryption used in the scheme with honest-but-curious users is not enough anymore. In this malicious user version, we have to check whether the coefficient of a user is a valid one without knowing the coefficient. Since there is only a limited set of coefficients, not every commitment scheme or zero-knowledge proof is suitable. Brute-forcing all possible coefficients is an efficient attack due to the limited size of the set. Therefore, checking whether the coefficient is correct is seen as a set membership verification. With this type of verification we check whether the coefficient is in the set of valid coefficients without leaking any information of the coefficient. The zero-knowledge proof we use for this purpose relies on the Paillier cryptosystem which is therefore used as an cryptosystem in this protocol. The same holds for the Boneh-Boyen signature scheme which is used to create a signature for every coefficient.

The scheme with malicious users improves upon the state-of-the-art with respect to communication and computation complexity. The main difference in the complexity is two-fold. The computation complexity of the aggregator in our protocol does not depend on the amount of malicious users because our protocol verifies a ciphertext during submission. State-of-the-art verifies ciphertexts after retrieving an incorrect aggregate which results in a computation complexity dependent on the amount of malicious users. Concerning the computation complexity for users and the communication complexity, state-of-the-art relies on the amount of values which is in most application fixed. Our scheme, on the other hand, relies on the amount of users in a group which is more flexible. The amount of users in a group can be set depending on the requirements of the application.

In both privacy-preserving data aggregation schemes, we split the users in groups of size k. However, attention should be paid when determining the value of k. It might seem logical to set k very low. The computation complexity for the aggregator would become linear in the amount of users and the communication would become linear in the amount of users and the amount of bits of a message. However, having a low value for k also has its consequences. As we have seen during the security analysis, the only way in order to retrieve the private value of an honest user is for $k-1$ users of the same group to collude. Also, the distribution of the honest users can be retrieved. For a small amount of honest users, an adversary knows that the value of an honest user is one of the little amount of values. Therefore, it is wanted to have as many honest users

as possible. Setting a k at a value which is too small results in the fact that it is easier for an adversary to compromise enough users. Therefore, depending on the application, a balance has to be found between the security and the computation- and communication complexity of the protocol.

In addition, both privacy-preserving data aggregation schemes do not rely on a trusted authority which is a requirement of the research goal.

7.2. Future work

Although our protocol is able to be used in a real-world application, applications might require characteristics that our protocol cannot provide. Inherent to the encoding scheme, which is used by our protocol, only a limited amount of values for a user to choose from is supported. Applications might need a bigger range of values. Besides that, our protocol is not fault-tolerant. When a user is faulty due to any problem such as power shortage or a lost network connection, the aggregator is not able to retrieve any data of other users in the same group as the faulty user. Our protocol, therefore, may need adjustments. Adjustments we see viable for future work are as follows.

- **Fault-tolerance** - Our protocol does not allow any node to be faulty. Users might encounter problems such as power shortage or a lost network connection. If such a faulty user did not share random numbers with other users, the protocol executes correctly, since the secret shares of the remaining users cancel each other. However, if a faulty user did share random numbers with other users and became faulty after that, the protocol executes incorrectly. The random numbers contained in the secret share of the faulty user are needed to cancel the secret shares of the remaining users. Only one faulty user, therefore, leads to an incorrect result.

  This problem is partially mitigated by splitting the users in groups. If one user is faulty, the result of the group containing the faulty user cannot be retrieved. The results of all the other groups are still retrieved correctly. The problem is only solved partially, because the results of non-faulty users in a faulty group are still lost. The smaller the group size the more data is retrieved from non-faulty users. However, as discussed in the previous section, decreasing the group size is not without consequences.

  An adjustment to make our protocol fault-tolerant must result in a protocol which supports any user to be faulty while having the possibility to retrieve the data of all other non-faulty users.

- **Less communication and computation complexity** - A decrease in communication and computation complexity without affecting the other properties of the protocol is always a beneficial adjustment. This can be achieved in any way. One way, in which we see potential, are homomorphic seeds. In our protocol every user must compute and send a ciphertext of a random number for every other user in the same group which dominates the complexity analysis for as well the communication as the computation. The protocol without malicious users in Chapter 5 circumvented the communication and computations by sending a random seed of a pseudo random number generator during the initialization of the protocol. Each user computed the shared random numbers on its own without needing any encryption or communication.

  In our protocol with malicious users in Chapter 6, the users can no longer simply share a random seed with other users. Due to the fact that a malicious user might include the wrong secret share in its ciphertext, the aggregator has to verify the secret share of every user in every round. In order to do this, the aggregator must possess the encryption of each random number shared between users. If users send a seed during the initialization, the aggregator must be able to compute the encryption of the random number of each round from that encrypted seed. This is, to the best of our knowledge, an open problem. If it is solved and applied to our protocol the communication and computation complexity could be reduced with a factor k.

- **Correlation** - In our protocol a user is able to send one value out of a list of values which is aggregated by the aggregator. Each value a user sends is treated separately. The aggregator is not able to retrieve the correlation between two or more different values a user might have, such as the correlation between the temperature and the location of a user.

  The correlation can also be used to extend the range of values which the protocol supports. A user could, for example, send a message for every digit. The aggregator, then, correlates all digits to form the resulting value without gaining the ability to link the value to any user.

7.3. Concluding remarks

Existing literature about privacy-preserving data aggregation schemes either focuses on the support of multi-functionality, the support of malicious user detection or neither of them. There is one exception which is not collusion-resistant. Their complexity also depends on the amount of values a user can have and the amount of malicious users. Our privacy-preserving data aggregation scheme also supports both multi-functionality and malicious user detection. In addition, our scheme is collusion-resistant. The complexity of our scheme depends on the group size instead of the amount of values a user can have and it does not depend on the amount of malicious users. The advantage of the group size over the amount of values is that the group size can be adapted easily to the context. The number of values, on the other hand, is most often fixed and cannot be changed. Due to the characteristics of our protocol, it brings the deployment of privacy-preserving data aggregation schemes in the real-world one step closer. With our protocol, applications no longer have to choose between multi-functionality, malicious user detection, collusion-resistancy and the privacy of the private data. They are all combined in one protocol.

Bibliography

[1] General data protection regulation (gdpr). https://gdpr.eu/tag/gdpr/, May 2018. Accessed: 02-12-2020.

[2] ANP. 'nog te veel verkeer om aantal besmettingen terug te dringen'. https://www.noordhollandsdagblad.nl/cnt/dmf20201008_22668936, Oct 2020. Accessed: 22-12-2020.

[3] Ghada Arfaoui, Jean-François Lalande, Jacques Traoré, Nicolas Desmoulins, Pascal Berthomé, and Saïd Gharout. A practical set-membership proof for privacy-preserving nfc mobile ticketing. *Proceedings on Privacy Enhancing Technologies*, 2015(2):25–45, 2015.

[4] Frederik Armknecht, Stefan Katzenbeisser, and Andreas Peter. Group homomorphic encryption: characterizations, impossibility results, and applications. *Designs, codes and cryptography*, 67(2):209–232, 2013.

[5] James Barron. The blackout of 2003: The overview; power surge blacks out northeast, hitting cities in 8 states and canada; midday shutdowns disrupt millions. https://www.nytimes.com/2003/08/15/nyregion/blackout-2003-overview-power-surge-blacks-northeast-hitting-cities-8-states.html, Aug 2003. Accessed: 22-12-2020.

[6] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 274–283, 2001.

[7] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269, 2020.

[8] Tiziano Bianchi, Thijs Veugen, Alessandro Piva, and Mauro Barni. Processing in the encrypted domain using a composite signal representation: pros and cons. In *2009 First IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 176–180. IEEE, 2009.

[9] Jens-Matthias Bohli, Christoph Sorge, and Osman Ugus. A privacy model for smart metering. In *2010 IEEE International Conference on Communications Workshops*, pages 1–5. IEEE, 2010.

[10] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *International conference on the theory and applications of cryptographic techniques*, pages 56–73. Springer, 2004.

[11] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of cryptography conference*, pages 325–341. Springer, 2005.

[12] Dan Boneh, Karl Rubin, and Alice Silverberg. Finding composite order ordinary elliptic curves using the cocks–pinch method. *Journal of Number Theory*, 131(5):832–841, 2011.

[13] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption. *ACM Computing Surveys (CSUR)*, 47(2):1–51, 2014.

[14] Shi-Cho Cha, Tzu-Yang Hsu, Yang Xiang, and Kuo-Hui Yeh. Privacy enhancing technologies in the internet of things: Perspectives and challenges. *IEEE Internet of Things Journal*, 6(2):2159–2187, 2018.

[15] Pyrros Chaidos and Jens Groth. Making sigma-protocols non-interactive without random oracles. In *IACR International Workshop on Public Key Cryptography*, pages 650–670. Springer, 2015.

[16] T-H Hubert Chan, Elaine Shi, and Dawn Song. Privacy-preserving stream aggregation with fault tolerance. In *International Conference on Financial Cryptography and Data Security*, pages 200–214. Springer, 2012.

[17] Chien-Ming Chen, Yue-Hsun Lin, Ya-Ching Lin, and Hung-Min Sun. Rcda: Recoverable concealed data aggregation for data integrity in wireless sensor networks. *IEEE Transactions on parallel and distributed systems*, 23(4):727–734, 2011.

[18] Le Chen, Rongxing Lu, Zhenfu Cao, Khalid AlHarbi, and Xiaodong Lin. Muda: Multifunctional data aggregation in privacy-preserving smart grid communications. *Peer-to-peer networking and applications*, 8(5):777–792, 2015.

[19] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015.

[20] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[21] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[22] Tassos Dimitriou and Mohamad Khattar Awad. Secure and scalable aggregation in the smart grid resilient against malicious entities. *Ad Hoc Networks*, 50:58–67, 2016.

[23] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[24] Costas Efthymiou and Georgios Kalogridis. Smart grid privacy via anonymization of smart metering data. In *2010 first IEEE international conference on smart grid communications*, pages 238–243. IEEE, 2010.

[25] Zekeriya Erkin and Gene Tsudik. Private computation of spatial and temporal power consumption with smart meters. In *International Conference on Applied Cryptography and Network Security*, pages 561–577. Springer, 2012.

[26] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. Smart grid—the new and improved power grid: A survey. *IEEE communications surveys & tutorials*, 14(4):944–980, 2011.

[27] 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*, pages 186–194. Springer, 1986.

[28] Sören Finster and Ingmar Baumgart. Smart-er: Peer-based privacy for smart metering. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 652–657. IEEE, 2014.

[29] Flavio D Garcia and Bart Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In *International Workshop on Security and Trust Management*, pages 226–238. Springer, 2010.

[30] Shanshan Ge, Peng Zeng, Rongxing Lu, and Kim-Kwang Raymond Choo. Fgda: Fine-grained data analysis in privacy-preserving smart grid communications. *Peer-to-Peer Networking and Applications*, 11(5): 966–978, 2018.

[31] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.

[32] Xuhui Gong, Qiang-Sheng Hua, Lixiang Qian, Dongxiao Yu, and Hai Jin. Communication-efficient and privacy-preserving data aggregation without trusted authority. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1250–1258. IEEE, 2018.

[33] Michael T Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 792–799. IEEE, 2011.

[34] Zhitao Guan, Yue Zhang, Longfei Wu, Jun Wu, Jing Li, Yinglong Ma, and Jingjing Hu. Appa: An anonymous and privacy preserving data aggregation scheme for fog-enhanced iot. *Journal of Network and Computer Applications*, 125:82–92, 2019.

[35] Song Han, Shuai Zhao, Qinghua Li, Chun-Hua Ju, and Wanlei Zhou. Ppm-hda: privacy-preserving and multifunctional health data aggregation with fault tolerance. *IEEE Transactions on Information Forensics and Security*, 11(9):1940–1955, 2015.

[36] Arne Holst. Amount of information globally 2010-2024. https://www.statista.com/statistics/871513/worldwide-data-created/, Dec 2020. Accessed: 21-12-2020.

[37] Arne Holst. Iot connected devices worldwide 2030. https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology/, Jan 2021. Accessed: 21-01-2021.

[38] Tobias Jeske. Privacy-preserving smart metering without a trusted-third-party. In *Proceedings of the International Conference on Security and Cryptography*, pages 114–123. IEEE, 2011.

[39] Maarten Keulemans. Onderzoek: lockdowns europa hebben ruim 3 miljoen sterfgevallen voorkomen. https://www.volkskrant.nl/nieuws-achtergrond/onderzoek-lockdowns-europa-hebben-ruim-3-miljoen-sterfgevallen-voorkomen~bce87b1f/, Jun 2020. Accessed: 17-03-2021.

[40] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 175–191. Springer, 2011.

[41] Xuefeng Liu, Yuqing Zhang, Boyang Wang, and Huaqun Wang. An anonymous data aggregation scheme for smart grid systems. *Security and communication networks*, 7(3):602–610, 2014.

[42] Rongxing Lu, Xiaohui Liang, Xu Li, Xiaodong Lin, and Xuemin Shen. Eppa: An efficient and privacy-preserving aggregation scheme for secure smart grid communications. *IEEE Transactions on Parallel and Distributed Systems*, 23(9):1621–1631, 2012.

[43] Tycho Malmberg. Filerijden ongezond? https://www.nemokennislink.nl/publicaties/filerijden-ongezond/, Jan 2005. Accessed: 14-04-2021.

[44] Eva Malten. What is the difference between personal data and privacy-sensitive information? https://www.zivver.com/blog/difference-between-personal-data-and-privacy-sensitive-information, Dec 2019. Accessed: 23-12-2020.

[45] Ben Meindertsma. Rivm: gedeeltelijke lockdown mogelijk tot in december nodig. https://nos.nl/artikel/2352645-rivm-gedeeltelijke-lockdown-mogelijk-tot-in-december-nodig.html, Oct 2020. Accessed: 17-03-2021.

[46] Andrés Molina-Markham, Prashant Shenoy, Kevin Fu, Emmanuel Cecchet, and David Irwin. Private memoirs of a smart meter. In *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building*, pages 61–66, 2010.

[47] Anup R Nimje, VT Gaikwad, and HN Datir. Attribute-based encryption techniques in cloud computing security: an overview. *Int. J. Comput. Trends Technol*, 4(3):419–422, 2013.

[48] University of Groningen. Sensitive data and medical confidentiality. https://www.futurelearn.com/info/courses/protecting-health-data/0/steps/39608, 2020. Accessed: 22-12-2020.

[49] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In *International conference on the theory and applications of cryptographic techniques*, pages 308–318. Springer, 1998.

[50] NOS op 3. 'supermarkten zijn verantwoordelijk voor zo'n 3 tot 5 procent van de verspilling'. https://nos.nl/op3/artikel/2037159-supermarkten-zijn-verantwoordelijk-voor-zo-n-3-tot-5-procent-van-de-verspilling.html, May 2015. Accessed: 17-03-2021.

[51] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999.

[52] Ronald Petrlic. A privacy-preserving concept for smart grids. *Sicherheit in vernetzten Systemen*, 18:B1–B14, 2010.

[53] Rick Plantinga and Kelly Adams. Samenwerking met duitse ziekenhuizen kan levens redden, maar makkelijk is het niet. https://www.tubantia.nl/enschede/samenwerking-met-duitse-ziekenhuizen-kan-levens-redden-maar-makkelijk-is-het-niet~ab081768/, Mar 2020. Accessed: 17-03-2021.

[54] Sushmita Ruj, Amiya Nayak, and Ivan Stojmenovic. A security architecture for data aggregation and access control in smart grids. *arXiv preprint arXiv:1111.2619*, 2011.

[55] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[56] Yun Shen and Siani Pearson. Privacy enhancing technologies: A review. *HP Laboratories*, 2739:1–30, 2011.

[57] Elaine Shi, TH Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Proc. NDSS*, volume 2, pages 1–17. Citeseer, 2011.

[58] Jing Shi, Rui Zhang, Yunzhong Liu, and Yanchao Zhang. Prisense: privacy-preserving data aggregation in people-centric urban sensing systems. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[59] Zhiguo Shi, Ruixue Sun, Rongxing Lu, Le Chen, Jiming Chen, and Xuemin Sherman Shen. Diverse grouping-based aggregation protocol with error detection for smart grid communications. *IEEE Transactions on Smart Grid*, 6(6):2856–2868, 2015.

[60] Skipr. De jonge: structueel naar 1700 ic-bedden kunnen opschalen. `https://www.skipr.nl/nieuws/de-jonge-structueel-naar-1700-ic-bedden-kunnen-opschalen/`, May 2020. Accessed: 17-03-2021.

[61] Slimhuis. Wat is een smart device. `https://slimhuis.tech/smart-home-kennisbank/wat-is-een-smart-device/`, Dec 2019. Accessed: 14-04-2021.

[62] Nigel P. Smart. *Cryptography Made Simple*. Springer, 2016.

[63] Ruixue Sun, Zhiguo Shi, Rongxing Lu, Min Lu, and Xuemin Shen. Aped: An efficient aggregation protocol with error detection for smart grid communications. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 432–437. IEEE, 2013.

[64] Ministerie van Infrastructuur en Waterstaat. Watersnoodramp 1953. `https://www.rijkswaterstaat.nl/water/waterbeheer/bescherming-tegen-het-water/watersnoodramp-1953`, Apr 2021. Accessed: 14-04-2021.

[65] Alexandre Viejo, Qianhong Wu, and Josep Domingo-Ferrer. Asymmetric homomorphisms for secure aggregation in heterogeneous scenarios. *Information Fusion*, 13(4):285–295, 2012.

[66] Shaowei Wang, Liusheng Huang, Pengzhan Wang, Yao Shen, Hongli Xu, and Wei Yang. Privacy preserving big histogram aggregation for spatial crowdsensing. In *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2015.

[67] Shaowei Wang, Liusheng Huang, Pengzhan Wang, Hou Deng, Hongli Xu, and Wei Yang. Private weighted histogram aggregation in crowdsourcing. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 250–261. Springer, 2016.

[68] Yang Wang. Privacy-enhancing technologies. In *Handbook of research on social and organizational liabilities in information security*, pages 203–227. IGI Global, 2009.

[69] Wilhelm Werner. Polynomial interpolation: Lagrange versus newton. *Mathematics of computation*, pages 205–217, 1984.

[70] Ye Yan, Yi Qian, Hamid Sharif, and David Tipper. A survey on cyber security for smart grid communications. *IEEE Communications Surveys & Tutorials*, 14(4):998–1010, 2012.

[71] Rui Zhang, Jing Shi, Yanchao Zhang, and Chi Zhang. Verifiable privacy-preserving aggregation in people-centric urban sensing systems. *IEEE Journal on Selected Areas in Communications*, 31(9):268–278, 2013.

[72] Yuan Zhang, Qingjun Chen, and Sheng Zhong. Privacy-preserving data aggregation in mobile phone sensing. *IEEE Transactions on Information Forensics and Security*, 11(5):980–992, 2016.