Performance and Verifiability of IoT Security Protocols.

Dissertation

submitted to

Sorbonne Université and EURECOM

in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Author:

Mohamad MANSOURI

Defended on April 6th 2023, in front of a committee composed of:

Reviewers

Prof.	**Ghassan KARAME**	Ruhr-Universität Bochum, Germany
Dr.	**Marc JOYE**	Zama, France

Examiners

Prof.	**Mauro CONTI**	University of Padua, Italy
Prof.	**Antonio FAONIO**	EURECOM, France
Dr.	**Ferhat KARAKOÇ**	Ericsson, Turkey

Industrial Supervisor

Dr.	**Wafa BEN JABALLAH**	Thales SIX GTS, France

Thesis Supervisor

Prof.	**Melek ÖNEN**	EURECOM, France

Performance et Vérifiabilité des Protocoles de Sécurité IoT.

Thèse

soumise à

Sorbonne Université et EURECOM

pour l'obtention du Grade de Docteur

présentée par:

Mohamad MANSOURI

Soutenue le 6 avril 2023 devant le jury composé de:

Reviewers

Prof.	**Ghassan KARAME**	Ruhr-Universität Bochum, Allemagne
Dr.	**Marc JOYE**	Zama, France

Examiners

Prof.	**Mauro CONTI**	University of Padua, Italie
Prof.	**Antonio FAONIO**	EURECOM, France
Dr.	**Ferhat KARKAOÇ**	Ericsson, Turquie

Industrial Supervisor

Dr.	**Wafa BEN JABALLAH**	Thales SIX GTS, France

Thesis Supervisor

Prof.	**Melek ÖNEN**	EURECOM, France

Abstract

The Internet of Things (IoT) is one of the most important technologies in our current world. It is composed of connected devices with sensors and processing abilities, all connected to a single platform that orchestrates them. The integration of these IoT devices into many real-life applications (eg., transportation, health-care, industries, ...) implied significant performance and efficiency improvements. As a consequence, we have seen a boom in the number of IoT devices deployed and their corresponding platforms. These IoT devices use real data from their deployment environment and send them to the platform. The collected data by these devices are often sensitive information. Hence, the privacy of users' data is one of the important concerns in IoT. Moreover, IoT applications rely on automating frequent tasks to achieve better efficiency. Unfortunately, moving control of usually human-controlled operations to the IoT risks the safety of IoT users .

This thesis deals with the privacy and safety concerns raised by IoT. We propose security protocols that preserve the privacy of the users' data. In addition to privacy, we aim to design verifiable solutions that guarantee the correctness of the computations performed by the IoT devices and the platform. We design this solution while focusing on their performance specifically for IoT. More precisely, we propose protocols that are scalable to cope with the increasing number of IoT devices. We also consider protocols that are fault-tolerant to cope with the mobility and frequent dropouts of IoT devices. We focus on two security protocols: Secure Aggregation and Remote Attestation.

Secure aggregation is a protocol where an aggregator computes the sum of the private inputs of a set of users. In this thesis, we propose the first verifiable secure aggregation protocol (VSA) that gives formal guarantees of security in the malicious model. Our solution preserves the privacy of all users' inputs and the correctness of the aggregation result. Moreover, we propose a novel fault-tolerant secure aggregation protocol (FTSA) based on additively-homomorphic encryption. The scheme allows users in secure aggregation to drop from the protocol and offers a mechanism to recover the aggregate without affecting the privacy of the data. We show that FTSA outperforms the state-of-the-art solutions in terms of scalability with respect to the number of users.

On the other hand, a remote attestation protocol is a protocol that allows an IoT device (acting as a prover) to prove its software integrity to the IoT platform (acting as the verifier). We propose a new collaborative remote attestation protocol (FADIA) in which devices collect attestations from each other and aggregate them. FADIA deals with the heterogeneity and dynamic nature of IoT by considering fairness in its design. The evaluation of FADIA shows an increase in the lifetime of a network.

Abrégé

L'internet des objets (IoT) est l'une des technologies les plus importantes de notre monde actuel. Il est composé d'appareils connectés dotés de capteurs et de capacités de traitement, tous reliés à une plateforme unique qui les orchestre. L'intégration de ces dispositifs IoT dans de nombreuses applications de la vie réelle (par exemple, le transport, les soins de santé, les industries, ...) a impliqué des améliorations significatives de la performance et de l'efficacité. En conséquence, nous avons assisté à un boom du nombre de dispositifs IoT déployés et de leurs plateformes correspondantes. Ces dispositifs IoT utilisent les données réelles de leur environnement de déploiement et les envoient à la plateforme. Les données collectées par ces dispositifs sont souvent des informations sensibles. Par conséquent, la confidentialité des données des utilisateurs est l'une des principales préoccupations de l'IoT. En outre, les applications IoT reposent sur l'automatisation de tâches fréquentes pour une meilleure efficacité. Malheureusement, le transfert du contrôle d'opérations habituellement contrôlées par l'homme vers l'IoT risque de compromettre la sécurité des utilisateurs de l'IoT.

Cette thèse traite des problèmes de confidentialité et de sécurité soulevés par l'IoT. Nous proposons des protocoles de sécurité qui préservent la confidentialité des données des utilisateurs. En plus de la confidentialité, nous voulons concevoir des solutions vérifiables qui garantissent l'exactitude des calculs effectués par les dispositifs IoT et la plateforme. Nous concevons ces solutions en nous concentrant sur leurs performances spécifiquement pour l'IoT. Plus précisément, nous proposons des protocoles qui sont évolutifs pour faire face au nombre croissant de dispositifs IoT. Nous considérons également des protocoles tolérants aux pannes pour faire face à la mobilité et aux abandons fréquents des dispositifs IoT. Nous nous concentrons sur deux protocoles de sécurité : l'agrégation sécurisée et l'attestation à distance.

L'agrégation sécurisée est un protocole où un agrégateur calcule la somme des entrées privées d'un ensemble d'utilisateurs. Dans cette thèse, nous proposons le premier protocole d'agrégation sécurisée vérifiable (VSA) qui donne des garanties formelles de sécurité dans le modèle malveillant. Notre solution préserve la confidentialité des entrées de tous les utilisateurs et l'exactitude du résultat de l'agrégation. En outre, nous proposons un nouveau protocole d'agrégation sécurisée tolérant aux pannes (FTSA) basé sur le cryptage additif-homomorphe. Le schéma permet aux utilisateurs de l'agrégation sécurisée de se retirer du protocole et offre un mécanisme pour récupérer l'agrégat sans affecter la confidentialité des données. Nous montrons que le FTSA surpasse les solutions de l'état de l'art en termes d'évolutivité par rapport au nombre d'utilisateurs.

D'autre part, un protocole d'attestation à distance est un protocole qui permet à un dispositif IoT (agissant en tant que prouveur) de prouver son intégrité logicielle à la plateforme IoT (agissant en tant que vérificateur). Nous proposons un nouveau protocole collaboratif d'attestation à distance (FADIA) dans lequel les dispositifs collectent des attestations les uns des autres et les agrègent. FADIA traite de l'hétérogénéité et de la nature dynamique de l'IoT en tenant compte de l'équité dans sa conception. L'évaluation de FADIA montre une augmentation de la durée de vie d'un réseau.

Acknowledgements

I would like to begin by expressing my profound gratitude to all those who have contributed to the completion of this PhD thesis. It is with great pleasure that I acknowledge the invaluable support, guidance, and encouragement I have received from numerous individuals and organizations throughout this journey.

I am immensely grateful to both of my supervisors who have provided invaluable guidance and support throughout this research journey. Melek Önen, my academic supervisor, has been instrumental in shaping the direction of this thesis and enhancing its quality. Wafa Ben Jaballah, my industrial supervisor, has provided valuable insights and practical expertise, bridging the gap between academia and industry. I am truly fortunate to have had the opportunity to work under their mentorship.

I extend my heartfelt appreciation to the members of my thesis committee who have significantly contributed to the success of this research. Ghassan Karame, Marc Joe, Mauro Conti, Ferhat Karakoç, and Aurélien Francillon provided invaluable feedback, constructive criticism, and rigorous evaluation. I am also grateful to Katerina Mitrokotsa and Sayantan Mukherjee, for their valuable insights and contributions to this research.

I would like to express my sincere gratitude to the French National Association for Research (ANTS) for funding this research. Furthermore, I am deeply thankful to EURECOM and Thales for hosting this thesis and providing the necessary resources and facilities to conduct this work. EURECOM provided an inspiring academic environment that nurtured intellectual growth, while Thales offered invaluable industry insights, collaborative opportunities, and technical expertise.

Within Thales, I had the privilege of being a part of the exceptional Therisis team, led by Olivier Bettan. I extend my deepest gratitude to the members of Therisis team for their unwavering support, fruitful discussions, and shared experiences, which greatly enriched my research journey.

I offer my deepest gratitude to my family for their exceptional support throughout this journey. Their love, encouragement, and belief in my abilities have been my greatest motivation. I would also like to extend a special thanks to my wife, Fatima Maali. During the long nights and demanding work, she stood by my side, providing unwavering support and understanding. I am deeply grateful for her enduring love and the sacrifices she made throughout this endeavor.

Lastly, I dedicate this work to the loving memory of my father, whose soul departed one week before my PhD defense day. His unwavering support, encouragement, and belief in my abilities were a constant source of inspiration. He dreamed of this moment and was my biggest advocate throughout my academic journey. Though he couldn't witness the culmination of my efforts, his presence will forever remain in my heart.

To the members of my thesis committee, collaborators, colleagues, my family, and the soul of my father, I offer my deepest appreciation. Your contributions, guidance, and support have played an integral role in the successful completion of this thesis. I am forever grateful for your presence in my academic journey.

Contents

List of Figures

List of Tables

List of Publications

We have presented the following publications during this PhD study. Most of them have already been presented through different venues enumerated below:

1. **Mohamad Mansouri**, Wafa Ben Jaballah, Melek Önen, Md Masoom Rabbani, and Mauro Conti. 2021. "FADIA: fairness-driven collaborative remote attestation". *In Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (WiSec '21). Abu Dhabi, UAE (Remote Conference), June 28 - July 2, 2021.

2. **Mohamad Mansouri**, Melek Önen, and Wafa Ben Jaballah. 2022. "Learning from Failures: Secure and Fault-Tolerant Aggregation for Federated Learning". *In Proceedings of the 38th Annual Computer Security Applications Conference* (ACSAC '22). Austin, Texas, USA, December 5 - December 9, 2022.

3. **Mohamad Mansouri**, Melek Önen, Wafa Ben Jaballah, and Mauro Conti. 2023. "SoK: Secure aggregation based on cryptographic schemes for federated Learning". *In Proceedings of the 23rd Privacy Enhancing Technologies Symposium* (PETS '23). Lausanne, Switzerland (Hybrid Conference), July 10 - July 14, 2023.

4. **Mohamad Mansouri**, Antonio Faonio, Melek Önen, Wafa Ben Jaballah, Sayantan Mukherjee, and Katerina Mitrokotsa. 2023. "Verifiable Secure aggregation in The Malicious Model". *Under Submission.*

Chapter 1

Introduction

1.1 IoT and its Applications

With the recent advances in embedded systems and communication technologies, the Internet of Things (IoT) paradigm emerge and evolves rapidly. Indeed, researchers expect that the number of IoT devices installed will raise to 41.6 billion devices that will generate 79.4 ZB of data in the year 2025 [Dig19]. IoT is composed of connected objects with sensors and processing abilities. The connected objects exchange data with each other and with external systems. IoT spans different life sectors such as medical care, end-consumers, industrial, and even military applications. The main benefit of IoT originates from its ability to automate daily tasks leading to improved efficiency, productivity, and accuracy of existing systems.

Many applications use IoT technologies to improve the quality of their services and to reduce their operational cost. We discuss the main applications where IoT made a significant contribution to their advancement.

Home Automation It may involve temperature and humidity sensors, live monitoring devices, electric appliances control, etc. Smart homes [Pro22] provide a platform for end consumers to collect data from their houses and automate actions. The main benefit of smart homes is offering more life convenience for end-consumers and cost-saving by optimizing energy consumption.

Healthcare The medical sector also benefits from IoT technologies to improve healthcare systems [PSJH+20]. It allows the collection of more rich medical data from patients which are used to improve the research work and studies on existing diseases [IHPS11]. Additionally, IoT can improve the live monitoring of patients leading to improved treatment of emergency cases.

Transportation Automated cars are the future of the world. This promising technology strongly depends on IoT sensors and devices that are installed inside vehicles and on the

roads [MTMH18]. These devices communicate with each other to provide autonomous driving features for passengers.

Industry 4.0 One of the main reasons behind the boom of IoT technologies is their use in industries [YSW18]. Industry 4.0 involves the deployment of smart sensors and IoT platforms in the production lines. The exchange of data between the product and the production line enables a much more efficient connection of the supply chain. Thus, it leads to a reduction in production costs and an improvement in product quality.

Infrastructures Large-scale applications that cover metropolitan projects have also a huge interest in IoT technologies. They involve smart cities [Poo18] where smart sensors enable services like parking search, environmental monitoring, water distribution, and traffic management. They also involve smart grids [SG17] that enable smart monitoring of electricity consumption in a large inhabited area.

1.2 IoT Components

The IoT involves two main components: the IoT end-devices (the sensors, actuators, etc.) and the IoT platform to which these end-devices are connected. We describe the role and the properties of each of these components.

1.2.1 IoT End-Devices

These are the devices installed at the peripheries of the IoT network. They collect live data from the environment and perform simple actions. An example of these devices is smart sensors, programmable logic units (PLC), cameras, electricity relays, etc. Although there exist many types of IoT end-devices, they all share some common characteristics:

- Deployed in large-scale: It is common in many IoT applications that the IoT end-devices are deployed in large numbers and spread across multiple geo-dispersed locations. For example, smart city applications install end-devices in different locations to track live information about the environment in multiple locations. This allows authorities to take better decisions that help improve the life quality in some regions.

- Low in Resource: Due to the need for large number of devices in some applications, the manufacturers have to decrease their costs to make them affordable. This comes at the cost of minimizing the resources that these devices possess such as their processing power, their data transfer speed, and their energy resources.

- Heterogeneity: The complexity of IoT applications requires deploying different types of IoT end-devices. These devices can have different properties in terms of their amount of resources and their software stack.

- Mobility: In many applications, the end-devices are supposed to move frequently within some areas. For example, in industrial applications, smart sensors may be connected to objects to track the production or business process.

1.2.2 IoT Platform

In addition to the IoT end-devices, IoT applications deploy an IoT platform to provide their servers. The IoT platform is the brain of the IoT which manages all the connected IoT end-devices. It involves the network architecture and the processing servers that collect data from the end-devices and analyze/process them. The IoT platform administrators control these servers. They define how the data is processed and what decisions and actions to take based on the data analysis. These IoT platforms may be deployed physically on-premises or might use cloud technologies and thus get installed as software services on the cloud.

1.3 The Dark Side of IoT

Along with the bright side of IoT comes a dark side. Security researchers raise concerns about the security of IoT devices and their impact on the privacy and safety of users [TMTQ20]. We identify three types of adversaries for IoT that raise serious security problems: Adversaries controlling some IoT devices, adversaries controlling the IoT platform, and external adversaries that are on the same network of the IoT devices. In this section, we present the different types of adversaries and their impact on the user's safety and privacy.

1.3.1 Adversaries Controlling The End-Devices

It is possible to have IoT end-devices performing attacks on the IoT application. This can happen either due to users of these devices acting maliciously or due to the devices being compromised. In such cases, the adversary aims to compromise the IoT application to influence its logic or to leak private information about other users. In these attacks, the adversary controls the inputs of the compromised devices. Hence, it can influence the results and decisions. An example of such attacks is the attack on the implantable cardiac devices from St. Jude Medical (2017) [Lar17]. The attacker compromised these implementable and threatened the safety of the users. Another example is the famous StuxNet Malware (2009) [Kus13]. This malware infected PLC devices at Iranian nuclear plant and caused damage to its components while remaining undetected by reporting false information about the actual performance. Therefore, to secure the IoT, one should consider the malicious behavior of IoT end-devices and thus verify their inputs.

1.3.2 Adversaries Controlling The Platform

The IoT platform collects users' data for further processing. However, users cannot trust the platform as this data often contains private information. IoT platform owners might be interested in increasing their profit by reselling these data to malicious entities. Indeed,

in 2009 the Dutch parliament rejected a smart metering program, basing their decision on privacy concerns [MRT12]. The smart metering program aims to collect live data from Dutch households about their electricity consumption. However, the Dutch parliament raised concerns about the misuse of this data by electricity companies. Therefore, to secure the IoT, one should consider the malicious behavior of the IoT platform.

1.3.3 External Adversaries

External attackers are connected to the IoT network. Hence, they can see the messages sent from the devices to the platform. This allows the adversary to tamper with these messages. Moreover, external attackers may exploit existing vulnerabilities on the IoT end-devices and thus take control of these end-devices. In addition, some external attackers may also have physical access to the IoT end-devices which allows them to perform hardware attacks on the devices. These types of adversaries are mainly interested in three attacks types:

- Leaking Sensitive Data: In this type of attack, the attacker exploits the lack of security mechanisms on IoT end-devices to collect private information from the user of that device. These attacks happen either due to insecure communication between the IoT end-devices and the IoT platform or due to software and hardware vulnerabilities that allow the attacker to leak private data.

- Privilege Escalation: External adversaries aim to get more privileges by compromising the end-devices. This allows the adversaries to perform more sophisticated attacks and gain more information as shown in Section 1.3.1.

- Creating Botnets: The adversaries can compromise a large number of IoT devices to create botnets. A botnet is a group of Internet-connected devices used to perform Distributed Denial-of-Service (DDoS) attacks, steal data, send spam, and allow the attacker to access the device and its connection. The owner can control the botnet using command and control (C&C) softwares [Sab16]. One of the most popular botnets that hit the IoT is "Mirai Botnet (2016)" [AAB+17]. It is composed primarily of embedded and IoT devices, and it took the Internet by storm in late 2016 when it overwhelmed several high-profile targets with some of the largest distributed denial-of-service (DDoS) attacks on record.

1.4 Problem Statement

It is clear how important IoT is to our world. It is even more clear that the trend of integrating more and more IoT devices will keep increasing. Therefore, one very important goal for researchers is securing these IoT devices to make sure that this integration is not a curse but rather a benefit for humanity. In this thesis, we aim for designing security protocols for IoT devices. We first start by posing the question: *Are existing protocols designed for standard IT devices (e.g., computers) can solve the security concerns of IoT?* The direct answer to this question is No. Researchers have already identified the

limitation of IT world security solutions and traced back this limitation to the unique characteristics of IoT devices (see Section 1.2.1). Therefore, in this thesis, we aim to design customized security protocols that take into consideration all the constraints of the IoT ecosystem. We specifically study the security protocols on three main axes: scalability, fault tolerance, and verifiability.

- **Scalability**: We aim to design scalable security protocols that involve thousands or possibly millions of devices. This allows for more practical solutions targeting IoT applications that run with a large number of IoT devices.
- **Fault Tolerance**: We aim to design security protocols that are reliable in case of failures of some devices. This allows for more practical solutions targeting IoT applications with mobile devices that may encounter connectivity problems and occasionally fail to receive and deliver messages.
- **Verifiability**: We aim to design security protocols that do not rely on trusting the different IoT components (processing platform, end-devices, etc.). By integrating verification mechanisms into our design, we can verify that third-party IoT platforms are correctly processing the collected data. Additionally, verifiable security protocols can also ensure that compromised devices cannot affect the results of our protocol without being detected.

1.4.1 Security Protocols for IoT

In this thesis, we focus on two very important security protocols that can be game-changers for improving IoT security. Namely, secure aggregation protocols and remote attestation protocols.

1.4.1.1 Secure Aggregation Protocols

Secure aggregation consists of computing the sum of data collected from multiple sources without disclosing these individual inputs. Hence, the goal of a secure aggregation protocol is to preserve the privacy of the data sources. Researchers identified that such protocols are useful to improve privacy guarantees in IoT applications. More precisely, secure aggregation can be used at the network layer to secure the data exchange from the end-devices to the platform. Smart grid applications are a good example. Moreover, secure aggregation can also be used at the processing layer to allow multiple IoT platforms to aggregate securely their collected data. For example, secure aggregation can be applied to IoT platforms that are using federated learning. Therefore, we classify secure aggregation into the following two categories.

- *Intra-Platform Secure Aggregation*: Smart grids are IoT end-devices that monitor energy consumption in households and industries. These devices collect live measurements of energy consumption and send them to service providers. The service providers compute statistics on the collected data thus enabling them to optimize their energy supply and improve their business model. Collecting these data from

households poses a privacy problem. It allows the service providers to perform real-time surveillance and profile the tenants by determining their behavior patterns. Service providers may also determine the used electric appliances in households and track the behavior of renters/leasers.

Secure aggregation can be used to protect the energy consumption measurements of each household and allows the service provider to only compute their sum. By only disclosing the statistical average value of a large inhabited area, the privacy of inhabitants is preserved.

- *Inter-Platform Secure Aggregation*: One of the most important technologies used at the IoT processing layer is machine learning. The processing units of IoT platforms use machine learning to train models from the data collected from IoT devices. These models are used to improve the quality of the IoT application. Recently, federated learning emerged as a new collaborative machine learning technology to train machine learning models. This new technology consists of several federated clients holding private data and contributing to the training of a machine learning model collaboratively: Each client first trains a local machine learning model using its dataset and further shares training updates with a server. The server computes the average of the training updates and sends the average back to the clients. The clients update their local model and repeat the process so that they finally converge to one shared, global model. Thanks to this new technology, several IoT platforms (acting as FL clients) can train more accurate machine learning models using larger and more diverse datasets (collected from different IoT infrastructures).

  Unfortunately, federated learning may compromise the privacy of IoT users. Adversaries, who have access only to the training updates (sent by each FL client) can perform inference attacks to reveal some data samples from the private training dataset. Therefore, it is very important to secure the federated learning protocol. Secure aggregation can be used to protect the training updates and allows the federated learning server to only compute the average of the updates. By protecting the individual training updates from each IoT platform, secure aggregation helps improve the privacy of IoT users.

In this thesis, we are interested in designing secure aggregation protocols that can scale to a large number of users, do not require the availability of all users, and where their results are publicly verifiable.

1.4.1.2 Remote Attestation Protocols

Remote Attestation (RA) schemes are protocols that enable a device to prove the integrity of its software to another remote device. It involves two roles: the *prover* (the device that proves its software integrity) and the *verifier* (the device that verifies the integrity of the prover's software). In RA, the verifier relies on a root of trust existing on the prover device to perform integrity checks on the prover's running software. It then creates a cryptographic proof and sends it to the verifier. The verifier upon validating

this proof authenticates the prover and thus trusts its software. RA is very useful for IoT applications since it can be used to detect compromised IoT end-devices. Basic RA solutions require installing a verifier in the IoT platform which verifies the state of all IoT end-devices. Such a solution is not practical since it does not scale well with the number of IoT devices. To solve this problem, collaborative RA protocols are used where the IoT devices attest to each other and create a single proof for their software integrity. In this thesis, we raise questions about the scalability of existing collaborative RA protocols in dynamic and heterogeneous networks. Thus, we aim to design efficient RA protocols that scale with a large number of IoT end-devices and can support their dynamic nature and heterogeneity of IoT devices in the network.

1.4.2 Contributions

The thesis describes novel secure aggregation and remote attestation protocols that are suitable for the IoT. The thesis is organized intro three parts. In what follows, we summarize the contributions in each part:

Part I: Secure Aggregation

In Chapter 3, we perform an in-depth literature study of secure aggregation protocols that are based on cryptographic techniques and consequently propose a new definition of this concept. We present existing techniques for secure aggregation as instantiations of our new definition and we compare them. The chapter gives a clear view of the advantages and disadvantages of each of the secure aggregation types.

In Chapter 4, we present the first formal definition of security for secure aggregation protocols in the malicious model. Our new security definitions captures malicious aggregators and users. Then, we design a novel verifiable secure aggregation (VSA) protocol and prove that it achieve our security definitions.

Part II: Secure Aggregation for Federated Learning

In Chapter 5, we conduct a large-scale study on secure aggregation solutions specifically designed for federated learning applications. We identify the main security, privacy, and performance challenges raised by using secure aggregation for federated learning and we categories the existing solutions based on the challenges they tackle. The study finishes with key takeaways that can help developers design, develop or improve secure aggregation schemes for federated learning.

In Chapter 6, we design a novel secure and fault-tolerant aggregation protocol for federated learning. For this purpose, we propose a new threshold encryption scheme (based on the Joye-Libert encryption scheme [JL13]) and we use it to achieve a highly scalable secure aggregation protocol that supports users dropping frequently from the network. We are able to reach the theoretical limit in terms of scalability with respect to the number of users.

Part III: Remote Attestation

In this Chapter 7 we design a novel collaborative remote attestation protocol that relies on device-to-device connection to perform attestation of a large network. Our solution has better scalability than the existing collaborative remote attestation protocol. It focuses on the heterogeneity aspect of IoT networks and integrates fairness by design. We show by experiments that by developing a fair-by-design protocol, we can achieve better scalability for a RA protocol.

Chapter 2

Preliminaries and Cryptographic Building Blocks

In this thesis, we propose new protocols that improve IoT in terms of security. To build these protocols we rely on a set of security assumptions and some existing cryptographic primitives. For this purpose, we introduce, in this chapter, the relevant assumptions and tools used throughout the thesis. Additionally, we present the basic security protocols and cryptographic schemes that are considered interesting building blocks for our solutions.

2.1 Notations

We use some common notations throughout the thesis. We present these notations in Table 2.1.

$\mathbb{N}$	The set of natural numbers $\{1, .., \inf\}$.
$\mathbb{Z}_n$	The set of integer modulus n $\{0, .., n-1\}$.
$[n]$	The set $\{1, .., n\}$.
$[n]^-$	The set $\{1, .., n-1\}$.
$\mathbb{I}$	An interval of integers.
$\mathbb{F}$	An algebraic field.
$\mathbb{G}$	An algebraic group.
$\mathbb{Z}_n^*$	The multiplicative group modulus n.
$x^{\langle i \rangle}$	The i-th bit of $x = x^{\langle 1 \rangle}\|\|...\|\|x^{\langle \ell \rangle}$.
$\|\mathcal{U}\|$	The cardinality of the set $\mathcal{U}$.
$r \leftarrow\$ \mathcal{D}$	r is sampled uniformly at random from distribution $\mathcal{D}$.
$\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$	The two distributions are not distinguishable by a computationally bounded adversary.
$\mathcal{D}_1 \overset{s}{\approx} \mathcal{D}_2$	The two distributions are not distinguishable by a computationally unbounded adversary.

Table 2.1: Common Notations

2.2 Hard Problems and Assumptions

In cryptography, it is common to rely on the hardness of some mathematical problems to build secure systems. In this thesis, we rely on a set of hard problems which we present

in this section.

2.2.1 Decisional and Computational Diffie-Hellman Assumptions

Given a finite, cyclic group $\mathbb{G}$ of order p and generator g, we present the following four assumptions (ordered from strongest to weakest):

Definition 2.2.1: Inv-DDH Assumption

Given the two distributions: $\mathcal{D}_1 : \{(g, g^a, g^c)\}$ and $\mathcal{D}_2 : \{(g, g^a, g^{a^{-1}})\}$ such that $a, c \leftarrow\!\!\$\ \mathbb{Z}_p$ and a^{-1} is the inverse of a in $\mathbb{Z}_p$, the assumption states that $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$.

Definition 2.2.2: DDH Assumption

Given the two distributions: $\mathcal{D}_1 : \{(g, g^a, g^b, g^c)\}$ and $\mathcal{D}_2 : \{(g, g^a, g^b, g^{ab})\}$ such that $a, b, c \leftarrow\!\!\$\ \mathbb{Z}_p$, the assumption states that $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$.

Definition 2.2.3: GDH Assumption

The assumption states that given $g^a, g^b \in \mathbb{G}$ and $\mathcal{O}_{\texttt{DDH}}$ such that $a, b \leftarrow\!\!\$\ \mathbb{Z}_p$ and $\mathcal{O}_{\texttt{DDH}}(g^x, g^y, g^z)$ is an oracle that returns $(g^{xy} \overset{?}{=} g^z)$, there is no polynomial time algorithm that outputs g^{ab} except with some negligible probability.

Definition 2.2.4: CDH Assumption

The assumption states that given $g^a, g^b \in \mathbb{G}$ such that $a, b \leftarrow\!\!\$\ \mathbb{Z}_p$, there is no polynomial time algorithm that outputs g^{ab} except with some negligible probability.

2.2.2 Decisional Composite Residuosity Assumption

Given $N = pq$ where p and q are two large primes. We define the following assumption (initially introduced in [Pai99]) in the multiplicative group $\mathbb{Z}^*_{N^2}$:

Definition 2.2.5: DCR Assumption

Given two distributions: $\mathcal{D}_1 : \{x^N \mod N^2\}$ and $\mathcal{D}_2 : \{x \mod N^2\}$ such that $x \leftarrow\!\!\$\ \mathbb{Z}^*_{N^2}$. The assumption states that $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$.

2.3 Cryptographic Schemes

In this section, we present the main cryptographic schemes used in our thesis.

2.3.1 Threshold Secret Sharing (SS)

A threshold secret sharing scheme (SS) is a scheme that allows sharing a secret value with n parties such that a threshold number of these parties can reconstruct the secret value. It is composed of two algorithms:

- $\texttt{SS.Share}(s, t, n, \mathbb{I}) \rightarrow \{(i, \langle s \rangle_i)\}_{\forall i \in [n]}$: The algorithm takes as input secret $s \in \mathbb{I}$ and a reconstruction threshold t, and generates n random shares.
- $\texttt{SS.Recon}(\{(i, \langle s \rangle_i)\}_{\forall i \in I}, \mathbb{I}\ ) \rightarrow s$: The algorithm takes as input a set of $|I|$ pairs in $[n] \times \mathbb{I}$. It outputs a value $s \in \mathbb{I}$.

Definition 2.3.1: Correctness

SS is said to be correct in $\mathbb{I}$ if and only if:

$$\forall s, \forall n, \forall t, \forall I \text{ s.t. } (I \subset [n]) \wedge (|I| \geq t) \wedge (s \in \mathbb{I}),$$
$$\Pr\left[r \neq s \;\middle|\; \begin{array}{l} \texttt{SS.Share}(s, t, n, \mathbb{I}) \rightarrow \{(i, \langle s \rangle_i)\}_{\forall i \in [n]}, \\ \texttt{SS.Recon}(\{(i, \langle s \rangle_i)\}_{\forall i \in I}, \mathbb{I}) \rightarrow r \end{array} \right] = 0$$

Definition 2.3.2: Security

SS is said to be secure in $\mathbb{I}$ if and only if:

$$\forall s, \forall s', \forall n, \forall t, \forall I, \forall \mathbb{I} \text{ s.t. } (I \subset [n]) \wedge (|I| < t) \wedge (s \in \mathbb{I}) \wedge (s' \in \mathbb{I}),$$
$$\texttt{SS.Share}(s, t, n, \mathbb{I}) \rightarrow \{(i, \langle s \rangle_i)\}_{\forall i \in [n]} : \{(i, \langle s \rangle_i)\}_{\forall i \in I}$$
$$\overset{s}{\approx}$$
$$\texttt{SS.Share}(s', t, n, \mathbb{I}) \rightarrow \{(i, \langle s' \rangle_i)\}_{\forall i \in [n]} : \{(i, \langle s' \rangle_i)\}_{\forall i \in I}$$

Definition 2.3.3: Homomorphism

SS is said to be homomorphic in $\mathbb{I}$ if and only if:

$$\forall s_1, \forall s_2, \forall n, \forall t, \forall I, \text{ s.t. } (I \subset [n]) \wedge (|I| \geq t) \wedge (s_1, s_2 \in \mathbb{I}),$$
$$\Pr\left[r \neq s_1 + s_2 \;\middle|\; \begin{array}{l} \texttt{SS.Share}(s_1, t, n, \mathbb{I}) \rightarrow \{(i, \langle s_1 \rangle_i)\}_{\forall i \in [n]}, \\ \texttt{SS.Share}(s_2, t, n, \mathbb{I}) \rightarrow \{(i, \langle s_2 \rangle_i)\}_{\forall i \in [n]}, \\ \texttt{SS.Recon}(\{(i, \langle s_1 \rangle_i + \langle s_2 \rangle_i)\}_{\forall i \in I}, \mathbb{I}) \rightarrow r \end{array} \right] = 0$$

Realization over the Field $\mathbb{Z}_p$ Shamir's secret sharinf scheme (SSS) [Sha79] is a realization of the threshold secret sharing scheme SSS in the field $\mathbb{Z}_p$. SSS is correct, secure, and homomorphic, and defined as follows:

- $\texttt{SSS.Share}(s, t, n, \mathbb{Z}_p) \to \{(i, \langle s \rangle_i)\}_{\forall i \in [n]}$: The algorithm first generates a polynomial $p(x)$ with uniformly random coefficients in $\mathbb{Z}_p$ and of degree $t-1$ such that $p(0) = s$. It then sets $\langle s \rangle_i = p(i), \forall i \in [n]$.
- $\texttt{SSS.Recon}(\{(i, \langle s \rangle_i)\}_{\forall i \in I}, \mathbb{Z}_p\ ) \to s$: The algorithm uses the Lagrange interpolation formula [Mei02] to compute the value of $p(0)$ as follows:

$$s = p(0) = \sum_{\forall i \in I} \eta_i \langle s \rangle_i \mod p \qquad \text{where} \qquad \eta_i = \prod_{\forall j \in I, j \neq i} \frac{j}{j-i} \mod p$$

We additionally define the following algorithm for Shamir's Secret Sharing:

- $\texttt{SSS.ReconExp}(\{(i, g^{\langle s \rangle_i})\}_{\forall i \in I}, \mathbb{G}) \to g^s$: The algorithm uses the Lagrange interpolation formula on the exponent to compute the value of $g^s = g^{p(0)} = \prod_{\forall i \in I} (g^{\langle s \rangle_i})^{\eta_i}$.

Lemma 2.3.1

Given the cyclic group $\mathbb{G}$ of prime order p and generator g, it holds that:

$$\forall s, \forall n, \forall t, \forall I, \forall g \in \mathbb{G} \text{ s.t. } (I \subset [n]) \wedge (|I| \geq t) \wedge (s \in \mathbb{Z}_p),$$
$$\Pr\left[r \neq g^s \;\middle|\; \begin{array}{l} \texttt{SSS.Share}(s, t, n, \mathbb{Z}_p) \to \{(i, \langle s \rangle_i)\}_{\forall i \in [n]}, \\ \texttt{SSS.ReconExp}(\{(i, g^{\langle s \rangle_i})\}_{\forall i \in I}, \mathbb{G}) \to r \end{array} \right] = 0$$

Lemma 2.3.2

Given the cyclic group $\mathbb{G}$ of prime order p and generator g, it holds that:

$$\forall s_1, \forall s_2, \forall n, \forall t, \forall I, \forall g_1, \forall g_2 \text{ s.t. } (I \subset [n]) \wedge (|I| \geq t) \wedge (s_1, s_2 \in \mathbb{Z}_p) \wedge (g_1, g_2 \in \mathbb{G}),$$
$$\Pr\left[r \neq g_1^{s_1} g_2^{s_2} \;\middle|\; \begin{array}{l} \texttt{SSS.Share}(s_1, t, n, \mathbb{Z}_p) \to \{(i, \langle s_1 \rangle_i)\}_{\forall i \in [n]}, \\ \texttt{SSS.Share}(s_2, t, n, \mathbb{Z}_p) \to \{(i, \langle s_2 \rangle_i)\}_{\forall i \in [n]}, \\ \texttt{SSS.ReconExp}(\{(i, g_1^{\langle s_1 \rangle_i} g_2^{\langle s_2 \rangle_i})\}_{\forall i \in I}, \mathbb{G}) \to r \end{array} \right] = 0$$

Realization over the Integers A variant of Shamir's secret sharing is defined over the integers (rather than in a field). The secret sharing scheme over the integers is defined by Rabin [Rab98] and we denote it by ISS. The scheme shares a secret integer s in an interval $\mathbb{I} = [-\eta, \eta]$ and provides σ-bits statistical security where σ is a security parameter.

- $\texttt{ISS.Share}(s, t, n, \mathbb{I}) \rightarrow \{(i, \langle s \rangle_i)\}_{\forall i \in [n]}$: The algorithm first generates a polynomial $p(x)$ with uniformly random coefficients in $[-2^\sigma \Delta^2 \eta, 2^\sigma \Delta^2 \eta]$ and of degree $t-1$ such that $p(0) = \Delta s$ where $\Delta = n!$. It then sets $\langle s \rangle_i = p(i), \forall i \in [n]$.
- $\texttt{ISS.Recon}(\{(i, \langle s \rangle_i)\}_{\forall i \in I}, \mathbb{Z}_p\ ) \rightarrow s$: The algorithm uses the Lagrange interpolation formula to compute the value of $p(0)$ as follows:

$$s = p(0) = \frac{\sum\limits_{\forall i \in I} \nu_i \langle s \rangle_i}{\Delta^2} \qquad \text{where} \qquad \nu_i = \frac{\Delta \prod\limits_{\forall j \in I, j \neq i} j}{\prod\limits_{\forall j \in I, j \neq i} j - i}$$

2.3.2 Symmetric Encryption

A symmetric encryption scheme parameterized with a security parameter λ, its key space $\mathcal{K}$, message space $\mathcal{M}$, and ciphertext space $\mathcal{C}$ is composed of the following algorithms.

- $\mathsf{Enc}_k(m) \rightarrow e$: It encrypts the message m with key k.
- $\mathsf{Dec}_k(e) \rightarrow m$: It decrypts the ciphertext e with key k

Definition 2.3.4: Correctness

A symmetric encryption scheme is correct if and only if it satisfies:

$$\forall k \in \mathcal{K}, m \in \mathcal{M}, \Pr[m \neq \mathsf{Dec}_k(\mathsf{Enc}_k(m))] = 0$$

Definition 2.3.5: Secure

A symmetric encryption scheme is secure if and only if it satisfies indistinguishability under chosen plaintext attacks (IND-CPA). See formal definition in [BN08].

Definition 2.3.6: Authenticated

A symmetric encryption scheme is authenticated if and only if it satisfies ciphertext integrity (INT-CTXT). See formal definition in [BN08].

Definition 2.3.7: Non-Committing

A symmetric encryption scheme is non-committing if $\forall m \in \mathcal{M}$ we have $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$ where $\mathcal{D}_1$ and $\mathcal{D}_2$ are defined as follows:

- $\mathcal{D}_1 : \{(e, k)\}$ such that $k \leftarrow\!\!\$\ \mathcal{K}$ and $e \leftarrow \mathsf{Enc}_k(m)$
- $\mathcal{D}_2 : \{(e', k')\}$ such that $e' \leftarrow\!\!\$\ \mathcal{S}_1(1^\lambda)$, $k' \leftarrow\!\!\$\ \mathcal{S}_2(e', m)$ and $\mathcal{S}_1$, $\mathcal{S}_2$ are any PPT

algorithms that may share a state.

Definition 2.3.7 of non-committing encryption says that it is possible for a simulator to come up with a ciphertext which can later be explained as an encryption of any message, in such a way that the joint distribution of the ciphertext and the key in this simulated experiment is indistinguishable from the normal use of the encryption scheme, where a key is first sampled and then an encryption of m is generated.

In this thesis, we will use $(\mathsf{Enc}, \mathsf{Dec})$ for authenticated encryption scheme and $(\mathsf{Enc}^*, \mathsf{Dec}^*)$ for a non-commiting authenticated encryption scheme.

2.3.3 Pseudo Random Generator

$\texttt{PRG}(m, R, b) \to B$: is a pseudo-random generator that extends seed $b \in \{0,1\}^\lambda$ to vector $B \in \mathbb{Z}_R^m$ (vector of m elements and each element is in $\mathbb{Z}_R$).

Definition 2.3.8: Security

Given the two distribution $\mathcal{D}_1 : \{A\}$ such that $A \leftarrow\!\!\$\ \mathbb{Z}_R^m$ and $\mathcal{D}_2 : \{B\}$ such that $B \leftarrow \texttt{PRG}(b)$ where $b \leftarrow\!\!\$\ \{0,1\}^\lambda$ and λ is a large security parameter, it holds that $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$.

2.3.4 Key Agreement Scheme

A key agreement scheme $\texttt{KA}$ is a scheme parameterized by the security parameter λ and the key space $\mathcal{K}$. It is composed of the following algorithms:

- $\texttt{KA.Param}(1^\lambda) \to pp$: Given a security parameter λ it generates the public parameters pp.
- $\texttt{KA.Gen}(pp) \to (pk, sk)$: This algorithm generates a key pair from the public parameter.
- $\texttt{KA.Agree}(pp, sk, pk_1, pk_2, G) \to k$: This algorithm uses a private key, two public keys, and a hash function G to generate an encryption key.

Definition 2.3.9: Correctness

The key agreement scheme is correct if and only if:

$$\forall\lambda, \Pr\left[\, k_{1,2} \neq k_{2,1} \;\middle|\; \begin{array}{l} \texttt{KA.Param}(1^\lambda) \to pp, \\ \texttt{KA.Gen}(pp) \to (pk_1, sk_1), \\ \texttt{KA.Gen}(pp) \to (pk_2, sk_2), \\ \texttt{KA.Agree}(pp, sk_1, pk_1, pk_2, G) \to k_{1,2}, \\ \texttt{KA.Agree}(pp, sk_2, pk_2, pk_1, G) \to k_{2,1} \end{array} \right] = 0$$

Definition 2.3.10: Security

The key agreement scheme is secure if and only if $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$ where the two distributions are defined as follows:

$$\mathcal{D}_1 : \left\{ (pk_1, pk_2, r) \;\middle|\; \begin{array}{l} \texttt{KA.Param}(1^\lambda) \to pp, \\ \texttt{KA.Gen}(pp) \to (pk_1, sk_1), \\ \texttt{KA.Gen}(pp) \to (pk_2, sk_2), \\ r \leftarrow\!\!\$\ \mathcal{K} \end{array} \right\}$$

$$\mathcal{D}_2 : \left\{ (pk_1, pk_2, k) \;\middle|\; \begin{array}{l} \texttt{KA.Param}(1^\lambda) \to pp, \\ \texttt{KA.Gen}(pp) \to (pk_1, sk_1), \\ \texttt{KA.Gen}(pp) \to (pk_2, sk_2), \\ \texttt{KA.Agree}(pp, sk_1, pk_1, pk_2, G) \to k \end{array} \right\}$$

Realization based on DDH Under the DDH assumption we can construct a KA scheme that is secure in the random oracle G (see Section 2.4.3) as follows:

- $\texttt{KA.Param}(1^\lambda) \to pp$: From the security parameter λ, it chooses a group $\mathbb{G}$ of order p and generator g where the DDH holds in $\mathbb{G}$. The public parameters are set to $pp = (\mathbb{G}, g, p)$.
- $\texttt{KA.Gen}(pp) \to (pk, sk)$: It outputs (g^a, a) where $a \leftarrow\!\!\$\ \mathbb{Z}_p$.
- $\texttt{KA.Agree}(pp, sk, pk_1, pk_2, G) \to$: It outputs $k \leftarrow G^{\mathbb{G}}_{(pk_1, pk_2)}(pk_2^{sk})$.

2.4 Ideal Functionalities and Protocols

2.4.1 The Universal Composability Framework (UC)

In this thesis, we use the standard universal composability framework proposed by Canetti [Can20]. The UC framework defines a probabilistic polynomial time (PPT) environment machine $\mathcal{Z}$ that oversees the execution of a protocol in one of two worlds: The "ideal world" execution involves "dummy parties" (some of whom may be corrupted by an ideal adversary $\mathcal{S}$) interacting with a functionality $\mathcal{F}$. The "real world" execution involves PPT parties (some of whom may be corrupted by a PPT real-world adversary $\mathcal{A}$) interacting only with each other in some protocol Π. Let $\texttt{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(x)$ denote the random variable (over the local random choices of all the real world parties) describing the output of an execution of Π with environment $\mathcal{Z}$ and adversary $\mathcal{A}$, on input x. Similarly, let $\texttt{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(x)$ denote the random variable (over the local random choices of a simulator $\mathcal{S}$) describing the output of an execution of $\mathcal{F}$ with environment $\mathcal{Z}$ and simulated adversary $\mathcal{S}$, on input x. Moreover, Let $\texttt{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ and $\texttt{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensembles $\{\texttt{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(x)\}_{x\in\{0,1\}^*}$ and $\{\texttt{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(x)\}_{x\in\{0,1\}^*}$ respectively. We refer to [Can20] for a more detailed description of both executions.

Definition 2.4.1: Protocol Realization

Let $\mathcal{F}$ be an ideal functionality. A protocol Π is said to UC-realize $\mathcal{F}$ if for any adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$,

$$\texttt{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \texttt{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}.$$

Additionally, we say that protocol Π operates in $\mathcal{G}$-hybrid-model, if Π uses the ideal functionality $\mathcal{G}$.

Definition 2.4.2: Protocol Emulation

Let $\mathcal{G}_1, \mathcal{G}_2$ be ideal functionalities and let Π_1, Π_2 be multi-party protocols. We say Π_1 in the $\mathcal{G}_1$-hybrid model UC-emulates Π_2 in the $\mathcal{G}_2$-hybrid model if for any adversary $\mathcal{A}_1$, there exists an adversary $\mathcal{A}_2$ such that for all environments $\mathcal{Z}$,

$$\texttt{EXEC}_{\Pi_1,\mathcal{A}_1,\mathcal{Z}} \overset{c}{\approx} \texttt{EXEC}_{\Pi_2,\mathcal{A}_2,\mathcal{Z}}.$$

Furthermore, we follow the definitions in [Can20] for modeling adversary corruptions. When the adversary corrupts a party it sends a backdoor message $(\texttt{sid}, \texttt{ID})$ where `sid` denotes the session id and `ID` denotes the identifier of the corrupted party. This happens at the very beginning of the protocol execution in the case of static corruption. For simplicity, we do not show these messages in the ideal functionalities as we assume the default behavior is that party `ID` is recorded as corrupted.

We also use the work in [CR03] which describes the execution of multi-sessions of a protocol. We denote by $\hat{\Pi}$ and $\hat{\mathcal{F}}$, the multi-session extensions of the protocol Π and functionality $\mathcal{F}$ respectively. We recall that the UC theorem with joint state (JUC theorem):

Theorem 2.4.1: Universal Composability with Joint State [CR03]

Given protocol Π that UC-realizes $\mathcal{F}$ in $\mathcal{G}$-hybrid-model and a protocol $\hat{\Psi}$ that UC-realizes functionality $\hat{\mathcal{G}}$ in the real-world model, it is true that the composition of Π and $\hat{\Psi}$: $\Pi \circ \hat{\Psi}$ in the real-world model UC-emulates Π in $\mathcal{G}$-hybrid-model.

In this section, we present the ideal functionality of the primitives we use in this thesis. Additionally, we discuss a possible realization for some of these functionalities.

2.4.2 Common Reference String (CRS)

We use the ideal functionality $\mathcal{F}_{\texttt{CRS}}^{\mathcal{D},n}$ parameterized by the sampling algorithm $\mathcal{D}$ and the number of parties n. The purpose of this functionality is to have a common string `crs` sampled from a fixed distribution and available to all parties of the protocol. Notice that this proposed functionality differs from the one proposed in [CR03] because it supports distributing the `CRS` to n parties. It is described in Figure 2.1.

Functionality $\mathcal{F}_{\texttt{CRS}}^{\mathcal{D},n}$

$\mathcal{F}_{\texttt{CRS}}^{\mathcal{D},n}$ runs with parties P_1,..., P_n and is parameterized by an algorithm $\mathcal{D}$ and a security parameter λ.

- When receiving a message $(\texttt{sid}, \texttt{CRS}, P_i)$ from P_i, if this is the first `CRS` message, compute $\texttt{crs} \leftarrow\$ \mathcal{D}(1^\lambda)$. Then, send $(\texttt{sid}, \texttt{CRS}, \texttt{crs})$ to P_i and $\mathcal{S}$. Check if received message $(\texttt{sid}, \texttt{CRS}, ...)$ from all parties. If yes, halt. If no, continue.

Figure 2.1: Ideal functionality of `CRS`

2.4.3 Random Oracle Model

Random oracles are typically used as an idealised replacement for cryptographic hash functions in schemes where strong randomness assumptions are needed. The ideal functionality of the random oracle returns a uniformly random response from the output domain. Additionally, if a query is repeated, it responds the same way every time that the same query is submitted. Throughout this thesis, we use the following hash functions modeled as random oracles.

- $H^{\mathbb{G}} : \{0,1\}^* \mapsto \mathbb{G}$. This hash function is used to generate random group elements from arbitrary strings. The group used in this hash function is indicated as a parameter in its superscript.
- $G^{\mathbb{G}} : \mathbb{G}^3 \mapsto \{0,1\}^\lambda$. This hash function is used as a key-derivation function to extract a λ-bit key from a group element, and the first two inputs are used to seed the function.

2.4.4 Sharing Random Number (`RShare`)

We define the ideal functionality $\mathcal{F}_{\texttt{RShare}}^{t,n,\mathbb{F}}$ parameterized by the field $\mathbb{F}$, the number of parties n, and a threshold $t \leq n$. The purpose of this functionality is to generate a uniformly random element in the field $\mathbb{F}$ and secretly share it (using Shamir's secret sharing) with the n parties. The functionality is described in Figure 2.2.

Realization This functionality can be realized by letting each party P_i generate its random value $r_i \leftarrow\$ \mathbb{F}$ and secretly share it (using authenticated channels) with the other parties $\texttt{SS.Share}(r_i, t, n, \mathbb{F}) \rightarrow \{(j, \langle r_i \rangle_j)\}_{\forall j \in [n]}$. Then, each partly P_i simply sums the shares it receives to obtain the share $\langle r \rangle_i = \sum_{j=1}^{n} \langle r_j \rangle_i$ where $r = \sum_1^n r_i$.

2.4.5 Oblivious Transfer (`OT`)

An oblivious transfer protocol (OT) is a protocol executed between two parties: the sender and the receiver. The sender inputs a pair (m_0, m_1) whereas the receiver inputs a selection bit $x \in \{0,1\}$. At the end of the protocol, the receiver outputs value m_x

Functionality $\mathcal{F}_{\texttt{RShare}}^{t,n,\mathbb{F}}$

$\mathcal{F}_{\texttt{RShare}}^{t,n,\mathbb{F}}$ runs with parties P_1,..., P_n and is parameterized by field $\mathbb{F}$, the number of parties n, and a threshold $t \leq n$.

- When receiving a message $(\texttt{sid}, \texttt{init}, P_i)$ from P_i, if this is the first `init` message, compute $r \leftarrow\$ \mathbb{F}$.
- When all `init` messages are received from the n parties, compute $\texttt{SS.Share}(r, t, n, \mathbb{F}) \rightarrow \{(i, \langle r \rangle_i)\}_{\forall i \in [n]}$ and send $(\texttt{sid}, \texttt{share}, \langle r \rangle_i)$ to each P_i and send $(\texttt{sid}, \texttt{share})$ to $\mathcal{S}$.

Figure 2.2: Ideal functionality of random secret sharing

while the sender outputs nothing. The security of the OT protocol ensures that the receiver does not learn any information about m_{1-x} and the sender does not learn any information about x. The ideal functionality of OT is shown in Figure 2.3.

Functionality $\mathcal{F}_{\texttt{OT}}$

$\mathcal{F}_{\texttt{OT}}$ interacts with a receiver R and a sender S.
Auxiliary Inputs: the length of the sender input ℓ

- Upon receiving a message $(\texttt{sid}, \texttt{send}, S, m_0, m_1)$ from S, where $m_i \in \{0,1\}^\ell$, store (m_0, m_1).
- Upon receiving a message $(\texttt{sid}, \texttt{recv}, R, x)$ from R, check if a $(\texttt{send}, \texttt{sid}, S, ...)$ message was previously sent. If yes, send $(\texttt{sid}, \texttt{sent}, m_x)$ to R and $(\texttt{sid}, \texttt{sent})$ to adversary $\mathcal{S}$ and halt. If not, send nothing to R (but continue running).

Figure 2.3: Ideal functionality of oblivious transfer protocol.

We present the construction proposed by Chou and Orlandi [CO15] in Figure 2.4. The authors of the OT protocol prove that it UC-realizes the functionality $\mathcal{F}_{\texttt{OT}}$ in the random oracle model. Their proof relies on the CDH assumption. Later Hauck and Loss [HL17] identify a flow in the proof from [CO15] and provide a patch for the proof that relies on the hardness of the GDH assumption. Based on the work of Hauck and Loss [HL17] we restate the following two Lemmas.

Lemma 2.4.1

For any PPT adversary $\mathcal{A}$ that statically corrupts a receiver in $\Pi_{\texttt{OT}}$ protocol, and any PPT environment $\mathcal{Z}$, the algorithm $\mathcal{S}_{\texttt{rcv}}$ (defined in [HL17]) simulates the execution for $\mathcal{A}$ under the GDH assumption (i.e., $\texttt{IDEAL}_{\mathcal{F}_{\texttt{OT}}, \mathcal{S}_{\texttt{rcv}}, \mathcal{Z}} \overset{c}{\approx} \texttt{EXEC}_{\Pi_{\texttt{OT}}, \mathcal{A}, \mathcal{Z}}$).

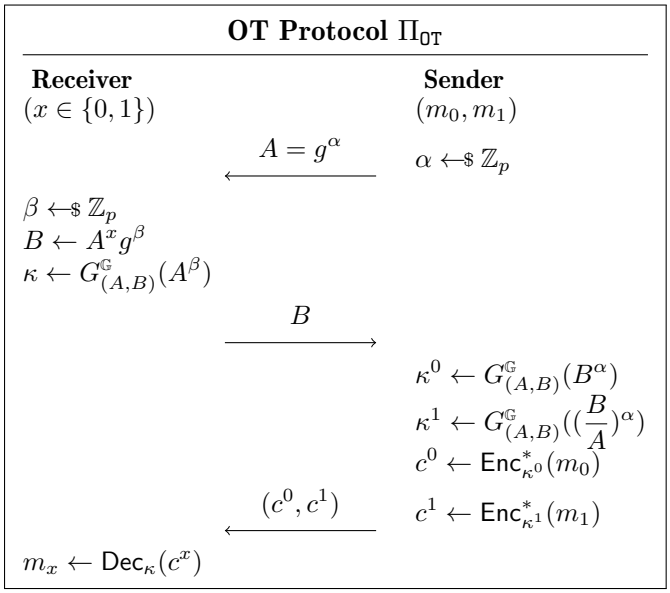


Figure 2.4: The OT protocol [CO15] parameterized by the cyclic group $\mathbb{G}$ of order p and generator g.

Lemma 2.4.2

For any PPT adversary $\mathcal{A}$ that statically corrupts a sender in $\Pi_{\mathtt{OT}}$ protocol, and any PPT environment $\mathcal{Z}$, the algorithm $\mathcal{S}_{\mathtt{snd}}$ (defined in [HL17]) simulates the execution for $\mathcal{A}$ under the CDH assumption (i.e., $\mathtt{IDEAL}_{\mathcal{F}_{\mathtt{OT}},\mathcal{S}_{\mathtt{snd}},\mathcal{Z}} \overset{c}{\approx} \mathtt{EXEC}_{\Pi_{\mathtt{OT}},\mathcal{A},\mathcal{Z}}$).

In a nutshell, the simulator $\mathcal{S}_{\mathtt{rcv}}$ takes the messages of the receiver and extracts the receiver's choice. It then gets the output from $\mathcal{F}_{\mathtt{OT}}$ and generates the simulated sender messages accordingly. On the other hand, the simulator $\mathcal{S}_{\mathtt{snd}}$ takes the messages of the sender and extracts the sender's input pair. In our work, we are interested in reusing $\mathcal{S}_{\mathtt{snd}}$ and $\mathcal{S}_{\mathtt{rcv}}$ algorithms for our proofs.

2.4.6 Garbled Circuits (GC)

A garbled circuit protocol proposed by Yao [Yao86] runs between two parties (a garbler $\mathcal{G}$ and an evaluator $\mathcal{E}$). It is used to evaluate an arbitrary function f chosen by $\mathcal{G}$ on the private input x of $\mathcal{E}$. Yao garbling scheme is defined as two algorithms:

- $\mathtt{GC.Grb}(1^{\lambda}, f) \rightarrow (F, \{l^b_{x,i}\}_{\forall b \in \{0,1\}, \forall i \in [\ell_x]}, \{l^b_{\mathtt{out},i}\}_{\forall b \in \{0,1\}, \forall i \in [\ell_{\mathtt{out}}]})$: Given the security parameter λ and the function f, the algorithm generates a garbled circuit F and a label for each bit of the input and output (ℓ_x and $\ell_{\mathtt{out}}$ are the bit-length of the input and output, respectively).
- $\mathtt{GC.Eval}(F, \{l^{x\langle i \rangle}_{x,i}\}_{\forall i \in [\ell_x]}) \rightarrow \{l^{\mathtt{out}\langle i \rangle}_{\mathtt{out},i}\}_{\forall i \in [\ell_{\mathtt{out}}]}$: Given the garbled circuit F and the labels of the input bits, the algorithm outputs the labels of the result $\mathtt{out} = f(x)$.

Protocol with GC and OT to compute $f^{\theta,M}$

Evaluator $(x = x_1 \| ... \| x_\ell)$		**Garbler** (θ, M)
		$F, \{l^b_{x,i}\}, l^b_{\texttt{out}} \leftarrow \texttt{GC.Grb}(1^\lambda, f^\theta)$ $\forall i \in [\ell], \forall b \in \{0,1\}$
	$\xleftarrow{A = g^\alpha, F}$	$\alpha \leftarrow\!\!\$\ \mathbb{Z}_p$
$\beta_i \leftarrow\!\!\$\ \mathbb{Z}_p\ \forall i \in [\ell]$ $B_i \leftarrow A^{x^{\langle i \rangle}} g^{\beta_i}\ \forall i \in [\ell]$ $\kappa_i \leftarrow G^{\mathbb{G}}_{(A,B_i)}(A^{\beta_i})\ \forall i \in [\ell]$		
	$\xrightarrow{\{B_i\}_{\forall i \in [\ell]}}$	
		$\kappa^0_i \leftarrow G^{\mathbb{G}}_{(A,B_i)}(B_i^\alpha)\ \forall i \in [\ell]$ $\kappa^1_i \leftarrow G^{\mathbb{G}}_{(A,B_i)}((\frac{B_i}{A})^\alpha)\ \forall i \in [\ell]$ $c^0_i \leftarrow \mathsf{Enc}^*_{\kappa^0_i}(l^0_{x,i})\ \forall i \in [\ell]$ $c^1_i \leftarrow \mathsf{Enc}^*_{\kappa^1_i}(l^1_{x,i})\ \forall i \in [\ell]$
	$\xleftarrow{\{(c^0_i, c^1_i\}_{\forall i \in [\ell]}, C}$	$C \leftarrow \mathsf{Enc}^*_{l^1_{\texttt{out}}}(M)$
$l^{x^{\langle i \rangle}}_{x,i} \leftarrow \mathsf{Dec}_{\kappa_i}(c_i^{x^{\langle i \rangle}})\ \forall i \in [\ell]$ $l^{\texttt{res}}_{\texttt{out}} \leftarrow \texttt{GC.Eval}(F, \{l^{x^{\langle i \rangle}}_{x,i}\}_{\forall i \in [\ell]})$ $M \leftarrow \mathsf{Dec}_{l^{\texttt{res}}_{\texttt{out}}}(C)$		

Figure 2.5: Protocol to evaluate $f^{\theta,M}$ using a garbled circuit and oblivious transfer. The protocol is parameterized by the cyclic group $\mathbb{G}$ of order p and generator g

In this thesis, we are interested in the function $f^{\theta,M}$ which checks if the evaluator's input is less than θ and if so, returns a message M. We define the two functions f^θ and $f^{\theta,M}$ as follows

$$f^\theta(x) = \begin{cases} 1 & \text{, if } x \leq \theta \\ 0 & \text{, otherwise} \end{cases} \qquad f^{\theta,M}(x) = \begin{cases} M & \text{, if } x \leq \theta \\ \perp & \text{, otherwise} \end{cases} \tag{2.1}$$

To evaluate $f^{\theta,M}$ using a garbled circuit, the garbler $\mathcal{G}$ generates and sends a garbled circuit F of f^θ and a label for each bit of the input. $\mathcal{G}$ sends F to $\mathcal{E}$ and then transfers the labels of x to $\mathcal{E}$ using OT. $\mathcal{G}$ also uses the output label $l^1_{\texttt{out}}$ to encrypt the message M and send the ciphertext to $\mathcal{E}$. Finally, $\mathcal{E}$ evaluates F on the labels of x and obtains $l^{\texttt{res}}_{\texttt{out}}$ where $\texttt{res} \in \{0,1\}$. If $\texttt{res} = 1$ the evaluator decrypts the ciphertext and obtains M. We describe the protocol in Figure 2.5. Notice, that this protocol is secure when the garbler is honest-but-curious (follows the protocol steps).

2.4.7 Zero-Knowledge Proofs (ZKP)

Zero-Knowledge Proof protocols are two-party protocols where a prover P interacts with a verifier V to convince it that a given statement is true while P avoids conveying any additional information apart from the fact that the statement is indeed true. We show the ideal functionality of ZKP in Figure 2.6.

Functionality $\mathcal{F}_{\mathtt{ZK}}^{\mathcal{R}}$

$\mathcal{F}_{\mathtt{ZK}}^{\mathcal{R}}$ interacts with a prover P and a verifier V and is parameterized by the relation $\mathcal{R}$.

- Upon receiving a message $(\mathtt{sid}, \mathtt{prove}, x, w)$ from P, store x and w, and continue.
- Upon receiving a message $(\mathtt{sid}, \mathtt{verify}, x')$ from P, abort if $x' \neq x$, otherwise, send $(\mathtt{sid}, \mathcal{R}(x, w))$ to V and $\mathcal{S}$, and halt.

Figure 2.6: Ideal functionality for zero-knowledge protocol.

2.4.7.1 Discrete Logarithm Relation $\mathcal{R}_{\mathtt{DL}}^{n}$

Given $\mathbb{G}$ (a group where DDH holds) of order p the multi-value discrete logarithm relation $\mathcal{R}_{\mathtt{DL}}^{n}(x, w) : \mathbb{G}^{2n} \times \mathbb{Z}_p \mapsto \{0, 1\}$ is defined as:

$$\mathcal{R}_{\mathtt{DL}}^{n}(\{A_i, B_i\}_{\forall i \in [n]}, w) : \bigwedge_{\forall i \in [n]} (B_i = A_i^w)$$

Realization: We can realize $\mathcal{F}_{\mathtt{ZK}}^{\mathcal{R}_{\mathtt{DL}}^{n}}$ as an extension of Schnorr's zero-knowledge proof for discrete logarithm [Sch90]. More precisely, P samples n uniformly random values $r_i \leftarrow\!\!\$\ \mathbb{Z}_p\ \forall i \in [n]$ and sends $\{g^{r_i}\}_{\forall i \in [n]}$ to V. V samples the challenge $c \leftarrow\!\!\$\ \mathbb{Z}_p$ and sends it to P. P computes the proof $z_i \leftarrow wc + r_i \mod \mathbb{Z}_p$, and sends $\{z_i\}_{\forall i \in [n]}$ to V which finally checks $\bigwedge_{\forall i \in [n]}(g^{r_i} B_i^c = g^{z_i})$. This protocol can be transformed into a non-interactive zero-knowledge proof using the Fiat-Shamir transformation [FS87].

2.4.7.2 Tag Relation $\mathcal{R}_{\mathtt{Tag}}$

Given $\mathbb{G}$ (a group where DDH holds) of order p, we define the tag relation $\mathcal{R}_{\mathtt{Tag}}(x, w) : \mathbb{G}^3 \times \mathbb{Z}_p^2 \mapsto \{0, 1\}$ as:

$$\mathcal{R}_{\mathtt{Tag}}(\{A, B, C\}, (a, b)) : C = A^a B^b$$

Realization: Similarly, we can realize $\mathcal{F}_{\mathtt{ZK}}^{\mathcal{R}_{\mathtt{Tag}}}$ as an extension of Schnorr's zero-knowledge proof for discrete logarithm [Sch90]. More precisely, P samples two uniformly random values $r_1, r_2 \leftarrow\!\!\$\ \mathbb{Z}_p$ and sends $(g^{r_1 + r_2})$ to V. V samples the challenge $c \leftarrow\!\!\$\ \mathbb{Z}_p$ and sends it to P. P computes the proof $z_1 \leftarrow ac + r_1 \mod \mathbb{Z}_p$ and $z_2 \leftarrow bc + r_2 \mod \mathbb{Z}_p$, then sends (z_1, z_2) to V which finally checks $C g^{r_1 + r_2} = g^{z_1} g^{z_2}$. This protocol can also be transformed into a non-interactive zero-knowledge proof using Fiat-Shamir transformation [FS87].

Part I

Secure Aggregation

Chapter 3

Characterization of Secure Aggregation

In this chapter, we provide a formal definition of a secure aggregation protocol and we study the existing threat models studied in the literature: the honest-but-curious model and the malicious model. We first identify the four major technologies used to build secure aggregation solutions: differential privacy, trusted execution environment, anonymity, and cryptography. Then, we focus on solutions that are based on cryptography. We study the existing cryptography-based solutions in the honest-but-curious model and we suggest a systematic categorization of these solutions.

3.1 Environment of Secure Aggregation

One of the most important functionalities of the IoT is aggregating the data collected by the end-devices. Many IoT applications such as smart grids [ADMC17], water management systems, and traffic control in cities use this functionality. In most of these applications, aggregating the data involves the operation of the statistical sum of the data points collected from multiple geo-dispersed data sources (eg., sensors).

Nevertheless, these individual data points usually involve sensitive data. This raises serious privacy concerns and thus calls for suitable privacy-enhancing technologies to protect the confidentiality of end-devices' data while still being able to perform the aggregation operation. During the past 20 years, a huge amount of research focused on designing secure aggregation (SA) solutions [ÖM07, LEM14a, BSK+19, BIK+17, DA16, CMT05] for various applications. The goal is to enable the computation of the sum of several parties' inputs without leaking any information about each individual input except the aggregate (the sum).

In this thesis, aggregation refers to the process where data collected from multiple sources are summed up. Usually, data is collected in consecutive timestamps and the sum (aggregate) is calculated for each timestamp. For many applications, the data contains sensitive information. Usually, secure aggregation involves the following actors:

- *Users* (U): Parties that provide the input data $x_{i,\tau}$ at each timestamp τ.

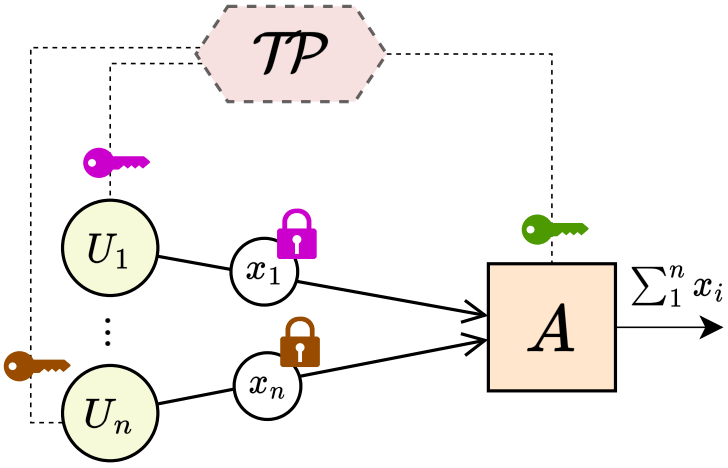


Figure 3.1: An illustration of secure aggregation between n users $(U_1, ..., U_n)$ and one aggregator A. A trusted party $\mathcal{TP}$ is used to setup up the parties.

- *Aggregators* (A): Parties that perform the aggregation to obtain the sum X_τ of the input data at each timestamp τ.
- *Trusted Third Party* ($\mathcal{TP}$): A trusted part that may be required for setup purposes in some secure aggregation protocols.

3.2 Threat Models and Security Requirements for Secure Aggregation

We describe the most popular threat models considered for secure aggregation. Namely, the *honest-but-curious* model and the *malicious* model. Then, we define the security requirements for secure aggregation.

3.2.1 Threat Models

Honest-but-curious Model A common security model for secure aggregation schemes is the *honest-but-curious* model. An honest-but-curious adversary correctly follows the protocol steps but remains curious to discover any private information. The adversary may corrupt the aggregator and some users. When the aggregator corrupts multiple parties it accesses all their private information. This models collusion between the corrupted aggregator and corrupted users.

Malicious Model In addition to the *honest-but-curious* model, several research works consider a *malicious* model where the adversaries are more powerful. A malicious adversary may deviate from the protocol steps and thus has arbitrary inputs to the protocol. There are some variations of this model where the adversary only corrupts the

aggregator (a.k.a. malicious aggregator model) or only corrupts a subset of users (a.k.a. malicious users model).

In this chapter, we only study SA solutions secure under the honest-but-curious model.

3.2.2 Security Requirements

We present the main two security requirements that define the security of secure aggregation protocols. The first requirement is *Aggregator Obliviousness* and it was initially proposed by Shi et al. [SCR+11]. We state this security requirement in Definition 3.2.1. The second requirement is *Aggregator Unforgeability* and it was initially proposed by Leontiadis et al. [LEÖM15] and improved later in several other works. We state this security requirement in Definition 3.2.2.

Definition 3.2.1

Aggregator Obliviousness: This security notion ensures that an adversary (by corrupting the aggregator) cannot learn more than what could be learned from the sum of the users' inputs. If the adversary corrupts some users, the notion only requires that the adversary gets no extra information about the values of the honest users beyond their sum.

Definition 3.2.2

Aggregate Unforgeability: This security notion ensures that an adversary (by corrupting the aggregator) cannot output an incorrect sum without being detected by honest users. If the adversary corrupts a small subset of users ($U_{\texttt{corr}}$), the notion requires that the adversary can only make an insignificant change in the result of the sum. This means that given X is the sum of the honest users' inputs, the adversary can only output a result X' such that $X \leq X' \leq X + \theta|U_{\texttt{corr}}|$ where θ is the maximum allowed value of a user input ($x_{i,\tau} < \theta$) .

3.3 Existing Secure Aggregation Protocols

There are mainly four types of secure aggregation (SA) protocols: SA based on differential privacy, SA based on Trusted-Execution Environment (TEE), SA based on anonymity, and SA based on cryptography. In the following, we elaborate on each type of secure aggregation. Table 3.1 presents a comparison between the different types.

3.3.1 SA based on differential privacy

Differential privacy (DP) [DR14] is a technique of adding a controlled amount of randomness (noise) to some data such that this noise is sufficient to hide sensitive information

SA Technique	Accurate Result	Untrusted Platform Owners	Public Communication Channels
DP	✗	✔	✔
TEE	✔	✗	✔
Secure Shuffle	✔	✔	✗
Cryptography	✔	✔	✔

Table 3.1: Comparison between different Secure Aggregation techniques.

but also keeps this data useful. Hence, differential privacy provides a mathematically quantifiable way to balance data privacy and data utility. There are mainly two methods for DP mechanisms: Global Differential Privacy (GDP) and Local Differential Privacy (LDP).

Global Differential Privacy GDP relies on the idea of adding noise to the output of some algorithm run on the private data as opposed to adding noise to the data entries themselves. This is not practical in the case of SA since adding noise to the aggregation results does not protect the users' inputs from untrusted aggregation servers (IoT platform).

Local Differential Privacy On the other hand, LDP solutions [MASN21, ASY+18, TF19, YWW21, CYD20, STL20, DHCP21] are more practical for SA. These solutions propose to add noise to the user input locally at each user device. The noise serves as a layer to prevent attackers from learning private information from the user's input. A common type of noise used is Gaussian noise. Since all users need to add noise to their inputs before sending them, this method ends up adding too much noise to the final aggregate. Thus, affecting significantly the accuracy of the results. Therefore, the main concern with this approach is the accuracy of the result as noise is amplified during the aggregation.

3.3.2 SA based on trusted-execution environment

Another method to perform secure aggregation is to use a Trusted-Execution Environment (TEE) at the server [ZJF+21, NMZ+21, ZWC+21, MHK+21]. The TEE is a hardware-separated area of the main processor that guarantees the confidentiality and integrity of the processed data [Tru23]. It allows secure storage of data and execution of sensitive code on an untrusted device by providing a secure environment for processing private data. This environment is often referred to as a secure enclave. Previous work [ZJF+21, MHK+21] propose to perform the secure aggregation operation within this secure enclave using TEEs. More precisely, each SA user and the SA server runs on TEE-enabled devices. Consequently, the inputs are processed within the user's TEE and sent to the server where they are aggregated within the secure enclave. Zhao et al. [ZJF+21] propose to encrypt the private user inputs using efficient symmetric encryption methods and send the ciphertexts to the server where they get decrypted and aggregated inside the TEE

environment. Mo et al. [MHK+21] follow the same approach while applying SA for federated learning. The authors propose to train machine learning classifiers at the user within the secure enclave and encrypt the models before sending them outside the TEE. Moreover, Kalapaaking et al. [KKR+23] and Zhang et al. [ZM21] improved the security guarantees of SA using TEEs to ensure the integrity of the aggregation result. The aforementioned work proposed to integrate a blockchain platform [QPG+21] to verify the aggregation result via a consensus mechanism before it is added to the blockchain.

Using a TEE provides important advantages in terms of isolation of the SA task from the real word execution environment which can be compromised by attackers. However, TEE can only protect against external attackers which remotely compromise the platform's devices. More clearly, it does not protect against malicious platforms (i.e., platform owners with malicious intentions). Indeed, a malicious platform owner has physical access to the TEE hardware and can simply misconfigure during the setup phase.

3.3.3 SA based on anonymity

A different method relies on the anonymous communication assumption [IKOS06]. This method is also known as secure shuffling where users split their inputs into shares and send them anonymously to the aggregator [ZWC+21] (i.e., using different anonymous channels). The aggregator can simply sum up all the received shares to compute the aggregate. Thanks to the anonymous communication, the server cannot map the received shares to their corresponding users and thus is unable to reconstruct the user inputs. This solution offers a good advantage in terms of the accuracy of the result and in terms of the security guarantee against a malicious platform. Unfortunately, realizing these communication channels is often impossible in the context of IoT. Indeed, it is very hard to hide the end-devices' identity against the IoT platform even when using anonymous networks such as Tor [Ödé22]. This is mainly because IoT devices can be profiled and de-anonymized based on side channel information such as activity time, frequency of input generation, network delays, etc.

3.3.4 SA based on cryptography

Finally, cryptographic techniques are also used to design secure aggregation protocols. This technique relies on strong mathematical assumptions to protect the user inputs and sends them in public. By relying on some hard mathematical problems, the user inputs are transformed to random looking values which can still undergo linear operations such as addition. Hence, it allows to compute the aggregation result without decrypting the individual user inputs at any point in time. Many researchers are particularly interested in this type of secure aggregation since it does not significantly affect the accuracy of the aggregation result. It also does not require the trust of the platform owners to configure a TEE environment. Moreover, it does not rely on strong assumptions regarding the communication channels between the users and the server as the case for secure shuffling. These important advantages render cryptographic-based SA a suitable choice for many IoT applications.

In this thesis, we only study secure aggregation (SA) solutions based on cryptographic schemes as these seem to be the most popular ones for many applications.

3.4 Secure Aggregation based on Cryptography

We first identify and investigate the three phases of secure aggregation protocol: Setup, Protection, and Aggregation. Then, we regroup secure aggregation solutions into two categories based on the underlying cryptographic tools used to build the SA protocol. Specifically, we distinguish encryption-based secure aggregation and multi-party-computation (MPC)-based secure aggregation. For each of the categories, we show how to build the baseline[1] secure aggregation protocol from the existing cryptographic schemes by defining the algorithms executed at each SA phase. We provide some realizations of the different categories and we summarize their advantages and disadvantages in Table 3.2.

3.4.1 Secure Aggregation Protocol Phases

A secure aggregation protocol consists of three consecutive phases: **SA**.**Setup**, **SA**.**Protect**, and **SA**.**Agg**. Each of these phases achieves a specific task described as follows:

- **SA**.**Setup**: In this phase, the n users and the aggregator get the public parameters and the key material. The public parameters and the keys are generated either using a trusted third party ($\mathcal{TP}$) or through a distributed mechanism. At the end of this phase, each user stores a single, unique key k_i where $i \in [n]$ and the aggregator stores its aggregation key k_0.
- **SA**.**Protect**: Each user U_i locally executes a protection algorithm to protect its input $x_{i,\tau}$ at timestamp τ. The resulting protected input is sent to the aggregator(s).
- **SA**.**Agg**: Once the aggregators collect all the protected inputs, they collaboratively execute an aggregation algorithm to retrieve the sum of user inputs for timestamp τ. In the case with a single aggregator, the aggregation algorithm is locally executed by the aggregator.

3.4.2 Encryption-based SA

Encryption-based SA protocols use encryption schemes to protect the inputs of the users. Encryption utilizes a secret key to ensure the confidentiality of the user input. To achieve *Aggregator Obliviousness* (see Definition 3.2.1), users should not be allowed to encrypt their inputs with the same key. Moreover, the encryption scheme should allow the computation of the sum of the inputs over the ciphertexts without leaking the individual cleartext values. There are three types of encryption schemes that are used to build a secure aggregation protocol: (i) masking, (ii) additively homomorphic encryption (AHE), and (iii) functional encryption (FE). In general, encryption-based SAs rely on a single

[1] Baseline secure aggregation protocols correspond to the basic constructions from the corresponding cryptographic tool.

aggregator to perform the aggregation which minimizes the communication overhead of the protocol.

3.4.2.1 Secure Aggregation using Masking

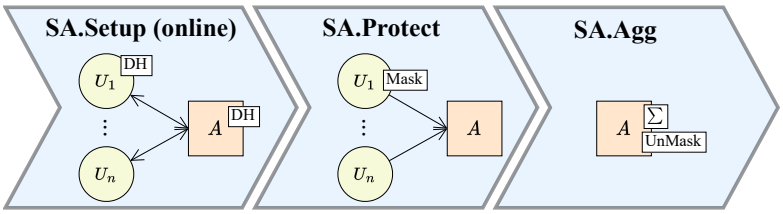


Figure 3.2: The main operations and communication between parties in Masking-based SA.

Masking is a symmetric encryption technique based on one-time pad [Rub96]. It uses modular addition to mask the data owner inputs. Given a shared key k between two parties and an upper bound R of the input, masking is defined by two algorithms:

- `Masking.Mask`$(k, x) \rightarrow c$: Masks an input x with the masking key k

$$c = x + k \mod R$$

- `Masking.UnMask`$(k, c) \rightarrow x$: Unmasks the ciphertext c using the masking key k

$$c = c - k \mod R$$

It is one of the oldest techniques for designing a secure aggregation protocol. It was first used in tree-structured networks. These schemes are called layered masking schemes [CMT05, ÖM07, CCMT09]. More recently, Dining Cryptographers network (DC-net) variants are proposed in [ÁC11, BIK+17, BBG+20, SGA21b].

Layered Masking Variant In this type of masking scheme, the users are assumed to have network connectivity with each other. Hence, the users are arranged in a tree structure. In the **SA.Setup** phase, the ones that are at a distance of h hops from each other, share the same keys (h being a security parameter). Each user representing a node in the tree runs a secure aggregation process with its children. In **SA.Protect**, each child node masks its input with the sum of the keys it holds using `Masking.Mask`. It then sends it to its parent node. In **SA.Agg**, the parent sums the masked inputs received from all its children and then removes the layers of the masks that correspond to their keys using `Masking.UnMask`. The same process is repeated from bottom to up for each parent node until the aggregated value reaches the root of the tree at which the final layers are removed. Castelluccia et al. [CMT05] proposed a specific version of this scheme when $h = \infty$. Önen et al. [ÖM07] later generalized this scheme.

DC-Net Variant In the variant, secure aggregation is seen as a variation of the dining cryptographers problem [Cha88]. The solution is realized from masking as follows: In the **SA.Setup** phase, each pair of users (U_i, U_j) agrees on a random key $k_{(i,j),\tau}$ using a *Key Agreement* protocol (ex., Diffie Hellman (DH) [DH06] using the aggregator as a proxy to forward the public keys). Also, each user U_i agrees on a random key $k_{(i,0),\tau}$ with the aggregator. As a result, each user U_i and the aggregator A computes their own unique key as follows:

$$\begin{aligned} k_{i,\tau} \leftarrow & \sum_{j=1}^{i-1} k_{(i,j),\tau} - \sum_{j=i+1}^{n} k_{(i,j),\tau} - k_{(i,0),\tau} \\ \text{s.t.} \quad & k_{(i,j),\tau} = k_{(j,i),\tau} \text{ and } k_{(i,i),\tau} = 0 \quad \forall\, i \in [0,..,n] \end{aligned} \tag{3.1}$$

In the **SA.Protect** phase, each user masks its own input $x_{i,\tau}$ with the key $k_{i,\tau}$:

$$c_{i,\tau} \leftarrow \texttt{Masking.Mask}(k_{i,\tau}, x_{i,\tau})$$

In the **SA.Agg** phase, the aggregator adds the masked inputs from all users. Then, it removes the mask using its key $k_{0,\tau}$ (all the operations are mod $R = nR_u$ where $\mathcal{Z}_{R_u}$ is the range for the input values of each user):

$$\begin{aligned} &\texttt{Masking.UnMask}(k_{0,\tau}, \sum_{i=1}^{n} c_{i,\tau}) = \sum_{i=1}^{n} c_{i,\tau} - k_{0,\tau} \\ &= \sum_{i=1}^{n} (x_{i,\tau} + \sum_{j=1}^{i-1} k_{(i,j),\tau} - \sum_{j=i+1}^{n} k_{(i,j),\tau} - k_{(i,0),\tau}) - k_{0,\tau} \\ &= \sum_{i=1}^{n} x_{i,\tau} + \sum_{i=1}^{n} (\sum_{j=1}^{i-1} k_{(i,j),\tau} - \sum_{j=i+1}^{n} k_{(i,j),\tau}) \xrightarrow{\quad} 0 \\ &\qquad\qquad - \sum_{i=1}^{n} k_{(i,0),\tau} - k_{0,\tau} \\ &= \sum_{i=1}^{n} x_{i,\tau} - \sum_{i=1}^{n} k_{(i,0),\tau} - (-\sum_{i=1}^{n} k_{(0,i),\tau}) = \sum_{1}^{n} x_{i,\tau} \end{aligned} \tag{3.2}$$

Analysis (DC-Net Variant): This scheme does not require a key dealer (KD) to distribute the masks. However, it relies on a trusted public key infrastructure (PKI). On the other hand, the masking operations are themselves very lightweight since they only include modular additions. However, the setup phase incurs significant overhead in terms of computation and communication costs per user which increases linearly with the total number of users. Since masking uses one-time pad encryption, the setup phase is performed on each timestamp τ (notice the use of the tag τ for each key). Another disadvantage is that once keys are distributed, all users should provide their protected inputs (i.e., does not support dynamic users). Indeed, if some users did not participate, the masks on the aggregated value cannot be removed. Note that masking itself is

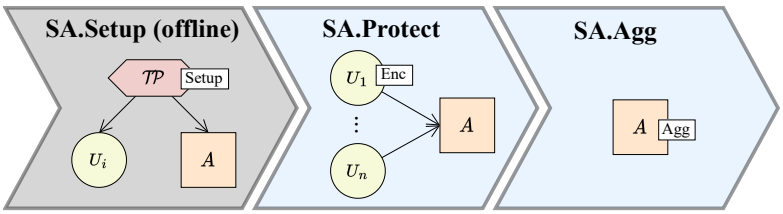


Figure 3.3: The main operations and the communication between parties in AHE-based SA.

information-theoretically secure but the setup relies on a key agreement protocol that is computationally secure.

3.4.2.2 Secure Aggregation using Additively Homomorphic Encryption

A special type of Additively Homomorphic Encryption (AHE) schemes can be used for secure aggregation. Specifically, multi-user AHE is proposed such that "addition homomorphism" property is maintained across ciphertexts generated by different users with different keys. These schemes are generally defined by the three following algorithms:

- $\texttt{AHE.Setup}(\lambda) \rightarrow (pp, \{k_i\}_{i\in[n]}, k_0)$: Given a security parameter λ, it generates the public parameters, the encryption keys, and the decryption key.
- $\texttt{AHE.Enc}(pp, k_i, \tau, x_{i,\tau}) \rightarrow c_{i,\tau}$: It encrypts message $x_{i,\tau}$ for timestamp τ using key k_i and outputs ciphertext $c_{i,\tau}$.
- $\texttt{AHE.Agg}(pp, k_0, \tau, \{c_{i,\tau}\}_{i\in[n]}) \rightarrow \sum_1^n x_{i,\tau}$: It evaluates the homomorphic operation on the n ciphertexts generated at timestamp τ. Then decrypts the resulting ciphertext using decryption key k_0.

A multi-user AHE scheme can guarantee *Aggregator Obliviousness* if each user encrypts only one input per timestamp. Several instantiations are proposed in [SCR+11, ET12, JL13, BJL16]. Multi-user AHE schemes are specifically designed for secure aggregation protocols: In the **SA.Setup** phase, $\mathcal{TP}$ runs $\texttt{AHE.Setup}(\lambda)$ and distributes the keys to the users and the aggregator. The **SA.Setup** phase is executed only once; In the **SA.Protect** phase, U_i executes $\texttt{AHE.Enc}(pp, k_i, \tau, x_{i,\tau})$ and sends the ciphertext to the aggregator; Finally, in the **SA.Agg** phase, the aggregator executes $\texttt{AHE.Agg}(pp, k_0, \tau, \{c_{i,\tau}\}_{i\in[n]})$ and retrieves the sum of the inputs.

Shi-Chan-Rieffel-Chow-Song Scheme (SCRCS) SCRCS scheme [SCR+11] is the first AHE scheme used for secure aggregation. It guarantees *Aggregator Obliviousness* based on Decisional Diffie Hellman (DDH) assumption. The three algorithms (`AHE.Setup`, `AHE.Enc`, and `AHE.Agg`) are defined as follows:

- $\texttt{AHE.Setup}(\lambda) \rightarrow (pp, \{k_i\}_{i\in[n]}, k_0)$: Given security parameter λ, it chooses generator $g \in \mathbb{G}$ where $\mathbb{G}$ is a cyclic group of prime order p for which DDH holds.

Additionally, it defines the hash function $H^{\mathbb{G}}$ (see Section 2.4.3). It also generates n random secrets $k_1, ..., k_n \in \mathbb{Z}_p$ and $k_0 = -\sum_1^n k_i$. It outputs the public parameters $pp = (\mathbb{G}, g, H^{\mathbb{G}})$, the secrets keys of each user $\{k_i\}_{i\in[n]}$, and the secret key of the aggregator k_0.

- $\texttt{AHE.Enc}(pp, k_i, \tau, x_{i,\tau}) \rightarrow c_{i,\tau}$:

$$c_{i,\tau} \leftarrow g^{x_{i,\tau}} H^{\mathbb{G}}(\tau)^{k_i}$$

- $\texttt{AHE.Agg}(pp, k_0, \tau, \{c_{i,\tau}\}_{i\in[n]}) \rightarrow \sum_1^n x_{i,\tau}$:

$$c_\tau \leftarrow \prod_1^n c_{i,\tau} = g^{\sum_1^n x_{i,\tau}} H^{\mathbb{G}}(\tau)^{\sum_1^x k_i}$$
$$V \leftarrow H(\tau)^{k_0} c_\tau = g^{\sum_1^n x_{i,\tau}} \mod n$$

Then, it computes the discrete logarithm base g of V to obtain $\sum_1^n x_{i,\tau} \bmod p$. For an efficient computation of the discrete logarithm using Pollard's method [Pol78], the output $\sum_1^n x_{i,\tau}$ should be a small number.

Theorem 3.4.1

The scheme provides *Aggregator Obliviousness* (see 3.2.1) security under the DDH assumption in the random oracle model if each user encrypts at most one value per timestamp.

Proof.

For the proof of correctness and security of this scheme, refer to [SCR+11].

Joye-Libert Scheme (JL) JL scheme [JL13] is another AHE scheme for SA which is designed as an improvement of the SCRCS scheme. JL scheme has a simpler decryption function as it does not require the computation of the discrete logarithm in a group in which the DDH assumption holds. The JL scheme guarantees *Aggregator Obliviousness* based on Decision Composite Residuosity (DCR) assumption [Pai99]. It defines the three algorithms (`AHE.Setup`, `AHE.Enc`, and `AHE.Agg`) as follows:

- $\texttt{AHE.Setup}(\lambda) \rightarrow (pp, \{k_i\}_{i\in[n]}, k_0)$: Given security parameter λ, it generates randomly two equal-size prime numbers p and q and sets $N = pq$. Then, it defines a cryptographic hash function $H^{\mathbb{Z}^*_{N^2}}$ (see Section 2.4.3). It randomly generates n secret keys $\{k_i\}_{i\in[n]} \in \{0,1\}^{2l}$ and sets $k_0 = -\sum_1^n k_i$. It outputs the public parameters $pp = (N, H^{\mathbb{Z}^*_{N^2}})$, the secrets keys of each user $\{k_i\}_{i\in[n]}$, and the secret key of the aggregator k_0.

- $\texttt{AHE.Enc}(pp, k_i, \tau, x_{i,\tau}) \rightarrow c_{i,\tau}$:

$$c_{i,\tau} \leftarrow (1 + x_{i,\tau}N) H^{\mathbb{Z}^*_{N^2}}(\tau)^{k_i} \mod N^2$$

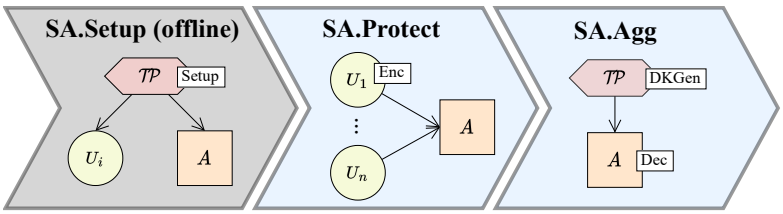


Figure 3.4: The main operations and the communication between parties in FE-based SA.

- $\texttt{AHE.Agg}(pp, k_0, \tau, \{c_{i,\tau}\}_{i\in[n]}) \rightarrow \sum_1^n x_{i,\tau}$:

$$c_\tau \leftarrow \prod_1^n c_{i,\tau} = (1 + N\sum_1^n x_{i,\tau}) H^{\mathbb{Z}^*_{N^2}}(\tau)^{\sum_1^n k_i} \mod N^2$$

$$\frac{H^{\mathbb{Z}^*_{N^2}}(\tau)^{k_0} c_\tau - 1}{N} = \sum_1^n x_{i,\tau} \mod N$$

Theorem 3.4.2

The scheme provides *Aggregator Obliviousness* (see 3.2.1) security under the DCR assumption in the random oracle model if each user encrypts at most one value per timestamp.

Proof.

For the proof of correctness and security of this scheme, refer to [JL13].

Analysis: The main advantage of AHE schemes is that they require running the setup phase only one time, and hence they are effective when aggregating a stream of data. This originally comes with the cost of relying on a trusted key dealer (KD) to perform the setup. Nevertheless, previous work has improved these schemes to enable running them without the need for a key dealer [LEM14b]. The per-user computational cost resulting from **SA.Protect** does not depend on the total number of users but remains significant. Similarly, the communication cost per user does not depend on the total number of users but incurs a size expansion because of the size of the ciphertext. Additionally, similar to masking schemes, AHE does not support dynamic users since all users should provide their inputs to correctly aggregate them.

3.4.2.3 Secure Aggregation using Functional Encryption

Functional encryption (FE) is a type of encryption scheme that enables a user to learn a function on the encrypted data [BSW11]. Multi-Input Function Encryption (MIFE), introduced by Goldwasser [GGG+14], enables the learning of a function over multiple

encrypted inputs. A special type of MIFE scheme can be designed to compute the inner product function of multiple inputs [AGRW17, ACF+18a, DOT18]. Assuming that we have two vectors x and y, each consisting of l elements, the inner product of x and y is as follows:

$$IP(x, y) = \sum_{i=1}^{\ell} x_i y_i \tag{3.3}$$

An inner product MIFE scheme is defined by four algorithms:

- $\texttt{MIFE.Setup}(\lambda) \rightarrow (pp, msk, \{k_i\}_{i\in[n]})$: Given a security parameter λ, it generates the public parameters pp, master secret key msk, and n user keys $\{k_i\}_{i\in[n]}$.
- $\texttt{MIFE.Enc}(pp, k_i, x_{i,\tau}) \rightarrow c_{i,\tau}$: It encrypts message $x_{i,\tau}$ using key k_i and outputs ciphertext $c_{i,\tau}$.
- $\texttt{MIFE.DKGen}(pp, msk, y_\tau) \rightarrow dk_\tau$: It generates decryption key dk_τ using the master secret key and vector y_τ of n elements.
- $\texttt{MIFE.Dec}(pp, dk_\tau, c_\tau, y_\tau) \rightarrow IP(x_\tau, y_\tau)$: It takes vector $c_\tau = [c_{1,\tau}, .., c_{n,\tau}]$, vector y_τ, and decryption key dk_τ generated from y_τ. It decrypts c_τ such that the result is the inner product of $x_\tau = [x_{1,\tau}, .., x_{n,\tau}]$ and y_τ.

MIFE schemes for inner product can be used to construct a secure aggregation protocol [XBZ+19, WPX+20]. In the **SA.Setup** phase, $\mathcal{TP}$ runs $\texttt{MIFE.Setup}(\lambda)$ and distributes the keys to the users. In the **SA.Protect** phase, U_i executes $\texttt{MIFE.Enc}(pp, k_i, x_{i,\tau})$ and sends the ciphertext $c_{i,\tau}$ to the aggregator. Finally, in the **SA.Agg** phase, the aggregator first sends vector $y_\tau = [1, ..., 1]$ to $\mathcal{TP}$ which executes $\texttt{MIFE.DKGen}(pp, msk, y_\tau)$ and sends decryption key dk_τ of timestamp τ to the aggregator. The aggregator then executes $\texttt{MIFE.Dec}(pp, dk_\tau, [c_{1,\tau}, .., c_{n,\tau}], y_\tau)$ and retrieves the inner product $\sum_{i=1}^{n} x_{i,\tau} y[i] = \sum_{1}^{n} x_{i,\tau}$.

Construction based on one-time pad An efficient FE-based secure aggregation can be built using one-time pad encryption and a pseudo-random generator, only. The MIFE scheme from [ACF+18b] defines the four algorithms (`MIFE.Setup`, `MIFE.Enc`, `MIFE.DKGen`, and `MIFE.Dec`) as follows:

- $\texttt{MIFE.Setup}(\lambda) \rightarrow (pp, msk, \{k_i\}_{i\in[n]})$: Given security parameter λ, it chooses prime p and a pseudo-random generator $\texttt{PRG} : (\mathbb{Z}_p, \mathbb{Z}) \rightarrow \mathbb{Z}_p$ as the public parameter then chooses n user keys at random $k_i \leftarrow \mathbb{Z}_p \forall i \in [n]$. Finally it sets the master key $msk = \{k_i\}_{i\in[n]}$.
- $\texttt{MIFE.Enc}(pp, k_i, x_{i,\tau}) \rightarrow c_{i,\tau}$: It first computes $k_{i,\tau} \leftarrow \texttt{PRG}(k_i, \tau)$ then $c_{i,\tau} \leftarrow x_{i,\tau} + k_{i,\tau} \mod p$

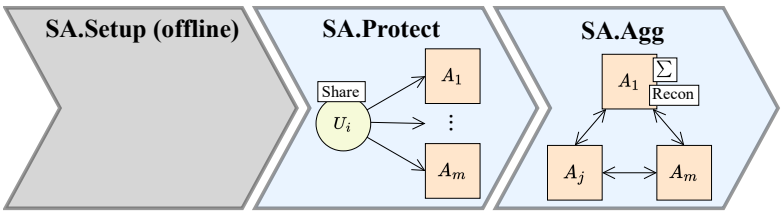


Figure 3.5: The main operations and communication between parties in MPC-based SA.

- $\texttt{MIFE.DKGen}(pp, msk, y_\tau) \rightarrow dk_\tau$:

$$dk_\tau \leftarrow \sum_{\forall k_i \in msk} y_{i,\tau} \texttt{PRG}(k_i, \tau) \mod p$$

- $\texttt{MIFE.Dec}(pp, dk_\tau, c_\tau, y_\tau) \rightarrow IP(x_\tau, y_\tau)$:

$$IP(c_\tau, y_\tau) - dk_\tau \mod p$$

Analysis: Similar to AHE schemes, MIFE-based SA incurs constant computation and communication costs per user with respect to the total number of users. A very important property of these schemes is that they can deal with dynamic users by replacing zero weights in vector y_τ for the users that do not provide input at timestamp τ. On the other hand, the disadvantage of these schemes is that they require an online key dealer (KD) as a trusted third party to generate the decryption key for each timestamp.

3.4.3 MPC-based SA

Another cryptographic tool used to build secure aggregation protocols is multi-party computation (MPC). In MPC, keys are not needed to protect user inputs. Instead, private messages are split into shares and distributed to multiple servers such that t of them can collaborate to reconstruct them. The secret sharing scheme presented in Section 2.3.1 can be used to construct a secure aggregation protocol.

To design a secure aggregation protocol from MPC, the **SA**.**Setup** phase is not needed since no keys are generated. In the **SA**.**Protect** phase, a user protects its input by splitting it into l random shares using $\texttt{SSS.Share}(x_{i,\tau}, t, l, \mathbb{Z}_p)$ where l is the number of aggregators and p is the first prime number such that $p \geq nR_u$. It then sends one unique share to each aggregator. In the **SA**.**Agg** phase, each aggregator locally sums up the shares. Because secret sharing is additively homomorphic, the sum of the shares will result in a share of the sum. Finally, at least t aggregators broadcast their shares of the sum so that any aggregator can then run $\texttt{SSS.Recon}$ and retrieve the sum $\sum_1^n x_{i,\tau}$.

Analysis: An important property of MPC-based SA is that it does not need a trusted third party since it does not need a key setup phase. Also, MPC supports dynamic users since it allows any subset of users to participate in the aggregation. This is mainly because MPC does not rely on secret keys that uniquely identify a user. On the contrary,

MPC incurs high computation and communication costs since the protection of a user input involves creating $O(nm)$ shares where n is the number of users and m is the size of the input. Furthermore, to distribute the shares, pre-existing secure channels are needed between the users and the aggregators. The secure channels ensure that each share is received and accessed only by its destined aggregator.

3.5 Summary of Honest-but-Curious Secure Aggregation

	Schemes	$\mathcal{TP}$ required	No SC required	Dynamic users	Comp.	Comm.
Encryption	**Masking** (DC-net) [ÁC11, DA16]	PKI	●	○	$O(n+m)$	$O(n+m)$
	AHE [LEM14a, JL13, SCR+11]	KD	●	○	$O(m)$	$O(m)$
	FE [ACF+18b, ABM+20, LT19]	KD	●	●	$O(m)$	$O(m)$
MPC	**n-out-of-n SS** [BSMD10, GJ11] **t-out-of-n SS** [Sha79]	-	○	●	$O(nm)$	$O(nm)$

Table 3.2: Table comparing the baseline constructions of the different categories of secure aggregation. $\mathcal{TP}$ stands for trusted third party. **SC** stands for pre-established secure channels. **Dynamic users** property shows whether the aggregation can be performed with a subset of the users, only. **Comp.** stands for computation cost on users. **Comm.** stands for communication cost between users and aggregators. n and m represent the number of users and the size of the user's input, respectively.

In this chapter, we introduced secure aggregation protocols by defining their three phases of execution. Then, we surveyed existing solutions and presented them as instantiations of our definitions. We categorized these solutions into encryption-based solutions (Masking, AHE, FE) and MPC-based solutions and we compared their pros and cons. We found out that secure aggregation based on MPC can offer less strict setup assumptions and allow user dynamics. However, this comes at a cost of higher computation and communication costs. On the other hand, secure aggregation based on additively homomorphic encryption can perform with much better scalability in terms of the number of users. However, the latter requires a strong setup assumption where a key dealer is needed to initialize the protocol. Secure aggregation based on masking serves as a mid-way between AHE and MPC solutions. Finally, all properties can be achieved by secure aggregation based on functional encryption, but it requires a very strong assumption that the trusted key dealer should stay online during the protocol execution.

This chapter tackled the solutions in the honest-but-curious threat model only. Unfortunately, the adversary in many real-life applications is more powerful. Therefore, these schemes provide a good starting point to build solutions that can achieve stronger and more realistic threat models. In the next chapters, we focus on secure aggregation in the malicious model and we propose a new secure aggregation protocol.

Chapter 4

Secure Aggregation in the Malicious Model

In this chapter, we study secure aggregation protocols when the aggregator and a subset of the users are considered malicious. We review some of the previous work and then we propose a new verifiable secure aggregation protocol (VSA). We first formalize the security definitions of a verifiable secure aggregation which capture stronger security guarantees than previous work. Then we construct our protocol VSA that satisfies these security definitions.

4.1 Secure Aggregation with Malicious Parties

In the malicious model, the adversary controlling the aggregator and a subset of the users may deviate from the protocol steps. This model represents more realistic scenarios than the honest-but-curious model. For example, in smart grid systems, the adversary may control some of the smart meters and additionally succeed in compromising the aggregation server. In such a case, we need to guarantee that the electricity consumption of honest users remains private. Moreover, we need to guarantee that the adversary is detected if the aggregation server produces incorrect statistics about the total electricity consumption.

In general, a secure aggregation protocol in the malicious model involves some verification mechanism to ensure *Aggregate Unforgeability* in addition to *Aggregator Obliviousness* (see Definitions 3.2.1 and 3.2.2). For this purpose, we propose a formal definition for verifiable secure aggregation protocols which capture both the privacy and verifiability of the protocol. Moreover, we present in this chapter the first verifiable secure aggregation protocol (VSA) that meets our definition.

4.2 Previous Work

Most of the existing work focus either on the malicious aggregator model [LEÖM15, TLB+21, HKKH21] or the malicious users model [KOB21, CGB17, BLV+21].

Secure Aggregation against Malicious Aggregator All the solutions in the malicious aggregator model share the idea of using tags created by the users to authenticate the sum of the inputs. More precisely, the user sends a tag along with its protected input. Then, the aggregator adds all the tags to produce a final tag that proves the correctness of the sum. There are mainly two types of tags proposed in the literature:

- Hash-based tags: In DEVA [TLB+21] and VERSA [HKKH21], the authors use homomorphic hashes [YWHZ18, KFM04] as tags. The problem with the hash approach is that the tags a need to be authenticated and saved on a public bulletin board. The public verifier later aggregates the hashes and verifies the sum. This approach leads to a linear increase in the size of the tags with respect to the number of users.

- Signature-based tags: In PUDA [LEÖM15], the authors modify the SCRCS secure aggregation scheme (see Section 3.4.2.2) to provide a verification mechanism of the aggregate. PUDA users generates homomorphic tags based on a homomorphic signature scheme. The homomorphic signature scheme is an extension of the signature scheme in [Fre12a] to the multi-user setting. The advantage of PUDA over hash-based tags is the succinctness of the aggregated tag (i.e., constant size with respect to the number of users). We present PUDA scheme in details in Section 4.4.1 since it is one of the building blocks for VSA.

The problem with these solutions is that they do not consider the case where one or more users are corrupted. In such a case, the adversary who controls both the aggregator and one user can forge a tag for any arbitrary value. Therefore, these solutions are limited to a few applications.

Secure Aggregation against Malicious Users Another type of secure aggregation protocols target applications where the user is not trusted. In these protocols, the user produces a tag that proves to the aggregator that the input is within a valid range. To generate these tags, Karakoç et al. propose KOB [KOB21] which uses an Oblivious Programable Pseudo-Random Function (OPPRF) protocol. The protocol runs between the user which provides its input and the aggregator which provides an upper-bound value. If the user input is less than the upper bound chosen by the aggregator, the user receives a valid tag of its input. They build their OPPRF using an OT protocol. More specifically, the user and the aggregator perform a secure comparison of their inputs using ℓ OT executions (ℓ is the bit-size of the input). In each run, the aggregator sends a masked part of the tag. After executing all the OTs, the user can reconstruct the tag and unmask it if its input is less than the aggregator's upper bound. This solution ensures that a malicious user cannot provide out-of-bound input and thus limits their influence on the sum of all users' inputs.

Unfortunately, in this solution, the aggregator is trusted to perform the check on the user inputs. Additionally, in the case of a malicious aggregator, this solution cannot ensure privacy. Indeed, the malicious aggregator can compute an arbitrary function of the user input instead of computing the tag. Therefore, this secure aggregation solution is insecure when the aggregator is controlled by the adversary.

Secure Aggregation against both Malicious Users and Aggregator There is only one solution that studies this extreme case where both the aggregator and the users are malicious. Guo et al. propose a solution [GLL+21] that considers the case where the aggregator is malicious and it colludes with a subset of the user. However, the authors assume that the users provide correct inputs. In their solution, the users commit to their inputs using a homomorphic commitment scheme [Ped92] and send the commitments to all the other users before the aggregation process starts. After the users receive the commitments, they send their protected inputs to the aggregator. This solution does not prevent the adversary from providing bad inputs and changing the sum. Instead, it only guarantees that the adversary can influence the aggregation result before knowing the actual sum of the honest users. Hence, this solution does not provide strong security guarantees. Therefore, we identify the need for a strong definition of security for secure aggregation in the case of malicious parties as a missing requirement.

4.3 VSA - Overview

We aim to build a verifiable secure aggregation protocol that ensures both *Aggregator Obliviousness* and *Aggregate Unforgeability* in the malicious model. Thus, we propose to extend PUDA [LEÖM15] (which is secure in the malicious aggregator model) to achieve security when both the aggregator and a subset of users are malicious. The main problem with PUDA is that users are trusted in generating tags for their inputs. Thus, a malicious user can produce tags for any input and thus compromise the sum while still having a valid tag. Alternatively, KOB protocol [KOB21] allows the aggregator to generate the tags instead of the user after privately checking the correctness of the user inputs. However, this approach requires the aggregator to be trusted. Therefore, our idea is to involve additional parties in the protocol called taggers. The purpose of the taggers is to generate user tags. More precisely, each user runs a tagging protocol with the taggers. If the user input is less than the upper bound set by each tagger, the user receives a PUDA-like tag. Then, the user continues as in PUDA protocol but instead of generating its own tag, it uses the tag computed by the taggers. The aggregator receives the protected input and the tag of each user and finally outputs the sum and its corresponding tag. In our protocol VSA, the final tag ensures both, that the users provided well-formed inputs, and that the aggregator correctly computed the sum.

One challenge in building VSA is building a secure and robust tagging protocol that even when some taggers (less than a threshold) are malicious, the tag can still be produced. To build this protocol we use threshold secret sharing to share the tagging keys among the taggers. Each tagger computes a partial tag using the key share. The user robustly aggregates the partial tags under the assumption that there is a sufficient number of honest taggers.

One more challenge to deal with is how each tagger privately checks that the user input is in the correct range. KOB proposes an OPPRF approach for this purpose. However, this solution is only secure when the tagger is honest-but-curious since it allows the tagger to compute any function of the user input instead of computing the tag. To solve this problem, we design our tagging protocol using zero-knowledge proofs to ensure

correct behavior of the tagger.

4.4 PUDA secure aggregation

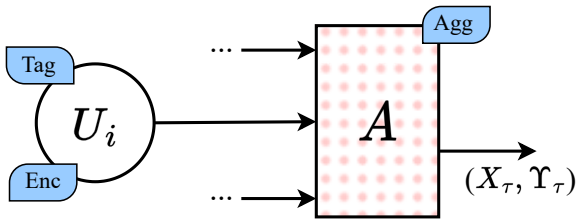


Figure 4.1: PUDA protocol. The aggregator is malicious and users are honest.

Leontiadis et al. presented PUDA [LEÖM15] as a model for aggregating inputs from n users using a single aggregator. We describe this model and its properties. Then, we present the construction proposed by the authors.

4.4.1 PUDA model

The PUDA model involves n users, one aggregator, and a trusted key dealer. It consists of the five following algorithms:

- $\texttt{PUDA.Setup}(1^\lambda)$: An interactive randomized algorithm which on the input of security parameter λ, generates public parameters $\texttt{pp}$, an encryption and tagging key $(\texttt{ek}_i, \texttt{tk}_i)$ for user U_i, an aggregator key $\texttt{ak}$, and a public verification key $\texttt{vk}$.
- $\texttt{PUDA.Enc}(\texttt{ek}_i, x_i, \tau) \rightarrow c_i$: It encrypts the private value x_i using key $\texttt{ek}_i$ for timestamp τ.
- $\texttt{PUDA.Tag}(\texttt{tk}_i, x_i, \tau) \rightarrow \sigma_i$: It generates a tag for private value x_i using tagging key $\texttt{tk}_i$ for timestamp τ.
- $\texttt{PUDA.Agg}(\texttt{ak}, \{c_i\}_{\forall i \in [n]}, \{\sigma_i\}_{\forall i \in [n]}, \tau) \rightarrow (X_\tau, \Upsilon_\tau)$: It aggregates the ciphertext and the tags of all users for timestamp τ using the aggregation key $\texttt{ak}$. It outputs the sum X_τ and the corresponding tag Υ_τ.
- $\texttt{PUDA.Verify}(\texttt{vk}, X_\tau, \Upsilon_\tau, \tau) \rightarrow \{0, 1\}$: It uses the verification key $\texttt{vk}$ to verify the tag Υ_τ of the sum X_τ for timestamp τ.

The authors define security using the two notions of *Aggregator Obliviousness* and *Aggregate Unforgeability* (see Section 3.2). They propose a formal definition of these notions using security games $\text{AO}^{\texttt{PUDA}}$ and $\text{AU}^{\texttt{PUDA}}$ respectively[1].

[1]For the formal definitions, please refer to the original paper.

4.4.2 PUDA construction

PUDA construction is based on SCRCS secure aggregation scheme (see Section 3.4.2.2). It is described as follows:

- $\texttt{PUDA.Setup}(1^\lambda)$:
  - The trusted key dealer generates the public parameters: It selects the groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ (where DDH holds) of prime order p; the generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$; a bilinear map $e : \{\mathbb{G}_1, \mathbb{G}_2\} \to \mathbb{G}_T$; and a hash function $H^{\mathbb{G}_1}$ (see Section 2.4.3). It gives $\texttt{pp} = (g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, H^{\mathbb{G}_1})$ to all users and the aggregator.
  - The key dealer generates a uniformly random encryption key per user $\texttt{ek}_i \leftarrow\!\!\$\ \mathbb{Z}_p$. It sets the aggregation key $\texttt{ak} = -\sum_{\forall i \in [n]} \texttt{ek}_i$. It gives $\texttt{ek}_i$ for each user and $\texttt{ak}$ to the aggregator.
  - The key dealer generates a uniformly random key a and sends it to all the users. Then, each user generates a uniformly random key $b_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and forwards $g_2^{b_i}$ to the key dealer. The user sets $\texttt{tk}_i = (a, b_i)$ as its tagging key.
  - The key dealer computes $\texttt{vk}_1 = g_2^a$ and $\texttt{vk}_2 = \prod_1^n g_2^{b_i} = g_2^{\sum_1^n b_i}$ and then sets the verification key as $\texttt{vk} = (\texttt{vk}_1, \texttt{vk}_2)$
- $\texttt{PUDA.Enc}(\texttt{ek}_i, x_i, \tau)$:
  The user encrypts x_i for timestamp τ as follows:

$$c_i \leftarrow (g_1^{x_i}) H^{\mathbb{G}_1}(\tau)^{\texttt{ek}_i}$$

- $\texttt{PUDA.Tag}(\texttt{tk}_i, x_i, \tau)$:
  The user parses $\texttt{tk}_i{=}(a, b_i)$ and generates a tag of x_i for timestamp τ as follows:

$$\sigma_i \leftarrow (g_1^a)^{x_i} H^{\mathbb{G}_1}(\tau)^{b_i}$$

- $\texttt{PUDA.Agg}(\texttt{ak}, \{c_i\}_{\forall i \in [n]}, \{\sigma_i\}_{\forall i \in [n]}, \tau)$:
  The aggregator aggregates the ciphertexts and the tags from each user:

$$V_\tau \leftarrow (\prod_{i \in [n]} c_i) H^{\mathbb{G}_1}(\tau)^{\texttt{ak}} = g_1^{X_\tau} \qquad \Upsilon\tau \leftarrow \prod_{i \in [n]} \sigma_i = (g_1^a)^{X_\tau} H^{\mathbb{G}_1}(\tau)^{\sum_{i \in [n]} b_i}$$

  It then finds $X_\tau = \sum_{i \in [n]} x_i$ by computing the discrete logarithm of V_τ.
- $\texttt{PUDA.Verify}(\texttt{vk}, X_\tau, \Upsilon_\tau, \tau)$:

$$e(\Upsilon_\tau, g_2) \stackrel{?}{=} e(H^{\mathbb{G}_1}(\tau), \texttt{vk}_1) e(g_1^{X_\tau}, \texttt{vk}_2)$$

4.5 VSA - Formal Definitions

In this section, we present the VSA model and its security properties. VSA is composed of n users $\mathcal{U} = \{U_1, ..., U_n\}$, m taggers $\mathcal{T} = \{T_1, ..., T_m\}$, and one aggregator A. The users generate inputs at each timestamp τ. The taggers are parties that compute the tag of a user's inputs at a given timestamp. Finally, the aggregator is the party that produces the sum of all inputs and its corresponding authenticating tag for a given timestamp. VSA is composed of the following PPT algorithms:

- $\texttt{VSA.Setup}(1^\lambda, t)$: An interactive randomized algorithm which on input of security parameter λ and threshold $t \leq m$, generates the public parameters $\texttt{pp}$, an encryption key $\texttt{ek}_i$ for user U_i, an aggregator key $\texttt{ak}$, a list of n tagging keys $\texttt{TK}_j = \{\texttt{tk}_{1,j}, ..., \texttt{tk}_{n,j}\}$ (one key per user) for each tagger T_j, and a public verification key $\texttt{vk}$.
- $\texttt{VSA.Enc}(\texttt{ek}_i, x_i, \tau) \rightarrow c_i$: It encrypts the private value x_i using key $\texttt{ek}_i$ for timestamp τ.
- $\texttt{VSA.Tag}(\{\texttt{tk}_{i,j}\}_{j \in J_i}, \{\theta_j\}_{j \in J_i}, x_i, \tau) \rightarrow \sigma_i$: An interactive randomized algorithm which takes an input x_i, a higher bound on the inputs θ, a subset of tagging keys $\{\texttt{tk}_{i,j}\}_{j \in J_i}$, and a timestamp τ. It outputs a tag σ_i.
- $\texttt{VSA.Agg}(\texttt{ak}, \{c_i\}_{\forall i \in [n]}, \{\sigma_i\}_{\forall i \in [n]}, \tau) \rightarrow (X_\tau, \Upsilon_\tau)$: It aggregates the ciphertext and the tags of all users for timestamp τ using the aggregation key $\texttt{ak}$. It outputs the sum X_τ and a tag Υ_τ.
- $\texttt{VSA.Verify}(\texttt{vk}, X_\tau, \Upsilon_\tau, \tau) \rightarrow \{0, 1\}$: It uses the verification key $\texttt{vk}$ to verify the tag Υ_τ of the sum X_τ for timestamp τ.

The main difference between VSA model and PUDA model is due to the additional parties called taggers. Moreover, each user's tagging key is distributed to the m taggers and not to the users. Consequently, the tagging algorithm becomes an interactive protocol between the user and the taggers to produce the tag of the user's input.

4.5.1 Correctness of VSA

We require that when all users provide inputs that are less than all the bounds set by the participating taggers, and when the number of taggers that participate in producing the tag of each user is higher than a threshold t, the verification algorithm outputs 1 with an overwhelming probability. More formally:

$$\forall \lambda, \forall t, \forall J_i, \forall x_i, \text{ and } \forall \theta_j \text{ s.t. } i \in [n], j \in J_i, J_i \subset [m], |J_i| \geq t, \text{ and } x_i \leq \theta_j,$$

$$\Pr\left[\texttt{VSA.Verify}(\texttt{vk}, X_\tau, \Upsilon_\tau) = 0 \;\middle|\; \begin{array}{l} \texttt{VSA.Setup}(1^\lambda) \rightarrow (\texttt{pp}, \{\texttt{ek}_i\}, \{\texttt{tk}_{i,j}\}, \texttt{ak}, \texttt{vk}), \\ \texttt{VSA.Enc}(\texttt{ek}_i, x_i, \tau) \rightarrow c_{i,\tau} \quad \forall i \in [n], \\ \texttt{VSA.Tag}(\{tk_{i,j}\}_{j \in J_i}, \{\theta_j\}_{j \in J_i}, x_i, \tau) \rightarrow \sigma_{i,\tau} \quad \forall i \in [n], \\ \texttt{VSA.Agg}(\texttt{ak}, \{c_i\}_{i \in [n]}, \{\sigma_i\}_{i \in [n]}, \tau) \rightarrow (X_\tau, \Upsilon_\tau) \end{array}\right] \leq \mu(\lambda)$$

where μ is a negligible function.

4.5.2 Security of VSA

We consider that the adversary controls the aggregator A, a set of taggers $\mathcal{T}_{\texttt{corr}} \subset \mathcal{T}$, and a set of users $\mathcal{U}_{\texttt{corr}} \subset \mathcal{U}$. We present the oracle $\mathcal{O}_{\texttt{EncTag}}$ that we use to define the security games. Throughout the games, we consider that the minimal upper bound value $\theta_{\texttt{min}} = \texttt{min}(\{\theta_j\}_{\forall j \in [m]})$ is known to the adversary. .

- $\mathcal{O}_{\texttt{EncTag}}$: When queried with a user identifier i, input x, and timestamp τ, the oracle first checks if i and τ are queried before. If so, it aborts. The oracle computes c_i and σ_i as $\texttt{VSA.Enc}(\texttt{ek}_i, x, \tau)$ and $\texttt{VSA.Tag}(\{\texttt{tk}_{i,j}\}_{\forall j \in [m]}, \{\theta_{\texttt{min}}, ..., \theta_{\texttt{min}}\}, x, \tau)$ respectively. Then, if $U_i \notin \mathcal{U}_{\texttt{corr}}$, it outputs the result (c_i, σ_i), otherwise it outputs (σ_i).

Aggregator Obliviousness (AO) We give a formal definition of AO using a security game. Our game is an improved version of $\text{AO}^{\texttt{PUDA}}$ game from PUDA [LEÖM15] that captures (in addition to the static corruption of the aggregator and some users) **the corruptions of some taggers**. The game proceeds in three phases:

Setup and Corruption Phase: The adversary $\mathcal{A}$ chooses the security parameter λ and the threshold $t \leq m$ and accordingly gets the public parameters of the protocol $\texttt{pp}$, the aggregator's key $\texttt{ak}$, the verification key $\texttt{vk}$, the secret keys of the parties in $\mathcal{U}_{\texttt{corr}}$ (i.e., $\{\texttt{ek}_i\}_{\forall U_i \in \mathcal{U}_{\texttt{corr}}}$), and the secret keys of the parties in $\mathcal{T}_{\texttt{corr}}$ (i.e., $\{\texttt{TK}_j\}_{\forall T_j \in \mathcal{T}_{\texttt{corr}}}$).

Learning Phase: The adversary $\mathcal{A}$ interacts with oracle $\mathcal{O}_{\texttt{EncTag}}$ in polynomial times by sending tuples (i, x, τ) such that for every user identifier i, if $U_i \notin \mathcal{U}_{\texttt{corr}}$, then $x \leq \theta$.

Challenge Phase: The adversary $\mathcal{A}$ chooses a timestamp τ^* that has never been queried, and a set of non-corrupted users $\mathcal{U}^*$ (i.e., $\mathcal{U}^* \cap \mathcal{U}_{\texttt{corr}} = \phi$). It chooses two lists $\mathcal{X}^0_{\tau^*}$ and $\mathcal{X}^1_{\tau^*}$ each corresponds to the inputs for each user in $\mathcal{U}^*$ at τ^*. It queries the oracle $\mathcal{O}_{\texttt{AO}}$ with these lists which is defined as follows:

- $\mathcal{O}_{\texttt{AO}}$: The oracle is called with the a set of users $\mathcal{U}^* \subset \mathcal{U} \setminus \mathcal{U}_{\texttt{corr}}$ and two list of inputs $(i, x^0_i, \tau)_{\forall U_i \in \mathcal{U}^*}$ and $(i, x^1_i, \tau)_{\forall U_i \in \mathcal{U}^*}$ such that $\sum_{\forall U_i \in \mathcal{U}^*} x^0_i = \sum_{\forall U_i \in \mathcal{U}^*} x^1_i$. It flips a coin $b \in \{0, 1\}$ and returns the ciphertexts and tags of the input $(i, x^b_i, \tau)_{\forall U_i \in \mathcal{U}^*}$.

After receiving the ciphertexts $\{c^b_i\}_{\forall U_i \in \mathcal{U}^*}$ and their corresponding tags $\{\sigma^b_i\}_{\forall U_i \in \mathcal{U}^*}$, $\mathcal{A}$ submits a guess bit b^* and wins the game if $b^* = b$.

[1]This does not affect the security proof as it gives in a general sense more power to the adversary

Game 1 Aggregator Obliviousness (AO)

The Learning Phase:

/* $\mathcal{A}$ queries the oracle poly-number of times for any i, x, and τ s.t. $U_i \notin \mathcal{U}_{\texttt{corr}}$ and $x \leq \theta$ */
$(c_i, \sigma_i) \leftarrow \mathcal{O}_{\texttt{EncTag}(i,x,\tau)}$
/* $\mathcal{A}$ queries the oracle poly-number of times for any i, x, and τ s.t. $U_i \in \mathcal{U}_{\texttt{corr}}$ */
$(\sigma_i) \leftarrow \mathcal{O}_{\texttt{EncTag}(i,x,\tau)}$

The Challenge Phase:

$\mathcal{A} \rightarrow \tau^*, \mathcal{U}^* \subset \mathcal{U} \setminus \mathcal{U}_{\texttt{corr}}$
$\mathcal{A} \rightarrow \mathcal{X}^0_{\tau^*}, \mathcal{X}^1_{\tau^*}$
$\{(c_i^b, \sigma_i^b)\}_{\forall U_i \in \mathcal{U}^*} \leftarrow \mathcal{O}_{\texttt{AO}(\mathcal{X}^0_{\tau^*}, \mathcal{X}^1_{\tau^*})}$
$\mathcal{A} \rightarrow b^* \in \{0, 1\}$

Definition 4.5.1

Let $Pr[\mathcal{A}^{\text{AO}}]$ denote the probability that the adversary $\mathcal{A}$ outputs $b^* = b$. Then, VSA is said to ensure Aggregator Obliviousness if for any poly-bounded adversary $\mathcal{A}$ the probability $Pr[\mathcal{A}^{\text{AO}}] \leq \frac{1}{2} + \mu(\lambda)$, where $\mu(\lambda)$ is a negligible function and λ is the security parameter.

Aggregate Unforgeability (AU) We give a formal definition of AU using a security game. Our game is an improved version of AU$^{\texttt{PUDA}}$ game that captures (in addition to the static corruption of the aggregator) **the corruptions of some users and taggers**.

Similar to the previous definition (i.e., AO) the game proceeds in three phases. The **Setup and Corruption Phase** and **Learning Phase** are the same as in AO game. In the **Challenge Phase** $\mathcal{A}$ submits a tuple $(\tau^*, \texttt{sum}_{\tau^*}, \Upsilon_{\tau^*})$.

Game 2 Aggregate Unforgeability (AU)

The Learning Phase:

/* $\mathcal{A}$ queries the oracle poly-number of times for any i, x, and τ s.t. $U_i \notin \mathcal{U}_{\texttt{corr}}$ and $x \leq \theta$ */
$(c_i, \sigma_i) \leftarrow \mathcal{O}_{\texttt{EncTag}(i,x,\tau)}$
/* $\mathcal{A}$ queries the oracle poly-number of times for any i, x, and τ s.t. $U_i \in \mathcal{U}_{\texttt{corr}}$ */
$(\sigma_i) \leftarrow \mathcal{O}_{\texttt{EncTag}(i,x,\tau)}$

The Challenge Phase:

$\mathcal{A} \rightarrow (\tau^*, \texttt{sum}_{\tau^*}, \Upsilon_{\tau^*})$

The adversary wins the game if it succeeds in forging a valid tag of a sum. In [LEÖM15], the authors define two types of forgeries. We reconsider these types and add a new type.

- **Type I Forgery**: $\texttt{VSA.Verify}(\texttt{vk}, \texttt{sum}_{\tau^*}, \Upsilon_{\tau^*}) = 1$ and the adversary $\mathcal{A}$ never made a previous query with timestamp τ^*.

- **Type II Forgery**: $\texttt{VSA.Verify}(\texttt{vk}, \texttt{sum}_{\tau^*}, \Upsilon_{\tau^*}) = 1$ and $\mathcal{A}$ already made queries with timestamp τ^*, however the sum $\texttt{sum}_{\tau^*} \neq \sum_{\forall U_i \in \mathcal{U}} x_{i,\tau^*}$.
- **Type III Forgery**: $\texttt{VSA.Verify}(\texttt{vk}, \texttt{sum}_{\tau^*}, \Upsilon_{\tau^*}) = 1$ and $\mathcal{A}$ made queries with timestamp τ^*, however the sum $\texttt{sum}_{\tau^*} > \sum_{\forall U_i \in \mathcal{U} \setminus \mathcal{U}_{\texttt{corr}}} x_{i,\tau^*} + |\mathcal{U}_{\texttt{corr}}|\theta_{\texttt{min}}$.

Definition 4.5.2

Let $Pr[\mathcal{A}_{\text{I}}^{\text{AU}}]$, $Pr[\mathcal{A}_{\text{II}}^{\text{AU}}]$, and $Pr[\mathcal{A}_{\text{III}}^{\text{AU}}]$ denote the probability that the adversary $\mathcal{A}$ outputs a tuple that satisfies **Type I**, **II**, and **III Forgery** respectively. Then, VSA is said to ensure Aggregate Unforgeability if for any poly-bounded adversary $\mathcal{A}$ the probability $Pr[\mathcal{A}^{\text{AU}}] = Pr[\mathcal{A}_{\text{I}}^{\text{AU}}] + Pr[\mathcal{A}_{\text{II}}^{\text{AU}}] + Pr[\mathcal{A}_{\text{III}}^{\text{AU}}] \leq \mu(\lambda)$, where $\mu(\lambda)$ is a negligible function and λ is the security parameter.

Definition 4.5.2 features stronger security guarantees than PUDA. First, it captures the **Type I and II Forgeries** that represent a corrupted aggregator forcing an incorrect sum, but this time, even when a corrupted aggregator colludes with corrupted users. Second, it captures a new type of forgery **Type III Forgery** that represents corrupted users adding invalid inputs to the sum.

4.6 Ideal Functionality for Distributed Tagging Protocol

In this section, we present the ideal functionality $\mathcal{F}_{\texttt{DTAG}}^{t,\mathbb{G}}$ for the distributed tagging protocol $\Pi_{\texttt{DTAG}}^{t,\mathbb{G}} \equiv \texttt{VSA.Tag}$. The protocol runs between the user and the m taggers. The user provides an input x and each tagger provides a share of the tagging key k and a bound θ_j. The protocol outputs a tag computed by $\texttt{PUDA.Tag}(k, x, \tau)$ if x is less than all the bounds of the honest taggers. Otherwise, it gets no output. Figure 4.2 shows the functionality $\mathcal{F}_{\texttt{DTAG}}^{t,\mathbb{G}}$.

4.7 VSA - Construction in the $\hat{\mathcal{F}}_{\texttt{DTAG}}^{t,\mathbb{G}}$-Hybrid Model

In this section, we first present our VSA protocol. Then, we prove its correctness and that it achieves the security properties defined in Section 4.5. We describe the protocol in the hybrid model of the functionalities $\mathcal{F}_{\texttt{CRS}}^{\mathcal{D},n}$ (Section 2.4.2), $\mathcal{F}_{\texttt{DTAG}}^{t,\mathbb{G}}$ (Section 4.6), and $\mathcal{F}_{\texttt{RShare}}^{t,n,\mathbb{F}}$ (Section 2.4.4). Figure 4.3 illustrates a single run of the protocol.

4.7.1 VSA Scheme

Let Λ be a sampling algorithm that returns a uniformly random tuple of the form $(g_1, g_2, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, H^{\mathbb{G}_1})$ where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are groups where DDH holds of prime order p; g_1 is a generator of $\mathbb{G}_1$; g_2 is generator of $\mathbb{G}_2$; $e : \{\mathbb{G}_1, \mathbb{G}_2\} \to \mathbb{G}_T$ is a bilinear map; and $H^{\mathbb{G}_1}$ is a hash function as defined in Section 2.4.3.

- $\texttt{VSA.Setup}(1^\lambda, t)$:

Functionality $\mathcal{F}_{\texttt{DTAG}}^{t,\mathbb{G}}$

$\mathcal{F}_{\texttt{DTAG}}^{t,\mathbb{G}}$ runs between user U and m taggers $\mathcal{T} = \{T_1, ..., T_m\}$ and is parameterized by the reconstruction threshold $t \leq m$ of the SSS scheme. Let $\mathcal{T}_h \subset \mathcal{T}$ be the set of honest taggers such that $|\mathcal{T}_h| \geq t$.
Auxiliary Inputs: The bit size ℓ of the user's input x and the tagger's bound θ_j. A function $\texttt{PUDA.Tag} : \mathbb{Z}_p^2 \times [2^\ell]^- \times \{0,1\}^* \mapsto \mathbb{G}$ where $\mathbb{G}$ is a cyclic group of order p.

- Upon receiving a message $(\texttt{sid}, \texttt{input}, x)$ from U where $x \in [2^\ell]^-$, store x. Then, send message $(\texttt{sid}, \texttt{input}, U)$ to $\mathcal{S}$.
- Upon receiving a message $(\texttt{sid}, \texttt{input}, \langle \texttt{tk} \rangle_j, \theta_j)$ from T_j where $\langle \texttt{tk} \rangle_j = (\langle a \rangle_j, \langle b \rangle_j) \in \mathbb{Z}_p^2$ and $\theta_j \in [2^\ell]^-$, store $\langle \texttt{tk} \rangle_j$ and θ_j. Then, send message $(\texttt{sid}, \texttt{input}, T_j)$ to $\mathcal{S}$ and continue.
- When $(\texttt{sid}, \texttt{input})$ is received from U and all T_j, set $\texttt{tk} \leftarrow (a, b)$ such that $a \leftarrow \texttt{SSS.Recon}(\{(j, \langle a \rangle_j)\}_{\forall j | T_j \in \mathcal{T}_h}, \mathbb{Z}_p)$ and $b \leftarrow \texttt{SSS.Recon}(\{(j, \langle b \rangle_j)\}_{\forall j | T_j \in \mathcal{T}_h}, \mathbb{Z}_p)$. Finally, check if $x \leq \texttt{min}\{\theta_j\}_{\forall j | T_j \in \mathcal{T}_h}$. If yes, compute $\sigma \leftarrow \texttt{PUDA.Tag}(\texttt{tk}, x, \texttt{sid}^a)$. If no, set $\sigma \leftarrow \perp$. Send $(\texttt{sid}, \texttt{output}, \sigma)$ to U and halt.

[a] The session id (sid) is the current timestamp ($\texttt{sid} = \tau$)

Figure 4.2: Ideal functionality for a Distributed Tagging Protocol.

- **Distributing the public parameters**:
  The m taggers, the aggregator, and the n users query $\mathcal{F}_{\texttt{CRS}}^{\Lambda, m+n+1}$ and receives the public parameters $pp = (g_1, g_2, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, H^{\mathbb{G}_1})$.
- **Distributing the encryption and aggregation keys**:
  Each tagger T_j uniformly samples $\texttt{ek}_{i,j} \leftarrow\!\!\$\ \mathbb{Z}_p$ per user (i.e. $\forall U_i \in \mathcal{U}$), and computes $\texttt{ak}_j = -\sum_{\forall U_i \in \mathcal{U}} \texttt{ek}_{i,j}$. It sends to each user $U_i \in \mathcal{U}$ the key $\texttt{ek}_{i,j}$ and to the aggregator $\texttt{ak}_j$. Each user computes its encryption key $\texttt{ek}_i = \sum_{j \in [n]} \texttt{ek}_{i,j}$ and the aggregator computes the aggregation key $\texttt{ak} = \sum_{j \in [n]} \texttt{ak}_j$.
- **Distributing the tagging keys**:
  Each tagger T_j queries $\mathcal{F}_{\texttt{RShare}}^{t,m,\mathbb{Z}_p}$ $n+1$ times and receives the share $\langle a \rangle_j$ and the n shares $\langle b_i \rangle_j$ for each $i \in [n]$. It sets tagging keys $\texttt{tk}_{i,j} = (\langle a \rangle_j, \langle b_i \rangle_j)$ $\forall i \in [n]$. The tagger key is $\texttt{TK}_j = \{\texttt{tk}_{1,j}, ..., \texttt{tk}_{n,j}\}$.
- **Distributing the verification keys**:
  Each tagger T_j publishes $\texttt{vk}_{1,j} = g_2^{\langle a \rangle_j}$ and $\texttt{vk}_{2,j} = g_2^{\sum_{i \in [n]} \langle b_i \rangle_j}$. Each tagger then computes $\texttt{SSS.ExpRecon}([m], \{\texttt{vk}_{1,j}\}_{\forall j \in [m]}, \mathbb{G}_2) \rightarrow \texttt{vk}_1 = g_2^a$ and $\texttt{SSS.ExpRecon}([m], \{\texttt{vk}_{2,j}\}_{\forall j \in [m]}, \mathbb{G}_2) \rightarrow \texttt{vk}_2 = g_2^{\sum_{i \in [n]} b_i}$ and finally $\texttt{vk} = (\texttt{vk}_1, \texttt{vk}_2)$ is published.

- $\texttt{VSA.Enc}(\texttt{ek}_i, x_i, \tau) \equiv \texttt{PUDA.Enc}(\texttt{ek}_i, x_i, \tau)$

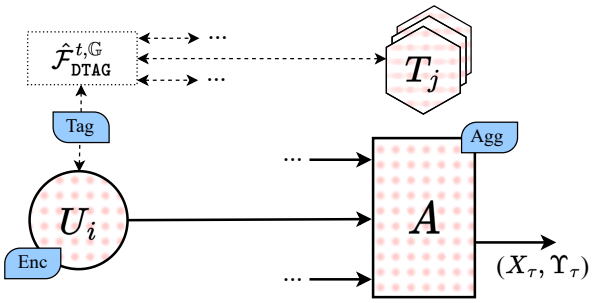


Figure 4.3: Our verifiable secure aggregation protocol. All parties can be malicious and may collude except a threshold number taggers.

- $\texttt{VSA.Tag}(\{\texttt{tk}_{i,j}\}_{j\in J_i}, \{\theta_j\}_{j\in J_i}, x_i, \tau)$:
  The user U_i (with input x_i) and each tagger T_j (with input $(\texttt{tk}_{i,j}, \theta_j)$) query $\hat{\mathcal{F}}^{t,\mathbb{G}_1}_{\texttt{DTAG}}$. The user receives the output σ_τ from $\hat{\mathcal{F}}^{t,\mathbb{G}_1}_{\texttt{DTAG}}$.

- $\texttt{VSA.Agg}(\texttt{ak}, \{c_i\}_{\forall i\in[n]}, \{\sigma_i\}_{\forall i\in[n]}, \tau) \equiv \texttt{PUDA.Agg}(\texttt{ak}, \{c_i\}_{\forall i\in[n]}, \{\sigma_i\}_{\forall i\in[n]}, \tau)$

- $\texttt{VSA.Verify}(\texttt{vk}, X_\tau, \Upsilon_\tau, \tau) \equiv \texttt{PUDA.Verify}(\texttt{vk}, X_\tau, \Upsilon_\tau, \tau)$

4.7.2 Proof of Correctness

The correctness of our construction directly follows from the correctness of PUDA protocol. To prove this argument, observe that $\forall\lambda$, $\forall t$, such that $\texttt{VSA.Setup}(1^\lambda, t)$ and $\texttt{PUDA.Setup}(1^\lambda)$, the following holds:

- $\forall x_i$, the ciphertexts produced by PUDA and VSA schemes are statistically indistinguishable since $\texttt{VSA.Enc} \equiv \texttt{PUDA.Enc}$.

- $\forall J_i, \forall x_i, \forall\theta_j$ s.t. $(J_i \subset [m]) \wedge (|J_i| \geq t) \wedge (x_i \leq \theta_j)$, the tags produced by PUDA and VSA schemes are statistically indistinguishable. This holds because VSA outputs the result of $\hat{\mathcal{F}}^{t,\mathbb{G}_1}_{\texttt{DTAG}}$ which produces tags computed with $\texttt{PUDA.Tag}$ algorithm if x_i is less than the minimum bound and a sufficient number of taggers participate.

- The aggregation and verification algorithms are equivalent: $\texttt{VSA.Agg} \equiv \texttt{PUDA.Agg}$ and $\texttt{VSA.Verify} \equiv \texttt{PUDA.Verify}$.

4.8 VSA - Security Analysis

We now perform a security analysis of the scheme. To prove that our VSA scheme achieves *Aggregator Obliviousness* and *Aggregate Unforgeability*, we first take a step to show that both our AO game and AU game can be reduced to $\text{AO}^{\texttt{PUDA}}$ game and $\text{AU}^{\texttt{PUDA}}$ game.

Lemma 4.8.1

For any $\epsilon > 0$ and poly-bounded adversary $\mathcal{A}$ that statically corrupts $\mathcal{U}_{\texttt{corr}} \subset \mathcal{U}$, A, or $\mathcal{T}_{\texttt{corr}} \subset \mathcal{T}$ s.t. $|\mathcal{T}| < t$ (t is the SSS threshold), or any combination of them where $Pr[\mathcal{A}^{\text{AO}}] \leq \frac{1}{2} + \epsilon$ there exists a poly-bounded adversary $\mathcal{B}$ that statically corrupt $\mathcal{U}_{\texttt{corr}} \subset \mathcal{U}$ and/or A such that $Pr[\mathcal{B}^{\text{AO}^{\text{PUDA}}}] \geq \frac{1}{2} + \epsilon$.

Proof.

Notice that this reduction is only true when the adversary does not corrupt t or more taggers ($|\mathcal{T}_{\texttt{corr}}| < t$). To prove this reduction, we construct the adversary $\mathcal{B}$ using $\mathcal{A}$. Let us denote the oracles that $\mathcal{B}$ has access to by $\mathcal{O}^{\texttt{PUDA}}_{\texttt{Setup}}$, $\mathcal{O}^{\texttt{PUDA}}_{\texttt{Corrupt}}$, $\mathcal{O}^{\texttt{PUDA}}_{\texttt{EncTag}}$, and $\mathcal{O}^{\texttt{PUDA}}_{\texttt{AO}}$ defined in [LEÖM15]. $\mathcal{B}$ proceeds as follows:

1. During the **Setup and Corruption Phase**: Given that the corrupted parties in $\mathcal{U}_{\texttt{corr}} \cup \mathcal{T}_{\texttt{corr}} \cup \{A\}$, when $\mathcal{A}$ chooses the security parameter λ and the threshold t, $\mathcal{B}$ queries $\mathcal{O}^{\texttt{PUDA}}_{\texttt{Setup}}$ which returns $\texttt{pp}$, $\texttt{ak}$, and $\texttt{vk}$. It then queries $\mathcal{O}^{\texttt{PUDA}}_{\texttt{Corrupt}}$ with each user identifier i of the corrupted users $U_i \in \mathcal{U}_{\texttt{corr}}$ which returns $\{(\texttt{ek}_i, \texttt{tk}_i)\}_{\forall U_i \in \mathcal{U}_{\texttt{corr}}}$. $\mathcal{B}$ samples uniformly at random $a' \leftarrow\!\!\$\ \mathbb{Z}_p$ and $b'_i \leftarrow\!\!\$\ \mathbb{Z}_p$ for each $i \in [n]$. Then it computes $\texttt{SSS.Share}(a', t, m, \mathbb{Z}_p)$ and $\texttt{SSS.Share}(b'_i, t, m, \mathbb{Z}_p)$ for each $i \in [n]$. It sets $\texttt{TK}_j = \{(\langle a' \rangle_j, \langle b'_1 \rangle_j), ..., (\langle a' \rangle_j, \langle b'_n \rangle_j)\}$ for each corrupted $T_j \in \mathcal{T}_{\texttt{corr}}$. $\mathcal{B}$ gives $\mathcal{A}$ the public parameters $\texttt{pp}$, the verification key of PUDA $\texttt{vk}$, the aggregation key $\texttt{ak}$, the corrupted users' encryption keys $\{\texttt{ek}_i\}_{\forall U_i \in \mathcal{U}_{\texttt{corr}}}$, and the randomly generated list of shares $\texttt{TK}_j$ of the corrupted taggers.

2. During the **Learning Phase**: When $\mathcal{A}$ queries $\mathcal{O}_{\texttt{EncTag}}$, $\mathcal{B}$ returns the values from $\texttt{History}$ if the user identifier and timestamp were previously queried. Otherwise,

   - If $U_i \notin \mathcal{U}_{\texttt{corr}}$: it queries $\mathcal{O}^{\texttt{PUDA}}_{\texttt{EncTag}}$ with (i, x, τ) and gets $(c_{u,\tau}, \sigma_{u,\tau})$, saves it to $\texttt{History}$ and sends σ_i to $\mathcal{A}$.
   - If $U_i \in \mathcal{U}_{\texttt{corr}}$ and $x \leq \theta_{\texttt{min}}$: $\mathcal{B}$ computes $\texttt{PUDA.Tag}(\texttt{tk}_i, x, \tau) \rightarrow \sigma_i$, saves σ_i to $\texttt{History}$ and sends it to $\mathcal{A}$.
   - If $U_i \in \mathcal{U}_{\texttt{corr}}$ and $x > \theta_{\texttt{min}}$ it sets $\sigma_i \leftarrow \perp$ saves σ_i to $\texttt{History}$ and sends it to $\mathcal{A}$.

3. During the **Challenge Phase**: When $\mathcal{A}$ queries $\mathcal{O}_{\texttt{AO}}$ with some parameters, $\mathcal{B}$ queries $\mathcal{O}^{\texttt{PUDA}}_{\texttt{AO}}$ with the same parameters then gives the result to $\mathcal{A}$ and outputs what it outputs.

In this reduction, the difference between $\mathcal{A}$'s simulated view (by $\mathcal{B}$) and $\mathcal{A}$'s real view (when it plays the real game) is only the tagging keys' shares of the corrupted taggers. Since $\mathcal{A}$ does not see a sufficient number of shares (because $|\mathcal{T}_{\texttt{corr}}| < t$), the shares from both views are indistinguishable (see Security definition of Shamir's secret sharing in section 2.3.1). Therefore, if $\mathcal{A}$ outputs $b^* = b$ (wins VSA AO game) with a probability

$\frac{1}{2} + \epsilon$, $\mathcal{B}$ will also output $b^* = b$ (win PUDA AO game) with the same probability. This proves the lemma.

Lemma 4.8.2

For any $\epsilon > 0$ and poly-bounded adversary $\mathcal{A}$ that statically corrupts $\mathcal{U}_{\texttt{corr}} \subset \mathcal{U}$, A, or $\mathcal{T}_{\texttt{corr}} \subset \mathcal{T}$ s.t. $|\mathcal{T}| < t$ (t is the SSS threshold), or any combination of them where $Pr[\mathcal{A}_{\text{I}}^{\text{AU}}] + Pr[\mathcal{A}_{\text{II}}^{\text{AU}}] \leq \epsilon$ there exists a poly-bounded adversary $\mathcal{B}$ that statically corrupt A such that $Pr[\mathcal{B}^{\text{AU}^{\texttt{PUDA}}}] \geq \epsilon$.

Proof.

Notice that this reduction is only true when the adversary does not corrupt t or more taggers ($|\mathcal{T}_{\texttt{corr}}| < t$). To prove this reduction, we construct the adversary $\mathcal{B}$ using $\mathcal{A}$. Denote the oracles that $\mathcal{B}$ has access to by $\mathcal{O}^{\texttt{PUDA}}_{\texttt{Setup}}$ and $\mathcal{O}^{\texttt{PUDA}}_{\texttt{EncTag}}$ defined in [LEÖM15]. Notice that $\mathcal{B}$ does not have access to oracle $\mathcal{O}^{\texttt{PUDA}}_{\texttt{Corrupt}}$ since in PUDA the aggregator is not allowed to collude with the users. However, $\mathcal{B}$ should provide $\mathcal{A}$ with the encryption keys of the corrupted users. It turns out that $\mathcal{B}$ can simply generate a random encryption key for each user and use that key to produce ciphertexts instead of using $\mathcal{O}^{\texttt{PUDA}}_{\texttt{EncTag}}$ ($\mathcal{B}$ needs $\mathcal{O}^{\texttt{PUDA}}_{\texttt{EncTag}}$ to only produce the tags). In more detail, $\mathcal{B}$ proceeds as follows:

1. During the **Setup and Corruption Phase**: Given the corrupted parties are $\mathcal{U}_{\texttt{corr}} \cup \mathcal{T}_{\texttt{corr}} \cup \{A\}$, $\mathcal{B}$ queries $\mathcal{O}^{\texttt{PUDA}}_{\texttt{Setup}}$ which returns $\texttt{pp}$, $\texttt{ak}$, and $\texttt{vk}$. It then samples the user encryption keys uniformly at random for all users $\texttt{rk}_i \leftarrow\!\!\$\ \mathbb{Z}_p$ such that $\sum_{i\in[n]} \texttt{rk}_i = -\texttt{ak}$. It also samples $a' \leftarrow\!\!\$\ \mathbb{Z}_p$ and $b'_i \leftarrow\!\!\$\ \mathbb{Z}_p$ uniformly at random for each $i \in [n]$. Then it computes $\texttt{SSS.Share}(a', t, m, \mathbb{Z}_p)$ and $\texttt{SSS.Share}(b'_i, t, m, \mathbb{Z}_p)$ for each $i \in [n]$. It sets $\texttt{TK}_j = \{(\langle a'\rangle_j, \langle b'_1\rangle_j), ..., (\langle a'\rangle_j, \langle b'_n\rangle_j)\}$ for each corrupted $T_j \in \mathcal{T}_{\texttt{corr}}$. $\mathcal{B}$ gives $\mathcal{A}$ the public parameters $\texttt{pp}$, the verification key of PUDA $\texttt{vk}$, the aggregation key $\texttt{ak}$, the randomly generated encryption keys of the corrupted users $\{\texttt{rk}_i\}_{\forall U_i \in \mathcal{U}_{\texttt{corr}}}$, and the randomly generated list of shares $\texttt{TK}_j$ of the corrupted taggers. $\mathcal{B}$ maintains a list $\texttt{History}$ of the queried ciphertexts and tags.

2. During the **Learning Phase**: When $\mathcal{A}$ queries $\mathcal{O}_{\texttt{EncTag}}$, $\mathcal{B}$ returns the values from $\texttt{History}$ if the user identifier and timestamp where previously queried. Otherwise,

   - If $x \leq \theta_{\texttt{min}}$: $\mathcal{B}$ queries $\mathcal{O}^{\texttt{PUDA}}_{\texttt{EncTag}}$ with (i, x, τ) and gets (c_i, σ_i). $\mathcal{B}$ computes $c'_i \leftarrow \texttt{VSA.Enc}(\texttt{rk}_i, x, \tau)$, saves (c'_i, σ_i) to $\texttt{History}$ and sends it to $\mathcal{A}$.
   - If $x > \theta_{\texttt{min}}$: $\mathcal{B}$ computes $c'_i \leftarrow \texttt{VSA.Enc}(\texttt{rk}_i, x, \tau)$, saves $(c'_i, \perp)$ to $\texttt{History}$ and sends it to $\mathcal{A}$.

3. During the **Challenge Phase**: $\mathcal{B}$ outputs what $\mathcal{A}$ outputs.

In this reduction, the differences between $\mathcal{A}$'s simulated view (by $\mathcal{B}$) and $\mathcal{A}$'s real view (when it plays the real game) are: (i) the encryption keys of the corrupted users, (ii) the tagging keys' shares of the corrupted taggers, and (iii) the ciphertexts received from the oracles. The tagging keys' shares are statistically indistinguishable in both views as shown in the previous proof. Similarly, the encryption keys are chosen uniformly randomly as if they were generated by the setup algorithm. It remains to show that the ciphertexts received from the oracles are indistinguishable in both views. In both views, the ciphertexts are generated by randomly generated encryption keys. Additionally, they both satisfy the relation $\prod_{i\in[n]} c_i = (g_1^{\sum_{i\in[n]} x_i})H(\tau)^{-\mathtt{ak}}$ thus, they are statistically indistinguishable.

We conclude that the simulated view of $\mathcal{A}$ (by $\mathcal{B}$) and the real view of $\mathcal{A}$ are statistically indistinguishable. Hence, if $\mathcal{A}$ succeeds in producing a **Type I or II Forgery** with probability ϵ then $\mathcal{B}$ succeeds with the same probability. This proves the lemma.

Corollary 4.8.1

For any poly-bounded adversary $\mathcal{A}$ that statically corrupts $\mathcal{U}_{\mathtt{corr}} \subset \mathcal{U}$, A, or $\mathcal{T}_{\mathtt{corr}} \subset \mathcal{T}$ s.t. $|\mathcal{T}| < t$ (t is the SSS threshold), or any combination of them, VSA achieves *Aggregator Obliviousness* under the decisional Diffie-Hellman (DDH) assumption in $\mathbb{G}_1$ in the random oracle model and $\mathcal{F}_{\mathtt{DTAG}}^{t,\mathbb{G}_1}$-hybrid model.

Proof.

Since PUDA ensures aggregator obliviousness under the DDH assumption in $\mathbb{G}_1$ in the random oracle model, and since our AO game reduces to $\mathrm{AO}^{\mathtt{PUDA}}$ game (see Lemma 4.8.1), we can conclude directly that our scheme also ensures aggregator obliviousness under the decisional Diffie-Hellman (DDH) assumption in $\mathbb{G}_1$ in the random oracle model and $\mathcal{F}_{\mathtt{DTAG}}^{t,\mathbb{G}_1}$-hybrid model.

Corollary 4.8.2

For any poly-bounded adversary $\mathcal{A}$ that statically corrupts $\mathcal{U}_{\mathtt{corr}} \subset \mathcal{U}$, A, or $\mathcal{T}_{\mathtt{corr}} \subset \mathcal{T}$ s.t. $|\mathcal{T}| < t$ (t is the SSS threshold), or any combination of them, VSA scheme achieves *Aggregate Unforgeability* against a **Type I Forgery** under the BCDH assumption in the random oracle model and $\mathcal{F}_{\mathtt{DTAG}}^{t,\mathbb{G}_1}$-hybrid model.

Proof.

Since PUDA ensures aggregate unforgeability against **Type I Forgery** under the BCDH assumption in the random oracle model, and since our AU game reduces to $\mathrm{AU}^{\mathtt{PUDA}}$ game (see Lemma 4.8.2), we can conclude directly that our scheme also ensures aggregate unforgeability against **Type I Forgery** under the BCDH assumption in the random oracle model and $\mathcal{F}_{\mathtt{DTAG}}^{t,\mathbb{G}_1}$-hybrid model.

Corollary 4.8.3

For any poly-bounded adversary $\mathcal{A}$ that statically corrupts $\mathcal{U}_{\texttt{corr}} \subset \mathcal{U}$, A, or $\mathcal{T}_{\texttt{corr}} \subset \mathcal{T}$ s.t. $|\mathcal{T}| < t$ (t is the SSS threshold), or any combination of them, VSA scheme achieves *Aggregate Unforgeability* against a **Type II Forgery** under the LEOM assumption in the random oracle model and $\mathcal{F}_{\texttt{DTAG}}^{t,\mathbb{G}_1}$-hybrid model.

Proof.

Since PUDA ensures aggregate unforgeability against **Type II Forgery** under the BCDH assumption in the random oracle model, and since our AU game reduces to $\text{AU}^{\texttt{PUDA}}$ game (see Lemma 4.8.2), we can conclude directly that our scheme also ensures aggregate unforgeability against **Type II Forgery** under the LEOM assumption in the random oracle model and $\mathcal{F}_{\texttt{DTAG}}^{t,\mathbb{G}_1}$-hybrid model.

Theorem 4.8.1

For any poly-bounded adversary $\mathcal{A}$ that statically corrupts $\mathcal{U}_{\texttt{corr}} \subset \mathcal{U}$, A, or $\mathcal{T}_{\texttt{corr}} \subset \mathcal{T}$ s.t. $|\mathcal{T}| < t$ (t is the SSS threshold), or any combination of them, VSA scheme achieves *Aggregate Unforgeability* against a **Type III Forgery** under the LEOM assumption in the random oracle model and $\mathcal{F}_{\texttt{DTAG}}^{t,\mathbb{G}_1}$-hybrid model.

Proof.

Let us assume that $\texttt{sum}_{\tau^*} > \sum_{\forall U_i \in \mathcal{U} \setminus \mathcal{U}_{\texttt{corr}}} x_{i,\tau^*} + |\mathcal{U}_{\texttt{corr}}|\theta_{\texttt{min}}$. This gives us that $\sum_{\forall U_i \in \mathcal{U}_{\texttt{corr}}} x_{i,\tau^*} > |\mathcal{U}_{\texttt{corr}}|\theta_{\texttt{min}}$. Notice that in this case the probability that $\texttt{VSA.Verify}(vk, \texttt{sum}_{\tau^*}, \Upsilon_{\tau^*})= 1$ is $Pr[\mathcal{A}_{\text{III}}^{\text{AU}}]$. Now observe that to have $\sum_{\forall U_i \in \mathcal{U}_{\texttt{corr}}} x_{i,\tau^*} > |\mathcal{U}_{\texttt{corr}}|\theta_{\texttt{min}}$ there exists at least one $x_{i,\tau^*} > \theta_{\texttt{min}}$. Let $\mathcal{U}'_{\texttt{corr}} \neq \phi$ be the set of users that their user identifiers were queried with $x_{i,\tau^*} > \theta_{\texttt{min}}$. Observe that $\texttt{VSA.Tag}$ returns $\perp$ when run with input $x > \theta_{\texttt{min}}$. Therefore, the oracle $\mathcal{O}_{\texttt{EncTag}}$ returns $\sigma_i = \perp$ (no output) for all queries (i, x, τ^*) where $U_i \in \mathcal{U}'_{\texttt{corr}}$. Let $\texttt{sum}'_{\tau^*} = \sum_{\forall U_i \in \mathcal{U} \setminus \mathcal{U}'_{\texttt{corr}}} x_{i,\tau^*} + \sum_{\forall U_i \in \mathcal{U}'_{\texttt{corr}}} x'_{i,\tau^*}$ such that $\texttt{sum}_{\tau^*} \neq \texttt{sum}'_{\tau^*}$ and $x'_{i,\tau^*} > \theta_{\texttt{min}}$ $\forall U_i \in \mathcal{U}'_{\texttt{corr}}$. The oracle $\mathcal{O}_{\texttt{EncTag}}$ returns the same result (i.e., $\perp$) for both queries $(i, x_{i,\tau^*}, \tau^*)$ and $(i, x'_{i,\tau^*}, \tau^*)$ $\forall U_i \in \mathcal{U}'_{\texttt{corr}}$. Thus, we can deduce that $Pr[\texttt{VSA.Verify}(vk, \texttt{sum}_{\tau^*}, \Upsilon_{\tau^*}) = 1] \simeq Pr[\texttt{VSA.Verify}(vk, \texttt{sum}'_{\tau^*}, \Upsilon_{\tau^*}) = 1]$. Therefore, $Pr[\mathcal{A}_{\text{III}}^{\text{AU}}] = Pr[\mathcal{A}_{\text{II}}^{\text{AU}}]$.

By relying on Corollary 4.8.3, we conclude that our scheme ensures aggregate unforgeability against **Type III Forgery** under the LEOM assumption in the random oracle model and $\mathcal{F}_{\texttt{DTAG}}^{t,\mathbb{G}_1}$-hybrid model.

4.9 Realization of Distributed Tagging Protocol

In this section, we show how to build the protocol $\Pi_{\texttt{DTAG}}^{t,\mathbb{G}} \equiv \texttt{VSA.Tag}$ that realizes the functionality $\mathcal{F}_{\texttt{DTAG}}^{t,\mathbb{G}}$. To build this protocol, we first propose a tagging protocol $\Pi_{\texttt{TAG}}^{\mathbb{G}}$

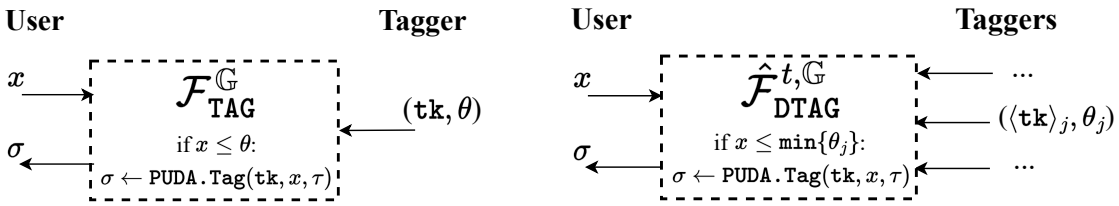


Figure 4.4: Overview of the tagging and distributed tagging functionalities

with ideal functionality $\mathcal{F}_{\mathtt{TAG}}^{\mathbb{G}}$ that runs between one user and one tagger only. The ideal functionality $\mathcal{F}_{\mathtt{TAG}}^{\mathbb{G}}$ is shown in Figure 4.5. Notice that $\mathcal{F}_{\mathtt{TAG}}^{\mathbb{G}}$ is very similar to $\mathcal{F}_{\mathtt{DTAG}}^{t,\mathbb{G}}$ (with one tagger) except that the adversary in $\mathcal{F}_{\mathtt{TAG}}^{\mathbb{G}}$ is allowed to perform a selective failure attack (i.e., choose to abort the protocol on certain inputs from the honest user). Given the functionality, $\mathcal{F}_{\mathtt{TAG}}^{\mathbb{G}}$, the protocol $\Pi_{\mathtt{DTAG}}^{t,\mathbb{G}}$ is defined in the $\mathcal{F}_{\mathtt{TAG}}^{\mathbb{G}}$-hybrid model as shown in Figure 4.6.

Theorem 4.9.1

Protocol $\Pi_{\mathtt{DTAG}}^{t,\mathbb{G}}$ UC-realizes the functionality $\mathcal{F}_{\mathtt{DTAG}}^{t,\mathbb{G}}$ in the $\mathcal{F}_{\mathtt{TAG}}^{\mathbb{G}}$-hybrid model under static corruption if the number of corrupted taggers $|\mathcal{T}_{\mathtt{corr}}| < t$ and $|\mathcal{T}_{\mathtt{corr}}| < m - t$ (eg., for $t = \frac{m}{2}$, it is sufficient to choose $|\mathcal{T}_{\mathtt{corr}}| < \frac{m}{2}$).

Proof.

To prove the theorem we build a simulator $\mathcal{S}$ that runs in the ideal world and interacts with $\mathcal{F}_{\mathtt{DTAG}}^{t,\mathbb{G}}$ and $\mathcal{Z}$. It simulates for the adversary $\mathcal{A}$ the interaction with the non-corrupted parties and $\mathcal{Z}$. We consider three different cases depending on which parties $\mathcal{A}$ corrupts. In each case, $\mathcal{S}$ runs internally an instance of $\mathcal{A}$. It simulates the interaction with $\mathcal{Z}$ by forwarding the messages sent from $\mathcal{Z}$ to $\mathcal{A}$ and vice-versa. It simulates the interaction with the protocol parties based on a well-defined strategy described in what follows.

Case 1: $\mathcal{A}$ corrupts U
Observe that U only interacts with $\hat{\mathcal{F}}_{\mathtt{TAG}}^{\mathbb{G}}$. Thus, $\mathcal{S}$ simply forwards the messages between $\mathcal{A}$ and $\hat{\mathcal{F}}_{\mathtt{TAG}}^{\mathbb{G}}$ and vice-versa. Hence, the simulated execution of $\mathcal{A}$ and the real one are identical. Additionally, the honest parties do not get any output. Therefore, we proved that $\Pi_{\mathtt{DTAG}}^{t,\mathbb{G}}$ UC-realizes the functionality $\mathcal{F}_{\mathtt{DTAG}}^{t,\mathbb{G}}$ in the $\mathcal{F}_{\mathtt{TAG}}^{\mathbb{G}}$-hybrid model if $\mathcal{A}$ corrupts only U.

Case 2: $\mathcal{A}$ corrupts $\mathcal{T}_{\mathtt{corr}} \subset \mathcal{T}$
Observe that each $T_j \in \mathcal{T}_{\mathtt{corr}}$ sends a message to $\hat{\mathcal{F}}_{\mathtt{TAG}}^{\mathbb{G}}$ only and receives nothing. So $\mathcal{S}$ simply forwards the message from $\mathcal{A}$ to $\hat{\mathcal{F}}_{\mathtt{TAG}}^{\mathbb{G}}$. Hence, the simulated execution of $\mathcal{A}$ and the real one are identical. We only need to show that the

Functionality $\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}}$

$\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}}$ runs between user U and a tagger T and is parameterized by the function f and the reconstruction threshold of the SSS scheme $t \leq m$.
Auxiliary Inputs: The bit size ℓ of the user's input x and the tagger's bound θ_j. A function $\texttt{PUDA.Tag} : \mathbb{Z}_p^2 \times [2^\ell]^- \times \{0,1\}^* \mapsto \mathbb{G}$ where $\mathbb{G}$ is a cyclic group of order p.

- Upon receiving a message $(\texttt{sid}, \texttt{input}, x)$ from U where $x \in [2^\ell]^-$, store x. Then, if the T is corrupted, send message $(\texttt{sid}, \texttt{input}, U)$ to $\mathcal{S}$ and receive the message $(\texttt{sid}, \texttt{ok}, \eta)$ where $\eta : [2^\ell]^- \mapsto \{0,1\}$. If $\eta(x) = 0$ abort, otherwise, continue.
- Upon receiving a message $(\texttt{sid}, \texttt{input}, k, \theta)$ from T where $k \in \mathbb{Z}_p^2$ and $\theta \in [2^\ell]^-$, store k and θ. Then, if U is corrupted send message $(\texttt{sid}, \texttt{input}, T)$ to $\mathcal{S}$.
  - If $\mathcal{S}$ sends $(\texttt{sid}, \texttt{abort})$, send abort to all parties and abort.
  - If $\mathcal{S}$ sends $(\texttt{sid}, \texttt{ok})$, continue.
- When $(\texttt{sid}, \texttt{input})$ is received from U and all T_i, compute $\sigma \leftarrow \texttt{PUDA.Tag}(k, x, \texttt{sid}^a)$. If no, set $\sigma \leftarrow \perp$. Send $(\texttt{sid}, \texttt{output}, \sigma)$ to U and halt.

[a] The session id (sid) is the same as the timestamp ($\texttt{sid}=\tau$)

Figure 4.5: Ideal functionality for a Tagging Protocol.

Protocol $\Pi_{\texttt{DTAG}}^{t,\mathbb{G}}$

$\Pi_{\texttt{DTAG}}^{t,\mathbb{G}}$ runs between user U and m taggers $\mathcal{T} = \{T_1, ..., T_m\}$ and is parameterized by the reconstruction threshold of the SSS scheme $t \leq m$.
Inputs: User's input $x \in [2^\ell]^-$ and tagger T_j input $(\langle \texttt{tk} \rangle_j = (\langle a \rangle_j, \langle b \rangle_j), \theta_j) \in \mathbb{Z}_p^2 \times [2^\ell]^-$.

- User U initializes an empty set Ω then interacts with each tagger T_j as follows:
  - U sends $(\texttt{sid}, \texttt{ssid}, \texttt{input}, x)$ to $\hat{\mathcal{F}}_{\texttt{TAG}}^{\mathbb{G}}$.
  - T_j sends $(\texttt{sid}, \texttt{ssid}, \texttt{input}, \langle \texttt{tk} \rangle_j, \theta_j)$ to $\hat{\mathcal{F}}_{\texttt{TAG}}^{\mathbb{G}}$.
  - U receives $(\texttt{sid}, \texttt{ssid}, \texttt{output}, \sigma_j)$ from $\hat{\mathcal{F}}_{\texttt{TAG}}^{\mathbb{G}}$ (or the sub-session ssid aborts). If $\sigma_j \neq \perp$, add (j, σ_j) to the set Ω.
- $\forall J_k \subset \Omega$ s.t. $|J_k| = t$, U computes $\texttt{SSS.ReconExp}(\{(j, \sigma_j)\}_{\forall (j,\sigma_j) \in J_k}, \mathbb{G}) \rightarrow \sigma'_k$. The user outputs $\sigma \leftarrow \texttt{majority}(\{\sigma'_k\}_{\forall k \in [\binom{t}{n}]})$.

Figure 4.6: Distributed Tagging Protocol in $\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}}$-hybrid model.

output of the honest parties (the user since it is the only party that gets an output) is identical in both worlds. In the real world, the user outputs $\sigma_{\texttt{REAL}} \leftarrow \texttt{majority}(\{\sigma'_k\}_{\forall k \in [\binom{t}{n}]})$ where $\sigma'_k \leftarrow \texttt{SSS.ReconExp}(\{(j, \sigma_j)\}_{\forall (j,\sigma_j) \in J_k}, \mathbb{G})$. In the ideal world, the user outputs $\sigma_{\texttt{IDEAL}} \leftarrow \texttt{PUDA.Tag}(\texttt{tk}, x, \texttt{sid})$ where $\texttt{tk} \leftarrow (a, b)$ such that $a \leftarrow \texttt{SSS.Recon}(\{(j, \langle a \rangle_j)\}_{\forall j | T_j \in \mathcal{T}_h}, \mathbb{Z}_p)$ and $b \leftarrow \texttt{SSS.Recon}(\{(j, \langle b \rangle_j)\}_{\forall j | T_j \in \mathcal{T}_h}, \mathbb{Z}_p)$. Thus, we need to show that $\sigma_{\texttt{IDEAL}} = \sigma_{\texttt{REAL}}$.

Notice that $\mathcal{T}_h = \mathcal{T} \setminus \mathcal{T}_{\texttt{corr}}$. Thus, for $|\mathcal{T}_{\texttt{corr}}| < m - t$, we have $|\mathcal{T}_h| > t$ and thus $\sigma_{\texttt{IDEAL}} = (g_1^x)^a H(\tau)^b$ where $g_1, H(\tau) \in \mathbb{G}$.

In the real world, $\sigma_j = (g_1^x)^{\langle a \rangle_j} H(\tau)^{\langle b \rangle_j} \ \forall T_j \in \mathcal{T}_h$. So, we have for each $\sigma'_k \leftarrow \texttt{SSS.ReconExp}(\{(j, \sigma_j)\}_{\forall (j,\sigma_j) \in J_k}, \mathbb{G})$, where $J_k \subset \mathcal{T}_h$, $\sigma'_k = (g_1^x)^a H(\tau)^b = \sigma_{\texttt{IDEAL}}$ (from Lemma 2.3.2). Since $|\mathcal{T}_h| > t$, then $\binom{t}{|\mathcal{T}_h|} > 1$. Hence, the number of σ'_k's that are equal to $\sigma_{\texttt{IDEAL}}$ are more than one. On the other hand, since $|\mathcal{T}_{\texttt{corr}}| < t$, for each J_k, we have $Pr[\sigma'_{k_1} = \sigma'_{k_2} \neq \sigma_{\texttt{IDEAL}}] \simeq 0$ (from Definition 2.3.2). Therefore, $Pr[\texttt{majority}(\{\sigma'_k\}_{\forall k \in [\binom{t}{n}]}) \neq \sigma_{\texttt{IDEAL}}] \simeq 0$. This proves that $\sigma_{\texttt{IDEAL}} = \sigma_{\texttt{REAL}}$ and consequently we proved $\Pi^{t,\mathbb{G}}_{\texttt{DTAG}}$ UC-realizes the functionality $\mathcal{F}^{t,\mathbb{G}}_{\texttt{DTAG}}$ in the $\mathcal{F}^{\mathbb{G}}_{\texttt{TAG}}$-hybrid model if $\mathcal{A}$ corrupts $\mathcal{T}_{\texttt{corr}} \subset \mathcal{T}$ and $|\mathcal{T}_{\texttt{corr}}| < t$ and $|\mathcal{T}_{\texttt{corr}}| < m - t$.

Case 3: $\mathcal{A}$ corrupts U and $\mathcal{T}_{\texttt{corr}} \subset \mathcal{T}$
Similar to the previous two cases, $\mathcal{S}$ will forward messages between $\mathcal{A}$ and $\hat{\mathcal{F}}^{\mathbb{G}}_{\texttt{TAG}}$. Hence, the simulated execution of $\mathcal{A}$ and the real one are identical. Additionally, the honest parties do not get any output. Therefore, we proved that $\Pi^{t,\mathbb{G}}_{\texttt{DTAG}}$ UC-realizes the functionality $\mathcal{F}^{t,\mathbb{G}}_{\texttt{DTAG}}$ in the $\mathcal{F}^{\mathbb{G}}_{\texttt{TAG}}$-hybrid model if $\mathcal{A}$ corrupts U and $\mathcal{T}_{\texttt{corr}} \subset \mathcal{T}$.

We finally deduce that, if $|\mathcal{T}_{\texttt{corr}}| < t$ and $|\mathcal{T}_{\texttt{corr}}| < m - t$, $\texttt{IDEAL}_{\mathcal{F}^{t,\mathbb{G}}_{\texttt{DTAG}},\mathcal{S},\mathcal{Z}} \overset{s}{\approx} \texttt{EXEC}_{\Pi^{t,\mathbb{G}}_{\texttt{DTAG}},\mathcal{A},\mathcal{Z}}$ which proves our theorem.

4.9.1 Realization of The Tagging Protocol

The ideal functionality of the protocol $\Pi^{\mathbb{G}}_{\texttt{TAG}}$ (depicted in Figure 4.5) computes the tag $\texttt{PUDA.Tag}(k = (a, b), x, \tau) = (g^a)^x H(\tau)^b$ if the user input x is less than a bound θ chosen by the tagger.

For illustration purposes, we first present a version of the protocol that is secure in the honest-but-curious (HBC) model (the user and the tagger follow the protocol steps). Next, we present our $\Pi^{\mathbb{G}}_{\texttt{TAG}}$ protocol and prove that it UC-realizes $\mathcal{F}^{\mathbb{G}}_{\texttt{TAG}}$ in the malicious model.

Tagging Protocol in the HBC model

The user generates a uniformly random value $r \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes $P{=}(g^x)^r$ and $Q{=}H(\tau)^r$ then send P and Q to the tagger. The tagger computes $E{=}P^a Q^b$. Notice that the user now can obtain the tag from E by computing $\sigma = E^{\frac{1}{r}}$. To prevent the user from obtaining the tag σ if its input $x > \theta$, the tagger does not send E to the user. Instead, it

runs a garbled circuit protocol as described in Section 2.4.6 with the function $f^{\theta,E}$:

$$f^{\theta,E}(x) = \begin{cases} E & \text{, if } x \leq \theta \\ \perp & \text{, otherwise} \end{cases}$$

This protocol is only secure in the HBC model since a malicious user may compute the tag on a value $x > \theta$, then evaluate the garbled circuit on a different value $x' \leq \theta$. On the other hand, a malicious tagger may garble any arbitrary function f' and thus influence the protocol to output $f'(x)$ instead of σ.

Tagging Protocol $\Pi^{\mathbb{G}}_{\texttt{TAG}}$ in the malicious model

To correctly realize the protocol $\Pi^{\mathbb{G}}_{\texttt{TAG}}$ in the malicious model, we need the tagger to ensure that the tag is evaluated on the same input used to evaluate the garbled circuit. Additionally, we need the user to ensure that the garbled circuit is evaluating a tag of data point x.

To solve the first issue, we propose to modify the OT protocol used in the garbled circuit protocol. The goal is to use the messages sent by the user in the OT protocol to evaluate the tag σ directly. By reusing these messages, we make sure that the tag is computed on the same input used to evaluate the circuit.

To solve the second issue, we use a zero-knowledge proof of knowledge of language $\mathcal{R}_{\texttt{Tag}}$ (see Section 2.4.7) to verify the result of the protocol. The details of the protocol are described in Figure 4.7.

Theorem 4.9.2

Protocol $\Pi^{\mathbb{G}}_{\texttt{TAG}}$ UC-realizes the functionality $\mathcal{F}^{\mathbb{G}}_{\texttt{TAG}}$ in the $\mathcal{F}^{\mathcal{R}}_{\texttt{ZK}}$-hybrid model under static corruption and under the Inv-DDH assumption in $\mathbb{G}$.

Proof.

To prove the theorem we build a simulator $\mathcal{S}$ that runs in the ideal world and interacts with $\mathcal{F}^{\mathbb{G}}_{\texttt{TAG}}$ and $\mathcal{Z}$. It simulates for the adversary $\mathcal{A}$ the interaction with the non-corrupted parties and $\mathcal{Z}$. We consider two different cases depending on what parties $\mathcal{A}$ corrupts. In each case, $\mathcal{S}$ runs internally an instance of $\mathcal{A}$. It simulates the interaction with $\mathcal{Z}$ by forwarding the messages sent from $\mathcal{Z}$ to $\mathcal{A}$ and vice-versa. It simulates the interaction with the protocol parties based on a well-defined strategy described in what follows.

Case 1: $\mathcal{A}$ corrupts the user U

1. $\mathcal{S}$ first chooses $\theta = 2^{\ell} - 1$ (largest bound possible). It garbles the circuit similar to an honest tagger $F, \{l^b_{x,i}\}, l^b_{\texttt{out}} \leftarrow \texttt{GC.Grb}(1^{\lambda}, f^{\theta})\ \forall i \in [\ell], \forall b \in \{0,1\}$ and samples value $\alpha, \gamma \leftarrow\$\ \mathbb{Z}_p$ uniformly at random. It sends the message $(\texttt{sid}, \gamma, A = g^{\alpha}, F)$ to $\mathcal{A}$ and waits for $\mathcal{A}'s$ response.

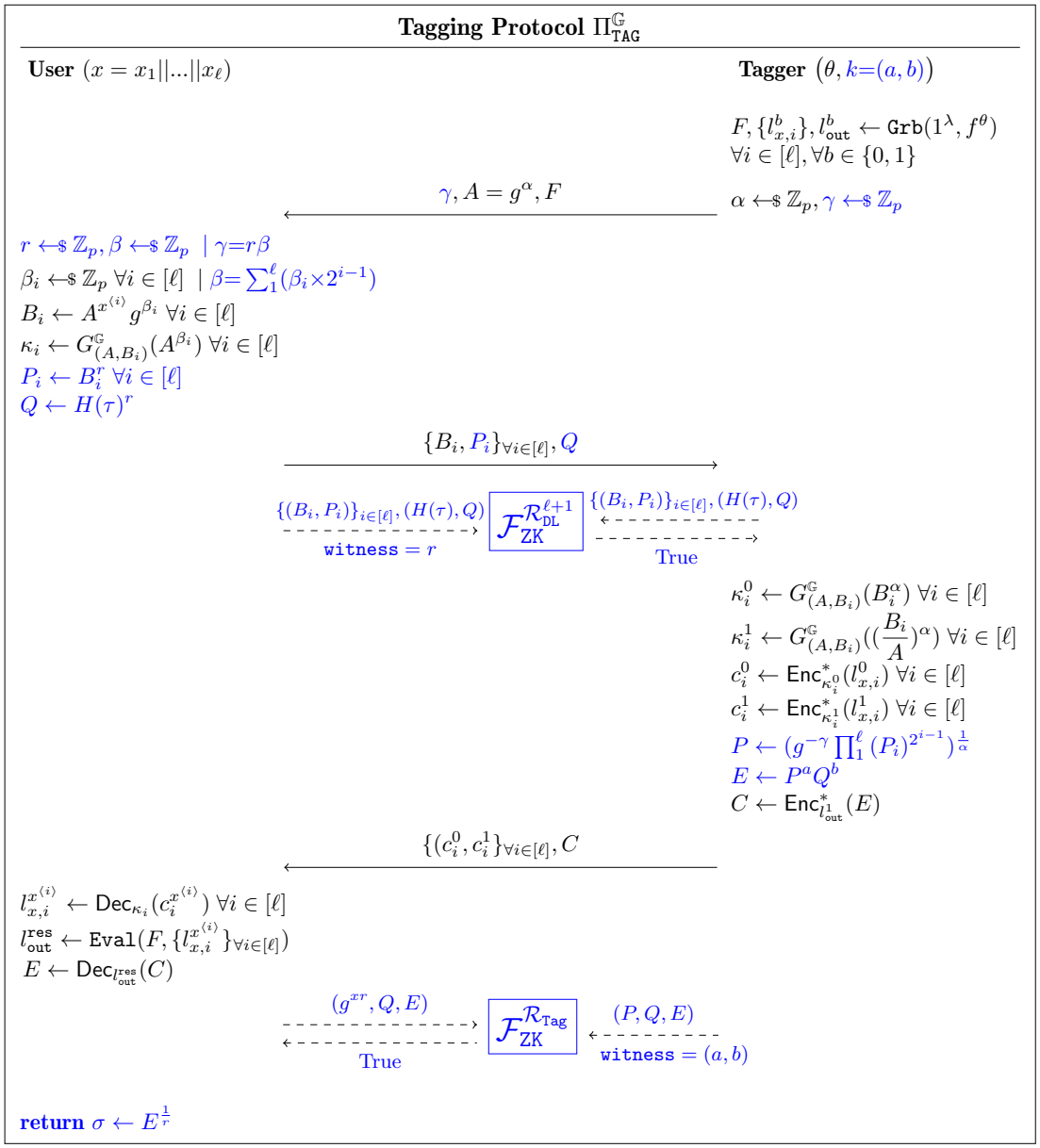


Figure 4.7: Tagging Protocol in $\mathcal{F}^{\mathcal{R}}_{\mathtt{ZK}}$-hybrid model. Black text is the GC part and Blue text is the modifications. Recall that timestamp $\tau{=}\mathtt{sid}$ represents the session id.

2. When $\mathcal{A}$ sends the message $(\texttt{sid}, \{B_i, P_i\}_{\forall i \in [\ell]}, Q)$, $\mathcal{S}$ starts internally $\mathcal{S}_{\texttt{rcv}}$ (defined in Lemma 2.4.1). It handles the OT receiver's message (B_i) and the random oracle $G^{\mathbb{G}}$ queries to $\mathcal{S}_{\texttt{rcv}}$. $\mathcal{S}_{\texttt{rcv}}$ extracts the receiver's choice $x^{\langle i \rangle}$ and sends it out to $\mathcal{S}$. $\mathcal{S}$ repeats the process for every $i \in [\ell]$. Hence, $\mathcal{S}$ extracts the user input x.

3. When $\mathcal{A}$ sends the message $(\texttt{sid}, \texttt{prove}, (\{(B'_i, P'_i)\}_{i \in [\ell]}, (H', Q')), r)$ to $\mathcal{F}_{\texttt{ZK}}^{\mathcal{R}_{\texttt{DL}}^{\ell+1}}$. $\mathcal{S}$ checks if $H'{=}H(\texttt{sid})$, $Q'{=}Q$, $B_i{=}B'_i$, and $P_i{=}P'_i$ $\forall i \in [\ell]$, and the relation $\mathcal{R}_{\texttt{DL}}^{\ell+1}$ hold, and send $(\texttt{sid}, \texttt{abort})$ to $\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}}$ if any of the checks fail. If the values are valid and the relation $\mathcal{R}_{\texttt{DL}}^{\ell+1}$ holds, then $\mathcal{S}$ sets $\beta = \frac{\gamma}{r}$ and checks if $\prod_1^{\ell} (B_i)^{2^{i-1}} = (g^{\alpha})^x g^{\beta}$. If this equality does not hold, this means $\mathcal{A}$ chose β_i such that $(\sum_1^n \beta_i \times 2^{i-1}) \neq \beta$. In this case, $\mathcal{S}$ sets the value $\texttt{bad\_input} = 1$ and continues.

4. $\mathcal{S}$ sends the message $(\texttt{sid}, \texttt{input}, x)$ to $\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}}$ and receives $(\texttt{sid}, \texttt{output}, \sigma)$. $\mathcal{S}$ computes:

   - $\kappa_i^0 \leftarrow G^{\mathbb{G}}_{(A,B_i)}(B_i^{\alpha})$ $\forall i \in [\ell]$,
   - $\kappa_i^1 \leftarrow G^{\mathbb{G}}_{(A,B_i)}((\frac{B_i}{A})^{\alpha})$ $\forall i \in [\ell]$
   - $c_i^0 \leftarrow \mathsf{Enc}^*_{\kappa_i^0}(l^0_{x,i})$ $\forall i \in [\ell]$
   - $c_i^1 \leftarrow \mathsf{Enc}^*_{\kappa_i^1}(l^1_{x,i})$ $\forall i \in [\ell]$

   Then, $\mathcal{S}$ computes:

   - if $\sigma = \perp$, samples $\varepsilon \leftarrow\!\!\$\ \{0,1\}^{\lambda}$ and $E \leftarrow\!\!\$\ \mathbb{G}$ then computes $C \leftarrow \mathsf{Enc}^*_{\varepsilon}(E)$.
   - if $\sigma \neq \perp$ and $\texttt{bad\_input} = 0$, sets $E = \sigma^r$ then computes $C \leftarrow \mathsf{Enc}^*_{l^1_{\text{out}}}(E)$.
   - if $\sigma \neq \perp$ and $\texttt{bad\_input} = 1$ sets $E \leftarrow\!\!\$\ \mathbb{G}$ then computes $C \leftarrow \mathsf{Enc}^*_{l^1_{\text{out}}}(E)$.

   $\mathcal{S}$ finally sends $(\texttt{sid}, \{(c_i^0, c_i^1\}_{\forall i \in [\ell]}, C)$ to $\mathcal{A}$.

5. (Proceed to this step only if $\sigma \neq \perp$). When $\mathcal{A}$ sends $(\texttt{sid}, (P', Q', E'))$ to $\mathcal{F}_{\texttt{ZK}}^{\mathcal{R}_{\texttt{Tag}}}$, if $P'{=}g^{rx}$, $Q'{=}Q$, and $E'{=}E$, and the relation $\mathcal{R}_{\texttt{Tag}}$ holds, $\mathcal{S}$ sends $(\texttt{sid}, \text{True})$ to $\mathcal{A}$, otherwise send $(\texttt{sid}, \text{False})$ to $\mathcal{A}$ and $(\texttt{sid}, \texttt{abort})$ to $\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}}$.

In this simulation, the difference between the simulated execution of $\mathcal{A}$ and the real one is in the generated garbled circuit F and in the ciphertext C. We argue that both values are indistinguishable between the real world and the ideal world.

Indistinguishability of the simulated garbled circuit F: The simulated garbled circuit is garbled from the function $f^{\theta_{\texttt{IDEAL}}}$ where $\theta_{\texttt{IDEAL}} = 2^{\ell} - 1$. The only difference is in the value of θ. However, the security of the garbled circuit scheme (see [Yao86]) guarantees computational indistinguishablity between these two circuits.

Indistinguishability of the simulated ciphertext C: There are three different cases in our simulation:

Case (i) when $\sigma = \perp$: This means that $\mathcal{A}$'s input x is greater than $\theta_{\texttt{REAL}}$ (input of the honest tagger). In this case, the adversary in the real world will not be able to decrypt the value of C since it will not obtain $l^1_{\texttt{out}}$ from the evaluation of the garbled circuit ($\mathcal{A}$ will obtain $l^0_{\texttt{out}}$ since $x \not\leq \theta_{\texttt{REAL}}$). Hence the ciphertext C in the ideal world which is encrypted by a random key ε is computationally indistinguishable from C in the real world based on the security property of the encryption scheme (see Definition 2.3.5 and Definition 2.3.7).

Case (ii) when $\sigma \neq \perp$ and $\texttt{bad\_input} = 0$: In the ideal world, the value of C is the encryption of σ^r with $l^1_{\texttt{out}}$ where σ is the output of $\mathcal{F}^{\mathbb{G}}_{\texttt{TAG}}$ and r is the randomness chosen by $\mathcal{A}$ (extracted from the $\mathcal{F}^{\mathcal{R}^{\ell+1}_{\text{DL}}}_{\texttt{ZK}}$ message). In the real world, the value of C is the encryption of E with $l^1_{\texttt{out}}$ where $E = P^a Q^b$. We have $P = (g^{-\gamma} \prod_1^\ell (P_i)^{2^{i-1}})^{\frac{1}{\alpha}} = g^{\frac{r\alpha x + r\beta - \gamma}{\alpha}} = g^{rx}$ (since $\gamma = r\beta$ when $\texttt{bad\_input} = 0$). Additionally, $Q = H(\tau)^r$, so we have $E = (g^{rx})^a H(\tau)^{rb} = \sigma^r$. Hence, the values of C are identical from both worlds.

Case (iii): $\sigma \neq \perp$ and $\texttt{bad\_input} = 1$: In the ideal world, the value of C is the encryption of $E_{\texttt{IDEAL}}$ with $l^1_{\texttt{out}}$ where $E_{\texttt{IDEAL}}$ is chosen uniformly random in $\mathbb{G}$. In the real world, the value of C is the encryption of $E_{\texttt{REAL}}$ with $l^1_{\texttt{out}}$ where $E_{\texttt{REAL}} = P^a Q^b$. We have $P = (g^{-\gamma} \prod_1^\ell (P_i)^{2^{i-1}})^{\frac{1}{\alpha}} = g^{\frac{r\alpha x + r\beta - \gamma}{\alpha}} \neq g^{rx}$ (since $\gamma \neq r\beta$ when $\texttt{bad\_input} = 1$). So we have $E_{\texttt{REAL}} = (g^{\frac{r\alpha x + r\beta - \gamma}{\alpha}})^a H(\tau)^{rb}$. Notice that $\mathcal{Z}$ does not know α which is chosen uniformly random in $\mathbb{Z}_p$. We can write the value of $E_{\texttt{REAL}} = g^{c_1 + c_2 \alpha^{-1}} H(\tau)^{c_3}$ where $c_1 = arx$, $c_2 = b(r\beta - \gamma) \neq 0$ (since $r\beta \neq \gamma$), and $c_3 = rb$ are three values chosen by $\mathcal{Z}$. Let us assume that $\mathcal{Z}$ can distinguish $E_{\texttt{REAL}}$ from $E_{\texttt{IDEAL}}$ with a non-negligible probability given only $g, g^\alpha \in \mathbb{G}$ and $\tau \in \{0,1\}^*$. We show that we can solve the Inv-DDH problem (i.e., given the tuple (g, g^X, g^Z) tell of $g^Z = g^{X^{-1}}$) using $\mathcal{Z}$. For a tuple (g, g^X, g^Z) query $\mathcal{Z}$ with $(g, g^\alpha{=}g^X, E_{\texttt{REAL}} = g^{c_1}(g^Z)^{c_2} H(\tau)^{c_3})$. Hence, the values of C are computationally indistinguishable from both worlds under the Inv-DDH assumption.

This proves that $\Pi^{\mathbb{G}}_{\texttt{TAG}}$ UC-realizes the functionality $\mathcal{F}^{\mathbb{G}}_{\texttt{TAG}}$ in the $\mathcal{F}^{\mathcal{R}}_{\texttt{ZK}}$-hybrid model under the Inv-DDH assumption in $\mathbb{G}$ if $\mathcal{A}$ statically corrupts U.

Case 2: $\mathcal{A}$ corrupts the tagger T

1. When $\mathcal{A}$ sends the message $(\texttt{sid}, \gamma, A = g^\alpha, F)$, $\mathcal{S}$ chooses $x{=}1$ (smallest input possible) and proceeds similarly to an honest user to compute $\{B_i\}_{\forall i \in [\ell]}, \{P_i\}_{\forall i \in [\ell]}$, and Q. $\mathcal{S}$ sends $(\texttt{sid}, \{B_i, P_i\}_{\forall i \in [\ell]}, Q)$ to $\mathcal{A}$.

2. $\mathcal{S}$ sends the message $(\texttt{sid},\texttt{prove},(\{(B_i,P_i)\}_{i\in[\ell]},(H(\texttt{sid}),Q)),r)$ to $\mathcal{F}_{\texttt{ZK}}^{\mathcal{R}_{\texttt{DL}}^{\ell+1}}$ and waits for $\mathcal{A}$ to sends $(\texttt{sid},(\{(B'_i,P'_i)\}_{i\in[\ell]},(H',Q')))$ to $\mathcal{F}_{\texttt{ZK}}^{\mathcal{R}_{\texttt{DL}}^{\ell+1}}$. If $H'{=}H(\texttt{sid})$, $Q'{=}Q$, $B_i{=}B'_i$, and $P_i{=}P'_i$ $\forall i \in [\ell]$, and the relation $\mathcal{R}_{\texttt{DL}}^{\ell+1}$ holds, $\mathcal{S}$ sends $(\texttt{sid},\text{True})$ to $\mathcal{A}$, otherwise send $(\texttt{sid},\text{False})$ to $\mathcal{A}$ and $(\texttt{sid},\texttt{abort})$ to $\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}}$.

3. When $\mathcal{A}$ sends the message $(\texttt{sid},\{(c_i^0,c_i^1\}_{\forall i\in[\ell]},C)$, $\mathcal{S}$ starts internally $\mathcal{S}_{\texttt{snd}}$ (defined in Lemma 2.4.2) and handles the OT sender's message $A=g^{\alpha}$,(c_i^0,c_i^1), and the random oracle $G^{\mathbb{G}}$ queries to $\mathcal{S}_{\texttt{snd}}$. $\mathcal{S}_{\texttt{snd}}$ extracts the sender's input $(l_{x,i}^0,l_{x,i}^1)$ and sends it out to $\mathcal{S}$. $\mathcal{S}$ repeats the process for every $i\in[\ell]$. Hence, $\mathcal{S}$ extracts all the input labels of the garbled circuit $\{l_{x,i}^b\}_{\forall i\in[\ell],b\in\{0,1\}}$. Then, $\mathcal{S}$ evaluates the garbled circuit F on all possible values of $x\in[2^{\ell}]^{-}$ using the labels and uses the output label to decrypt C and obtain E. If this operation fails for any x, $\mathcal{S}$ sets $\eta(x){=}0$, otherwise $\eta(x){=}1$. $\mathcal{S}$ finally sends $(\texttt{sid},\texttt{ok},\eta)$ to $\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}}$.

4. When $\mathcal{A}$ sends $(\texttt{sid},((P',Q',E'),(a,b))$ to $\mathcal{F}_{\texttt{ZK}}^{\mathcal{R}_{\texttt{Tag}}}$, If $P'{\neq}P$, $Q'{\neq}Q$, and $E'{\neq}E$, or the relation $\mathcal{R}_{\texttt{Tag}}$ does not hold, send $(\texttt{sid},\texttt{abort})$ to $\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}}$. Otherwise, $\mathcal{S}$ chooses $\theta=2^{\ell}-1$ and then sends the message $(\texttt{sid},\texttt{input},k=(a,b),\theta)$ to $\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}}$.

In this simulation, the difference between the simulated execution of $\mathcal{A}$ and the real one is only in the choice of the input $x=1$. As a result, the only messages that are different are B_i and P_i ,$\forall i\in[\ell]$. Recall that B_i is the OT message defined in the OT protocol in Section 2.4.5. By relying on the UC proof of [HL17] (when simulating the corrupted sender), we can show that B_i are computationally indistinguishable from both views under the CDH problem. The indistinguishability of P_i follows directly since $P_i=B_i^r$, and r is chosen uniformly random in $\mathbb{Z}_p$. This proves that $\Pi_{\texttt{TAG}}^{\mathbb{G}}$ UC-realizes the functionality $\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}}$ in the $\mathcal{F}_{\texttt{ZK}}^{\mathcal{R}}$-hybrid model if $\mathcal{A}$ statically corrupts T under the CDH assumption.

We finally deduce that, $\texttt{IDEAL}_{\mathcal{F}_{\texttt{TAG}}^{\mathbb{G}},\mathcal{S},\mathcal{Z}} \overset{s}{\approx} \texttt{EXEC}_{\Pi_{\texttt{TAG}}^{\mathbb{G}},\mathcal{A},\mathcal{Z}}$ under static corruption and under the Inv-DDH assumption which proves our theorem.

4.10 Conclusion on Verifiable SA

In conclusion, we proposed a new formal definition for verifiable secure aggregation protocol. Our new definition captures the privacy of the user inputs and the verifiability of the aggregation result in the malicious model. Moreover, we presented VSA protocol and proved that it satisfies these security guarantees. VSA improves PUDA protocol which originally provides a verification mechanism against a malicious aggregator. VSA extends it by generating the PUDA tags using a distributed tagging protocol. Our distributed tagging protocol runs between a user and m taggers. It guarantees that an honest user receives a valid tag of its input and that a malicious user receives nothing. Thanks to this tagging protocol, VSA guarantees that an adversary controlling the aggregator and few users cannot compromise the aggregation result.

Part II

Secure Aggregation for Federated Learning

Chapter 5

Privacy-Preserving Federated Learning with Secure Aggregation

In this chapter, we study federated learning, one of the recent technologies used in IoT platforms. We specifically discuss its limitations in terms of privacy and we propose to improve its security using secure aggregation. For this purpose, we study all existing solutions that integrate secure aggregation within federated learning and we regroup them based on the specific challenge they tackle. We finally derive some takeaway messages that would help for a secure design of federated learning protocol and identify research directions in this topic.

5.1 Introduction to Privacy-Preserving Federated Learning

With the recent advancements in information technologies, machine learning techniques take a substantial part of data processing. Machine learning is a set of techniques that uses real data (e.g., measurements) to improve the accuracy and performance of existing systems [Mit97]. These techniques aim to develop the so-called machine learning models by learning to perform some well-specified tasks. This operation is known as training machine learning models on collected data. As a consequence, data becomes the most important resource for such systems to achieve better accuracy.

In the case of IoT platforms, machine learning is an essential technique to improve their business targets. IoT platforms collect data from IoT devices and process them using machine learning techniques. Lately, these IoT platforms started to collaborate with each other to train better machine learning models. However, they cannot simply share their collected data due to privacy reasons. Thus, federated learning technology emerged as a privacy-preserving technique to train on private datasets from multiple sources.

The term federated learning (FL) was initially introduced by McMahan et al. [MMR+17] and refers to a technology that enables training machine learning models on data from different sources without the need to store the data at a central location. Federated Learning is performed in several rounds with n clients and a server. A FL

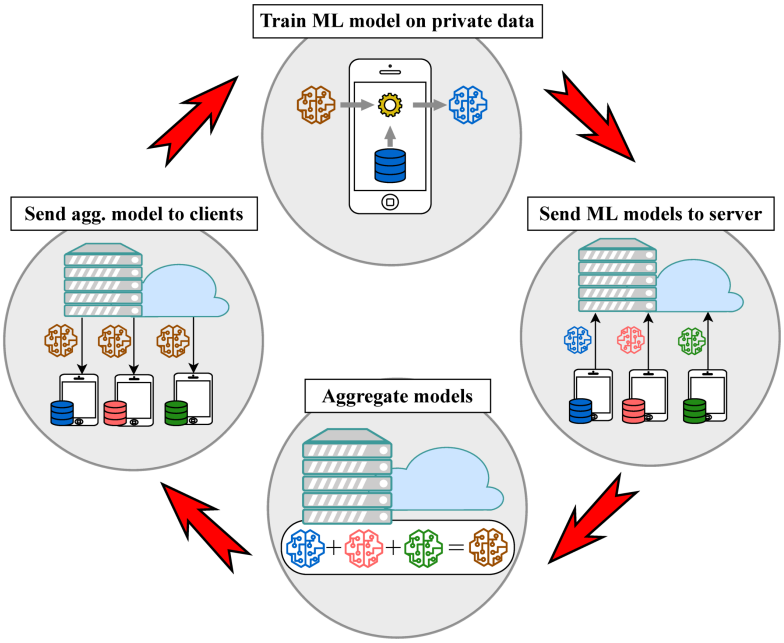


Figure 5.1: One federated learning round with three FL clients and one server.

client i (the IoT platform in our case) holds a dataset D_i. In the beginning, the FL server initiates the same model M for all clients. At each FL round, the client i receives the model M^τ from the server and trains it on D_i which results in the trained model M_i^τ. The client sends the updated model to the server. The FL server aggregates the trained models received from clients by averaging them and then sends the and sends the aggregated model $M^{\tau+1}$ back to the clients. Once a client receives the aggregated model, a new FL round starts where clients and the server repeat the steps. FL stops when the aggregated model converges. Figure 5.1 illustrates one FL round.

The main goal of FL is to protect the privacy of the local data while still being able to use them for training public models. This technology provides a great advantage over other techniques that try to achieve the same goal (eg., training on encrypted data [VNP+20, HTGW18, WGC19, DGBL+16, WTB+20]). The latter adds a large computational overhead since it involves encryption of the inputs then performing complex computations on encrypted data. FL requires less computation as it only involves the averaging operation at the server.

While FL is proposed for privacy-protection purposes, it lacks a formal guarantee of privacy. For example, adversaries who have access to the training results sent from each client to the server might be able to infer a training sample from a client's private dataset. Many types of inference attacks on FL are investigated and researched in [MSDCS19, ZLH19, LHCH20, NSH19]. One of the solutions to mitigate these inference attacks is secure aggregation. In this chapter, we study the use of secure aggregation to perform the averaging operation in federated learning. To better understand the integration of secure aggregation in federated learning, we study the specific requirements

and characteristics of federated learning compared to the legacy application of secure aggregation. Additionally, we survey all 37 existing solutions that propose to do this integration and we regroup them based on the specific challenge they tackle. As a result, we identify the limitations and gaps in the literature and present them as a set of take-aways.

5.2 Characteristics of Federated Learning

Federated learning has a wide range of use cases and applications. These applications differ based on the scale of federation, the partitioning of the training data, and the learning algorithm used in training.

5.2.1 Scale of Federation

Two types of FL exists with respect to the scale of federation: *Cross-silo* FL and *cross-device* FL [KMA+19].

- *Cross-silo Scenarios (X-Silo):* A small number of powerful users host the data. These often have decent computational power with a reliable and high bandwidth network connection.
- *Cross-device Scenarios (X-Device):* It involves a large number of users. These users often correspond to end-devices with moderate computational power. In many applications, these devices directly interact with end-users from which they collect data.

5.2.2 Partitioning of the training data

There exist three categories of data partitioning [YLCT19, LWH19, DP21]: *Horizontal partitioning*, *vertical partitioning*, and *hybrid partitioning*.

- *Horizontal Partitioning:* Each FL client holds a set of complete training samples. Each sample contains all the training features and the corresponding label. Hence, each client can train a local model on these samples.
- *Vertical Partitioning:* A client may hold part of the features of each training sample while the other parts might be held by other FL clients. In this FL type, the clients are not able to locally train a model without collecting the missing information of each sample from other clients.
- *Hybrid Partitioning:* A hybrid partitioned dataset is a combination of horizontally and vertically partitioned datasets.

Secure aggregation is only suitable to FL based on horizontally partitioned datasets since those based on vertically partitioned datasets require more operations than just summing the clients' updates. In this thesis, we only consider horizontally partitioned datasets since it is the most realistic case in real-life applications.

5.2.3 Learning algorithm

The most used learning algorithm for horizontal FL is *Federated Averaging* [MMR+17], which is based on Stochastic Gradient Descent (SGD) [iA93]. SGD is an iterative algorithm used to train a model on a dataset (i.e., find the best weights of a model that can fit the dataset). At each SGD step, client i uses model M^τ and a loss function f to compute gradient g_i^τ from the values in its dataset D_i:

$$g_i^\tau = \Delta f(M^\tau, D_i) = \Delta \sum_{(x,y)\in D_i} f(M^\tau, x, y)$$

Then, the gradient is used to update the weights of the model with learning rate η ($M_i^\tau = M^\tau - \eta g_i^\tau$). The FL clients send their new trained model M_i^τ to the FL server who aggregates them:

$$M^{\tau+1} \leftarrow \frac{\sum_1^n M_i^\tau}{n}$$

Finally, each FL client obtains the aggregated model $M^{\tau+1}$ and starts a new federated learning round.

5.3 Privacy of the Datasets in Federated Learning

An adversary having access to the model update M_i^τ sent by client i can perform inference attacks. These attacks allow an attacker to retrieve some private information about the client's datasets. Based on the type of private information, there exist three categories of inference attacks:

- *Membership Inference Attacks [SSSS17a, NSH19]:* The attackers learn whether a specific data record is part of the training dataset or not.
- *Reconstruction Attacks [DN03, WLW+09]:* The attacker learns some of the attributes of a record in the dataset. These attacks are also known as model invasion attacks [FJR15].
- *Data Properties Inference Attacks [AMS+15, GWY+18]:* The attacker learns global properties of the training dataset, such as the environment in which the data was produced.

All these attacks show that federated learning is not sufficient to preserve data privacy when used alone. To mitigate these attacks, it is crucial to protect the model updates sent by the federated learning clients while still being able to compute their aggregate.

5.4 Existing Secure Aggregation for Privacy-Preserving Federated Learning

Secure aggregation schemes aim to prevent inference attacks by hiding the model updates from any potential adversary. Based on the definition given in Chapter 3, it involves two

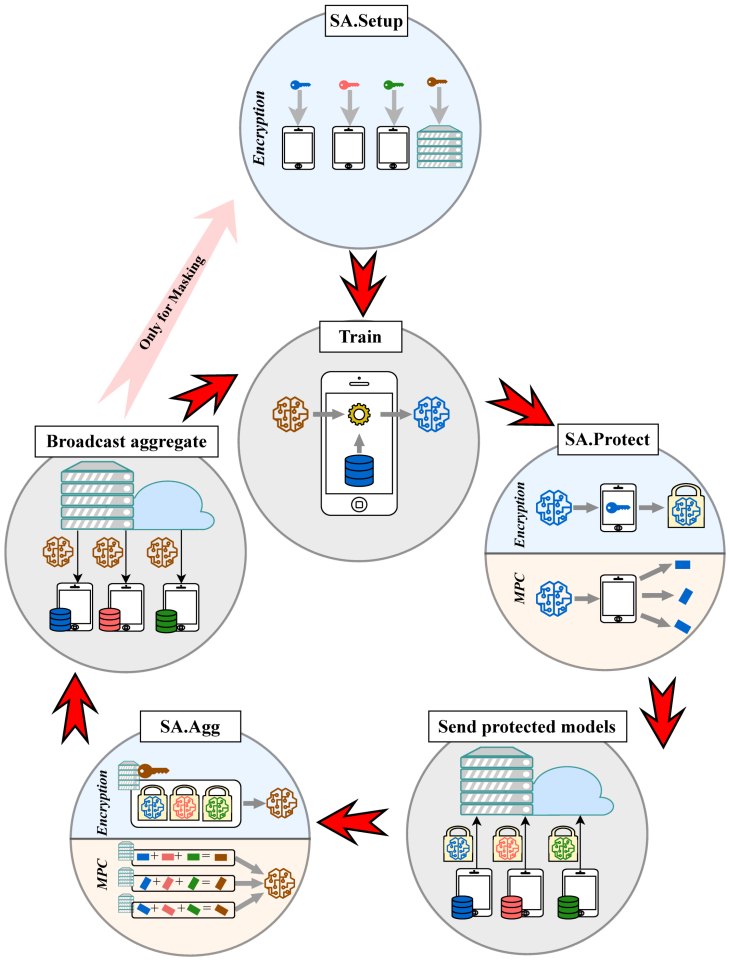


Figure 5.2: A secure aggregation protocol integrated into federated learning. The secure aggregation protocol ensures that the aggregators do not learn anything about the clients' locally trained ML models except their aggregate.

main players (i.e., users U and aggregators A) which execute the three SA phases (i.e., **SA**.**Setup**, **SA**.**Protect**, and **SA**.**Agg**). The users correspond to the FL clients and their inputs in each round are the locally trained model weights (M_i^τ). On the other hand, the aggregator (or the set of aggregators) acts as the FL server. Any secure aggregation algorithm consisting of the three defined phases can be used for running a secure version of the FL protocol. To run FL with secure aggregation, **SA**.**Setup** phase is performed before the training starts. Then for each FL round τ, client U_i trains its model on its local data and obtains the model M_i^τ. It then runs **SA**.**Protect** to protect the locally trained model and sends it to the server. Finally, the server runs **SA**.**Agg** after it collects all protected trained models. As a result, the clients get the aggregated model and starts a new FL round. Figure 5.2 shows the components of secure aggregation integrated in federated learning.

5.4.1 Challenges in using Secure Aggregation for Federated Learning

	Masking-based SA	AHE-based SA	FE-based SA	MPC-based SA
Client Failure ($\mathbb{C}_1$)	○	○	●	●
High Dim. Inputs ($\mathbb{C}_2$)	○	○	●	○
Float Inputs ($\mathbb{C}_3$)	○	○	○	○
Scalability ($\mathbb{C}_4$)	○	●	●	○
Privacy Leaks ($\mathbb{C}_5$)	○	○	○	○
Malicious Users ($\mathbb{C}_6$)	○	○	○	○
Malicious Agg ($\mathbb{C}_7$)	○	○	○	○

Table 5.1: Challenges in using secure aggregation for FL based on the specific requirements of FL. It shows for each challenge whether the baseline SA protocols defined in Chapter 3 can originally cope with that challenge.

Federated learning features some unique properties and characteristics that differ from previous applications where secure aggregation was used. This makes integrating secure aggregation schemes to federating learning a challenging task. We hereby identify seven unique properties for FL that raise significant challenges for the integration of secure aggregation in FL. We further analyze the suitability of each secure aggregation category (see Chapter 3) to cope with these characteristics. We summarize the results in Table 5.1.

Failures and Drops of Clients at Realtime ($\mathbb{C}_1$) In cross-device FL scenarios, it is common to have mobile, unreliable FL clients. The mobility of a client may cause failures (drops) of some FL clients causing their unavailability for some federated learning rounds. Failures of clients may even happen within the FL round as well. All this can be a problem for some secure aggregation schemes that do not support dynamic users. In particular, SA schemes based on masking and AHE are not fundamentally designed to cope with user failures. Therefore, the need for fault-tolerant secure aggregation is a requirement for FL.

Client's Inputs are Vectors of High Dimension ($\mathbb{C}_2$) In FL, the user's input is a vector that holds all the model parameters (weights). Not all types of secure aggregation protocols can efficiently work with vectors. For example, MPC-based SA incurs a significant communication overhead since the shares of the inputs have the same size of the input. Therefore, it is not practical to run secret sharing to share large vectors. Also, in masking-based SA, the users should run a key-agreement protocol to compute new masks for each FL round. This adds significant overhead. Additionally, for AHE, the encryption algorithm results in large ciphertexts. Thus, encrypting each element in the input vector adds a large overhead. This calls for efficient packing techniques designed for AHE-based SA. Unlike other methods, for FE-based secure aggregation, it is possible to construct a solution where each party sends exactly one value of the size of the input vector (see the SA scheme construction presented in Section 3.4.2.3 in Chapter 3).

Client's Inputs are of Floating Point Type ($\mathbb{C}_3$) All the baseline secure aggregation protocols are designed to operate on integer types. In FL, the user's input usually is of floating point (float) type. This calls for efficient quantization techniques that transform floats to integers while preserving a high accuracy in the result. Representing floats with integers incurs an increase in the size of the input. Hence, the use of quantization techniques helps achieve a good trade-off between accuracy and communication overhead.

Huge Number of Clients ($\mathbb{C}_4$) Recently, we start to observe FL applications involving thousands of FL clients. Google is researching how to train Gboard (the Android's keyboard application) search suggestion system using federated learning on large scale [YAE+18, HKR+18]. With secure aggregation integrated with federated learning, the scalability problem becomes a serious challenge. MPC-based SA protocols do not scale well with huge number of users since they suffer from a quadratic complexity in terms of communication and computation. Similarly, masking-based SA suffers from a quadratic complexity in the setup phase. Additionally, with a large number of clients, the typical synchronized FL protocol is not practical. In an asynchronous FL protocol, clients do not wait for the updates of a sufficient number of users at each FL round. Instead, the updates of the users are incorporated as soon as they arrive at the server. Adopting SA for asynchronous FL is challenging because updates may be protected with keys corresponding to different FL rounds.

Privacy Attacks that Bypass SA ($\mathbb{C}_5$) The aggregated model $M^{\tau+1}$ is a public information that is accessible for all FL clients. Therefore, secure aggregation is not used to hide this value. There exists a different type of inference attacks that can still infer private information from the aggregated model, only [SSSS17b]. For example, recently So et al. [SAG+21] pointed out a new attack to leak the client's updates even when protected with secure aggregation. The authors notice that the models from the FL clients do not change a lot between one training step and another one when the trained model starts to converge. This causes a privacy leakage if a FL client did not participate. In more details, if all FL clients participate in round $\tau - 1$ and all clients except one participate in round τ, and if the inputs did not change a lot, an adversary who has access to the aggregated model updates for rounds τ and $\tau - 1$ will be able to approximate the inputs of the missing FL client. Such specific attacks can bypass the security measures of SA. Gao et al. [GHG+21] implemented these types of attacks and show how they can effectively infer the category of the given data samples.

Secure aggregation protocols by definition do not provide protection against these types of attacks. Therefore, additional security mechanisms should be used with secure aggregation to mitigate these attacks.

Malicious Users ($\mathbb{C}_6$) Earlier SA protocols proposed before the FL paradigm appeared, mainly consider a *honest-but-curious* threat model with *colluding* users (see Section 3.2 in Chapter 3). Such a threat model is not sufficient in the context of federated learning. Specifically, FL clients cannot be trusted to provide their inputs truthfully at each FL

round. Thus, we should consider an extended threat model which considers malicious users.

Indeed, poisoning attacks (a.k.a., backdooring attacks) are attacks where malicious FL clients manipulate their model updates M_i^τ to affect the aggregated model $M^{\tau+1}$. Their goal is to install a backdoor in the trained model. A "backdoored" model behaves almost normally on all inputs except for some attacker-chosen inputs at which it outputs attacker-desired predictions. Malicious FL clients use two main methods to poison a model: Dataset poisoning [STS16] where attackers insert malicious records in their dataset; and model poisoning [BVH+20a] (a.k.a., constrain-and-scale attacks) where the attacker replaces the trained model by a malicious model and send it instead of the trained model. An even more recent attack method consists of distributed poisoning attacks [XHCL20] in which the poison is distributed among several malicious clients inputs so that it is harder to detect malicious models. On the other hand, malicious clients can perform less stealthy attacks by sending ill-formed inputs to prevent the calculation of the aggregation. To further prevent all these types of attacks we need to implement additional security mechanisms for SA.

Malicious Aggregator ($\mathbb{C}_7$) Similar to challenge $\mathbb{C}_6$, the *honest-but-curious* threat model is not sufficient to prevent the cheating of a server in the context of federated learning. More specifically, the SA protocols described in Section 3 prevent a curious FL server from learning the clients inputs, but cannot protect against a malicious server that modifies the aggregated model. Indeed a malicious server can cause a huge damage because it has full control of the final aggregated value. Therefore, an adversary controlling the FL server can force the clients to receive an adversary chosen model. In fact, the impact of a malicious aggregator can even go beyond forging the aggregation result. Pasquini et al. [PFA21] showed that a malicious aggregator can even compromise the privacy by bypassing the secure aggregation protocol. An example for these attacks illustrated by the authors is when the malicious aggregator chooses specific values for the aggregated result. The values are chosen such that when the clients train the forged model sent by the aggregator, the training outputs a model of zero parameters. Hence, the malicious aggregator can suppress arbitrary clients of his choice from the aggregation by sending them malformed models. Therefore, it can suppress all clients except a targeted one and leak its input. To prevent such attacks, SA protocols should consider a malicious aggregator in their threat model.

5.4.2 Secure Aggregation Solutions for Federated Learning

A lot of research has been conducted on designing secure aggregation protocols based on cryptographic schemes for federated learning applications. Most of the proposed schemes are improvements of the basic secure aggregation protocols described in Chapter 3 and tackle one or more of the aforementioned challenges ($\mathbb{C}_1$-$\mathbb{C}_7$). The proposed schemes can be categorized based on the challenge they tackle. We summarize how these solutions propose different solutions for each of the challenges. Table 5.2 presents an overview of these solutions grouped by their challenge scope. Also, Figure 5.3 regroups them per SA

Scope	Solution (year)	SA scheme	FL scale	Technique
Fault-Tolerance ($\mathbb{C}_1$)	Bonawitz et al. [BIK+17]	Masking	X-Dev	Integrating Shamir SS with Masking
	HybridAlpha [XBZ+19]	MIFE	X-Silo	Assigning zero weights for dropped clients
	FastSecAgg [KRKR20]	MPC-t-of-n	X-Dev	Using FFT with Shamir SS
	Stevens et al. [SSV+21]	Masking	X-Dev	Integrating Shamir SS with LWE-based Masking
	LightSecAgg [YSH+21]	Masking	X-Dev	Integrating MDS code with Masking
Comm. Efficiency ($\mathbb{C}_2$)	Phong et al. [PAH+18]	AHE	X-Silo	Batch encryption
	Liu et al. [LCV19]	AHE	X-Silo	Batch encryption
	BatchCrypt [ZLX+20]	AHE	X-Silo	Batch encryption
	Wu et al. [WPX+20]	MIFE	X-Silo	All or nothing transformation
	Bonawitz et al. [BIK+17]	Masking	X-Dev	Generate masks from small seeds
Accuracy ($\mathbb{C}_3$)	Bonawitz et al. [BSK+19]	Masking	X-Dev	Auto-tuned quantization
	HeteroSAg [EA20]	Masking	X-Dev	Auto-tuned quantization
	EastFly [DCSW20]	MPC-n-of-n / AHE	X-Silo	Quantization: Ternary FL
	Safer [BT20]	MPC-n-of-n	X-Silo	Top-k sparsification with 1-bit quantization
Scalability ($\mathbb{C}_4$)	Bonawtiz et al. [BEG+19]	Masking	X-Dev	Sub-grouping: Running multiple SA instances
	Bell et al. [BBG+20]	Masking	X-Dev	Sub-grouping: Creating random connected graphs
	TurboAgg [SGA21b]	Masking	X-Dev	Sub-grouping: Circular subgroups of clients
	SAFE [SMH21]	Masking	X-Dev	Arranging clients in a circular chain
	So et al. [SAGA21]	Masking	X-Dev	Adapting SA for asynchronous FL
	SwiftAgg [JNMALC22]	Masking	X-Dev	Sub-grouping: Running multiple SA instances
Privacy Enhancing ($\mathbb{C}_5$)	Truex et al. [TBA+19]	AHE	X-Silo	DDP: clients add gaussian noise
	Peter et al. [KLS21]	Masking	X-Dev	DDP: clients add gaussian noise
	So et al. [SAG+21]	Masking	X-Dev	Multi-round privacy using client selection
	Timothy et al. [SSV+21]	Masking	X-Dev	DDP: clients use LWE-based masking
	Joaquín et al. [FMLF21]	Masking	X-Dev	DP: aggregator add noise to the aggregate
Verify Inputs ($\mathbb{C}_6$)	MLGuard [KTC20]	MPC-2-of-2	X-Silo	Boolean circuits to compute cosine distance
	FLGuard [NRY+21]	MPC-2-of-2	X-Silo	Bool/Arth circuits to perform clustering
	RoFL [BLV+21]	AHE	X-Silo	Commitment scheme to compute euclidean distance
	BREA [SGA21a]	Masking	X-Dev	Arithmetic circuits to compute square distance
	Karakoc et al. [KOB21]	AHE	X-Silo	OPPRF to compare with a threshold
	SAFELearning [ZLYM21]	Masking	X-Dev	Multi-step aggregation to verify intermediate results
	Velicheti et al. [VXK21]	Masking	X-Dev	Multi-step aggregation to verify intermediate results
Verify Agg. ($\mathbb{C}_7$)	Zhang et al. [ZFW+20]	AHE	X-Silo	Using HHF based on Bilinear Maps
	VerifyNet [XLL+20]	Masking	X-Dev	Using HHF with ZKP scheme
	VERSA [HKKH21]	Masking	X-Dev	Using keyed HHF with ZKP scheme
	NIVA [BTL+21]	MPC	X-Dev	Verifiable secret sharing
	DEVA [TLB+21]	MPC	X-Dev	Verifiable secret sharing
	VeriFL [GLL+21]	Masking	X-Dev	Using commitment scheme with HHF

Table 5.2: Categorization of secure federated learning solutions based on the challenge tackled with a short description of the proposed solution. All the solutions are secure in the honest-but-curious model except those addressing $\mathbb{C}_7$ (malicious aggregator) and $\mathbb{C}_6$ (malicious users) thus addressing a specific malicious setting. An exception is for the solutions in red which do not protect against collusions between users and aggregators and thus are considered not secure (based on our security definitions in Chapter 3).

category and shows the relation between the solutions.

Fault-tolerant Secure Aggregation To tackle the problem of client failures (see $\mathbb{C}_1$), a fault-tolerant secure aggregation protocol should be used.

MPC-based SA, more specifically, Shamir's SS scheme [Sha79] is fault-tolerant by design. It is used in [DCSW20, KRKR20] where the FL server role is distributed among

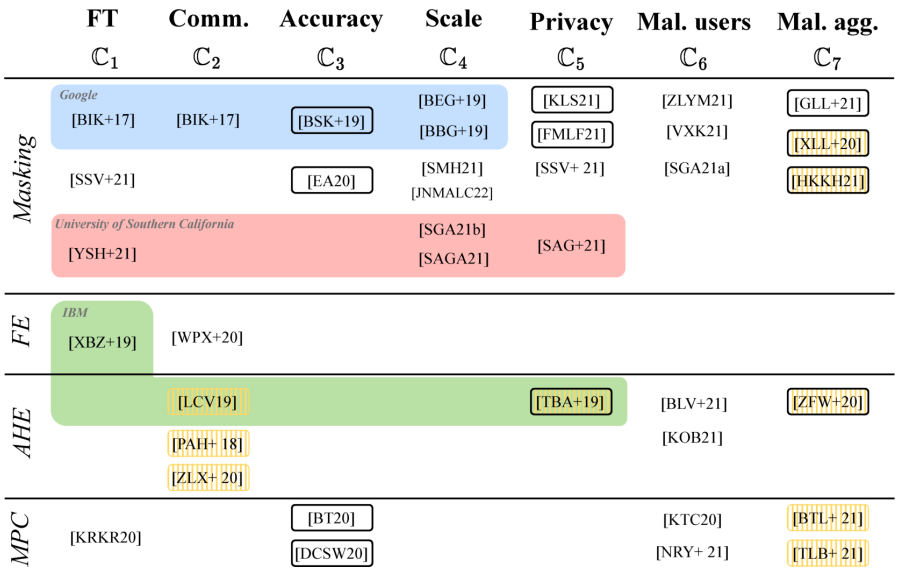


Figure 5.3: Summary of existing FL solutions that use crypto-based secure aggregation grouped by the type of secure aggregation used and the specific challenge they tackle. Bordered boxes indicate that the solution presents a technique that can be deployed in other types of SA protocols (eg., [XLL+20] is implemented on masking-based SA but can be also used for AHE-based SA). Hatched boxes indicate that the scheme cannot achieve the security requirements since they do not support collusions (this is discussed in Section 5.5). Different colors represent research groups of the authors.

multiple aggregators. The high communication cost that these solutions incur, encourages researchers to look for alternative fault-tolerant solutions.

MIFE-based SA also are fault-tolerant by design since the data aggregator can assign zero weights for missing clients [XBZ+19]. However, these schemes require a key dealer to stay online for each federated learning round.

On the other hand, Bonawitz et al. [BIK+17] propose a fault-tolerant variant of the **masking-based SA**. Later, this scheme is widely adopted and improved by [BBG+20, EA20, SGA21b, KLS21, XLL+20, GLL+21]. The idea of this scheme is to merge Shamir's SS scheme with masking. More specifically, it benefits from the lightweight operations and low communication overhead of the masking scheme and on the other hand, the solution inherits the fault-tolerance property of Shamir's SS scheme. Thanks to this trade-off, it is considered a big jump towards designing practical secure aggregation scheme for cross-device FL scenarios. A similar scheme is proposed by Stevens et al. [SSV+21] that replaces the standard masking with a Learning With Error masking and uses a packed and verifiable version of Shamir's secret sharing. Also, Yang et al. propose LightSecAgg [YSH+21] which replaces the Shamir's secret sharing scheme with a secret sharing scheme based on Maximal Distance Separable (MDS) code [RL89]. The work of Yang et al. reduces the computation time at the server. Another approach is proposed by Swanand et al. [KRKR20] that uses Fast Fourier Transform (FFT) for secret sharing.

Communication Efficient Secure Aggregation Researchers propose some techniques to bound the communication overhead incurred by SA (see $\mathbb{C}_2$).

For **AHE-based SA**, batch encryption has been leveraged by Liu et al. [LCV19], Phong et al. [PAH+18], and Yang et al. [ZLX+20]. Batch encryption allows encrypting of multiple values in a single operation and thus optimizing the encryption of vector inputs (representing machine learning models). In BatchCrypt [ZLX+20], the authors propose a method to quantize and batch the elements of a model before encryption. The strength of their approach is that it preserves the additively homomorphic property of the ciphertexts. Another interesting technique presented by Wu et al. [WPX+20] is to use the All Or Nothing Transformation (AONT) [Riv97]. AONT is a technique for transforming data into a different form such that, the new data can only be understood if all of it is known. The authors show that by transforming clients' models with AONT, it is sufficient to encrypt a small part of the transformed model. Thanks to AONT property, the non-encrypted part of the transformed model does not give any useful information as long as the other part of it is encrypted. This can decrease the size of the protected user input by several orders.

For **masking-based SA**, Bonawitz et al. [BIK+17] propose to execute a key agreement protocol to produce small random numbers that are used as seeds of a pseudo-random generator. These seeds are then used to generate the masks.

Accurate Secure Aggregation To deal with the floating point challenge while preserving good accuracy, researchers propose to use different quantization techniques. Elkordy et al. [EA20] and Bonawitz et al. [BSK+19] propose to use auto-tuned quantization. This technique allows adapting the quantization level of the model vector based on the requirements. More specifically, for some elements of the trained model that do not have a large impact on its accuracy, this technique reduces the quantization level which consequently reuced the communication cost. Auto-quantization is integrated with FT-Masking [BIK+17]. Alternatively, in [BT20], Beguier et al. propose a quantization technique called TopBinary quantization. This technique is essentially a combination of top-k sparsification [SCJ18] with 1-bit quantization [BWAA18]. We observe that these quantization techniques is that they can be used for all categories of secure aggregation protocols.

Scalable Secure Aggregation To tackle challenge $\mathbb{C}_4$, scalability of SA started to gain researchers' attention thanks to the new large-scale applications of FL. Bonawitz et al. [BEG+19] set up a general framework to scale a secure aggregation framework to millions of devices. The authors propose to simply run multiple instances of the scheme, one for each subgroup of clients. Each subgroup computes intermediate aggregates which are combined later. The same intuition of grouping clients is followed up by Bell et al. [BBG+20] and by So et al. [SGA21b]. Bell et al. observe that the FT-Masking scheme in [BIK+17] does not require that all the clients are connected. Thus, they propose to generalize the scheme by creating random graphs. Each FL client executes the FT-Masking with its neighbors. The new protocol assumes that not all the neighbors will be corrupted at the same time and it proposes a method to build the so-called "good" graphs.

Similarly, both So et al. (TurboAgg) [SGA21b] and Sandholm et al. (SAFE) [SMH21] propose a circular topology. Clients perform a chain of aggregations by passing the aggregated updates to the next client. To further deal with a large number of clients, So et al. [SAGA21] propose a SA protocol that can be integrated into asynchronous FL. The solution uses the scheme proposed in [YSH+21] and adapts it to enable secure aggregation of inputs from different timestamps.

Secure Aggregation Resilient to Privacy Attack To deal with inference attacks on the aggregated model ($\mathbb{C}_5$), Differential Privacy (DP) solutions [Dwo06] should be used with secure aggregation. Notice that the goal of using DP mechanism with SA is to protect the aggregated model, only, and not user inputs.

A simple method is to let the aggregator apply DP mechanism on the aggregated model [FMLF21]. However, this requires trusting the aggregator. A better method is to use a distributed version of DP (DDP) along with SA to mitigate the information leakage caused by the public aggregated model. Few works have followed this approach for FL [KLS21, TBA+19]. These solutions add Gaussian noise to the FL clients' inputs. They leverage the fact that FL clients' inputs are protected with cryptographic tools (thanks to SA) which permit them to decrease the level of noise while achieving sufficient privacy level. Therefore, using DDP with SA limits the degradation of the accuracy of the trained model compared to using DP alone. Stevens et al. [SSV+21] follow a similar approach by using Learning with Error (LWE) masking technique to make the final aggregate differentially private.

On the other hand, a multi-round privacy concept is introduced by So et al. [SAG+21]. This concept is to ensure that an adversary cannot learn valuable information by monitoring the changes in the aggregated model across different FL rounds. The authors propose a solution enlightened by the work in [TNW+21]. They propose to randomly and fairly (using weights) select participants in each FL round based on well-defined criteria called *Batch Partitioning*. Using this technique they can guarantee the long-term privacy of the data at the FL clients.

Secure Aggregation Against Malicious Users To deal with malicious users who perform *poisoning attacks* ($\mathbb{C}_6$), the FL server needs a mechanism to validate the inputs of the clients. Mitigating poisoning attacks is studied by researchers independently from using secure aggregation for FL [FYB18, AMMK20]. One of the methods used to prevent such attacks is to use the cosine distance [FYB18] to detect poisoned inputs that deviate from the other benign inputs. Clustering [STS16, BEMGS17] and anomaly detection methods [AMMK20] are also used to detect malicious model updates. An orthogonal approach is to use clipping and noising to smooth the model updates and remove the differences [BVH+20b]. While all these solutions are shown to be efficient in preventing poisoning attacks, using them with secure aggregation is a big challenge. The problem is that all these solutions rely on analyzing the FL clients' inputs while secure aggregation aims to hide and protect these inputs. Several methods are proposed to verify the inputs while keeping them protected to preserve their privacy.

For **MPC-based SA**, it is possible to build circuits that can perform complex operations

on the shares. This can be used to evaluate functions on the inputs other than just computing the sum. Indeed, MLGuard [KTC20] proposes to verify the users' inputs by transforming a verification function into a circuit that gets executed by the two servers using 2PC. The verification function computes the distance between the clients' inputs. The circuit compares the distance to pre-defined thresholds and thus rejects the input if it exceeds it. FLGuard [NRY+21] follows the same approach by building two circuits: One circuit for detecting poisoned inputs using a dynamic clustering algorithm (HDBSCAN [CMS13]) and another circuit for reducing the impact of poisoned inputs using clipping and noising. The communication cost of running these circuits is significant thus making scalability even harder to achieve for SA in the federated learning context. A promising approach to reduce this cost is through the use of *secret-sharing non-interactive proof (SNIP)*. This approach was proposed in [CGB17] (Prio). Using SNIP enables the aggregators to validate the user inputs without interacting with the users and with minimal interaction between themselves. This scheme is not yet deployed in FL applications. *SNIP* brings a great advantage over standard 2-PC validation circuits since it does not limit the number of aggregators thanks to its lower communication cost. The limitation of SNIP is that it only supports specific validation functions. Therefore, it is an open challenge to design validation circuits for detecting poisoning attacks using SNIP.

On the other hand, regarding **AHE-based SA**, Karakoc et al. [KOB21] propose *OPPRF*, an algorithm based on private set membership (PSM) [CO18] and oblivious transfer (OT) [NP05]. *OPPRF* uses PSM to perform equality checks between values (i.e., equivalent to finding an intersection between sets of cardinality equal to one [Cou18]). Using *OPPRF*, the users can create tags that are only valid if their inputs are lower than a threshold provided by the aggregator. Karakoc et al. [KOB21] applied this scheme for AHE-based SA schemes and evaluated it in FL applications. The scheme enables the FL server to detect poisoning attacks by checking that the minimum, maximum and average of the model elements do not cross a certain threshold value. The threshold is configured based on an observation of the models of benign clients. Another approach is proposed by Lukas et al. [BLV+21]. The authors use a *non-interactive commitment* scheme proposed in [Ped92]. Using this scheme, the users create proofs that the Euclidean distance of their inputs satisfies the bound set by the aggregator. Upon receiving the client's protected input and the commitment, the server verifies that the proof is valid.

For **masking-based SA**, two techniques are proposed. One technique is proposed by So et al. [SGA21a] in which users secretly share their model updates with all other clients and then compute the squared distance between the model shares. The server can finally reconstruct the squared distances and use the result to detect malicious inputs. An alternative technique is proposed by Zhang et al. [ZLYM21] and Velicheti et al. [VXK21]. In more detail, users are anonymously and randomly grouped into clusters. Aggregation happens per cluster and then a following round of aggregation happens on the results of each cluster. For each cluster, the intermediate aggregation results are checked to prevent poisoning attacks. The fact that attackers do not know to which cluster the compromised device belongs to, protects from distributed poisoning attacks (see $\mathbb{C}_6$).

Secure Aggregation Against Malicious Aggregator In a malicious aggregator threat model, the FL server forges false aggregation results ($\mathbb{C}_7$). Mitigating these attacks requires a verifiable secure aggregation scheme. Many solutions are proposed to enable the verification of the aggregation outcome [KShS12, SS11, DOS18, CDE+18]. However, these solutions do not fit well in federated learning applications due to their high communication overhead. In the context of federated learning, six solutions are proposed [GLL+21, HKKH21, XLL+20, ZFW+20, TLB+21, BTL+21].

For **masking-based** and **AHE-based SA**, Zhang et al. [ZFW+20] and Xu et al. (VerifyNet) [XLL+20] use *Homomorphic Hash Functions (HHF)* to verify the result of the aggregation. *HHF* can be built using bilinear maps [BGLS03a, Fre12b]. First, the data owners create authentication tags for their inputs and send them to the aggregator. The latter aggregate them to prove the outcome of the aggregation. Finally, the aggregator verifies the result. Hahn et al. [HKKH21] detect possible brute-force attacks on VerifyNet and improve it by deploying a keyed *HHF*. All these solutions do not prevent collusions between the users and the aggregator. Therefore, they cannot be considered secure based on our security definitions for secure aggregation in Chapter 3). Another problem with these solutions is that they significantly affect the performance of SA. This is because of the linear increase in computation and communication overhead with the increase of the dimension of inputs. This is a clear limitation since the performance of the ML model highly depends on its size (i.e., number of parameters). To solve this problem, Guo et al. (VeriFL) [GLL+21] focus on designing a verification scheme specifically for secure aggregation applications with inputs of high dimension. To support user-aggregator collusions, the authors integrated a *commitment* scheme to prevent users from changing their hashes after the computation of the aggregate. The authors of VeriFL apply this scheme to the fault-tolerant masking scheme in [BIK+17]. The evaluation of their solution on federated learning applications shows a significant reduction in communication overhead with respect to other verification schemes. However, VeriFL still suffers from a quadratic computation and communication cost with respect to the number of FL clients. Achieving better scalability for verification systems is an open problem.

For **MPC-based SA**, Brunetta et al. [BTL+21] propose NIVA as a non-interactive secure aggregation protocol that includes the verification of the result. The users create a tag for each of their input shares. Upon computing the aggregate, the result can be verified using all the generated tags. Tsaloli et al. [TLB+21] propose DEVA which improves the number of tags created for each user. DEVA requires that a user creates a single tag for its input rather than creating a tag for each share. Both approaches do not support collusions between users and the aggregator and incur very high communication overhead since they use MPC.

5.5 Observations and Conclusion on Secure Aggregation for FL

We have extensively studied federated learning solutions that integrate secure aggregation schemes. In this section, we identify and share the following observations and takeaway messages:

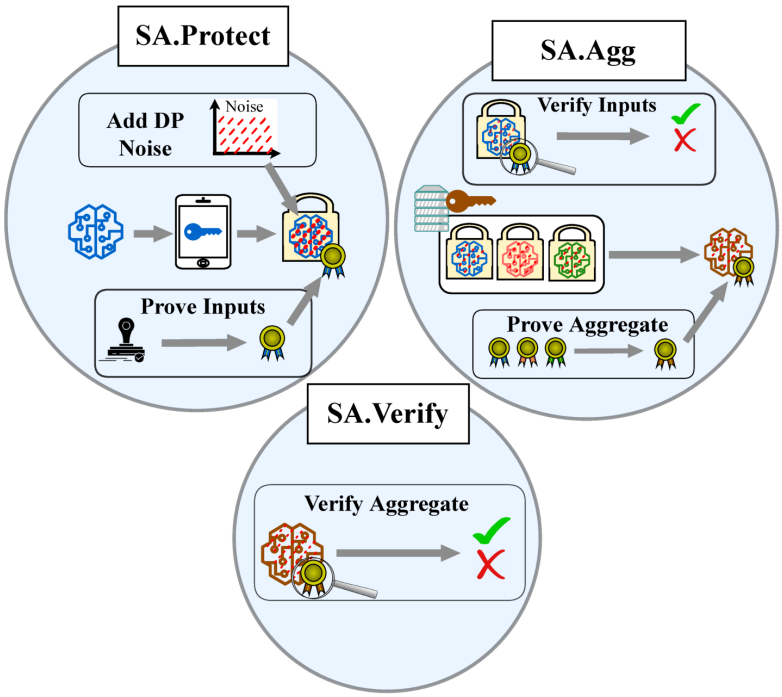


Figure 5.4: New Components of Secure Aggregation

$\mathbb{O}_1$ We can observe that masking-based SA is the most integrated secure aggregation solution for federated learning. More specifically FT-Masking [BIK+17], appeared in 20 solutions where each one tries to improve it in a certain direction. It is interesting to see all these parallel improvements integrated into a single solution.

$\mathbb{O}_2$ We notice that secure aggregation solutions based on AHE are not widely adopted in federated learning. This is mainly because they do not support user dynamics. However, we see that AHE-based SA is promising since they provide long-term security using the same user keys. We hope to see more research improving these schemes towards a practical deployment in federated learning context.

$\mathbb{O}_3$ We notice that some of the solutions proposed to preserve the privacy in federated learning do not adhere to the minimal security requirements for secure aggregation protocols (see Definition 3.2.1 for *Aggregator Obliviousness*). Specifically, AHE schemes [PAH+18, LCV19, ZLX+20, ZFW+20] and masking-based verification schemes [ZFW+20, XLL+20, HKKH21] that use the same key for all users should not be considered secure since they do not guarantee security in case of a collusion between a user and the aggregator.

$\mathbb{O}_4$ We note that secure aggregation alone is insufficient to guarantee the privacy of the clients' datasets in the context of federated learning. Although SA helps prevent inference attacks, the global model that is collaboratively computed from private individual

inputs can still leak information. Therefore, additional protection mechanisms are required. Differentially private mechanisms and multi-round privacy solutions are suitable candidates to cope with this problem.

$\mathbb{O}_5$ Poisoning attacks against federated learning call for some integrity mechanisms that would allow the aggregator (the FL server in this context) to verify the correctness/veracity of received inputs. Nevertheless, the cost of such mechanisms can be significant. Therefore, we can consider the design of such mechanisms that can (i) detect stealthy and sophisticated poisoning attacks, and (ii) ensure the security and scalability requirements, as an open challenge.

$\mathbb{O}_6$ Similar to the previous observation, we identify the need for an integrity mechanism for verifying the correctness of the actual aggregate. Basic solutions would linearly increase the size of the transmitted data between parties w.r.t. the model size. Using incremental HHF is promising as shown in VeriFL [GLL+21]. However, this solution is still far from being applied for FL applications in larger scale since it still implies a linear increase in communication and computation cost w.r.t the number of clients.

Based on all the previous observations, we propose revisiting the definition of crypto-based secure aggregation to make it suitable for FL. Specifically, we revise the description of the protocol phases (i.e., **SA**.**Setup**, **SA**.**Protect**, and **SA**.**Agg**) to meet all the security requirements for FL applications. Following observation $\mathbb{O}_4$, the **SA**.**Protect** phase should be modified such that users first pre-process their inputs with distributed DP mechanisms before running the actual protection algorithm. Additionally, based on observation $\mathbb{O}_5$, **SA**.**Protect** should also generate integrity proofs of inputs which are sent together with the protected inputs to the data aggregators. On the other hand, **SA**.**Agg** should include a verification mechanism of the inputs which validates the integrity proofs. Moreover, observation $\mathbb{O}_6$ indicates that **SA**.**Agg** should compute a proof of the aggregation which is sent to the users along with the aggregation result. In order for the users to validate the aggregation result, we require an additional phase: Namely, the **SA**.**Verify** phase should be performed as a final step by the users. In this phase, the users verify the received result of the aggregation.

Summary In summary, we propose a better definition of secure aggregation protocols based on cryptographic schemes which cope with the security requirements of federated learning. The definition consists of four phases: **SA**.**Setup**, **SA**.**Protect**, **SA**.**Agg**, and **SA**.**Verify**. Figure 5.4 shows the details of the improvements in each of the phases. It is worth noting that this new definition combines and generalizes all the improvements proposed by the state-of-the-art solutions. It would be interesting to develop the first SA solution for FL implementing our proposed definition by combining all the state-of-the-art techniques.

Chapter 6

Scalable and Fault-Tolerant Secure Aggregation for Federated Learning

In the previous chapter, we have identified and enumerated the main challenges of integrating secure aggregation schemes in federated learning (FL) protocol. In this chapter, we tackle one of these challenges, namely, failures and drops of FL clients in real-time. We first provide an extensive study of the existing solutions against such failures. We further propose a novel fault-tolerant secure aggregation solution for federated learning (FTSA) that is the first one based on additively homomorphic encryption.

6.1 Fault-Tolerant Secure Aggregation

Consider a scenario where hundreds of companies manage IoT platforms and would like to collaboratively train a machine learning model over their private datasets. Their goal is to provide better services to their customers by sharing their resources. The different IoT platforms use federated learning to train a common model. Additionally, these platforms do not wish to rely on a trusted third party to run the federated learning server. Therefore, they integrate secure aggregation into federated learning to preserve the privacy of their customers. Each company may have multiple federated learning clients (clients may correspond to different geolocated branches or different company sectors). Consequently, running federated learning protocols on large scale (with a large number of clients) may frequently encounter client failures and dropouts. Such a use case calls for a secure and fault-tolerant federated learning solution with an untrusted server (that can be located on any of the companies' premises).

Most secure aggregation solutions fall short to address such a problem (see Chapter 6). Previous work from Bonawitz et al. [BIK+17] develops a fault-tolerant secure aggregation that enables the server to recover the aggregate from up to t out of n client failures. The authors design their solution by extending an existing masking scheme [DA16] with Shamir's secret sharing to enable fault tolerance. Their scheme has been used as a building block for a significant number of privacy-preserving federated learning solutions [BSK+19,EA20,BEG+19,BBG+20,SGA21b,KLS21,SAG+21,XLL+20,GLL+21,SGA21a].

Nevertheless, masking is based on one-time-pad encryption (i.e., modular addition) and hence requires the re-distribution of new keys for each aggregation process. This incurs a significant computation and communication overhead originating from the execution of the key re-distribution among clients and the generation of new masks at each FL round.

We, therefore, propose a new solution that enables a client to use the same key for multiple FL rounds. We revisit the Joye-Libert (JL) secure aggregation scheme proposed in [JL13] (see Section 3.4.2.2 in Chapter 3) and propose a variant that supports clients' failures. Compared to [BIK+17], we provide a more efficient and fault-tolerant secure aggregation scheme since it does not require the re-distribution of the protection keys for each FL round.

6.2 Threat Model

We consider a threat model with an untrusted FL server that colludes with some clients. Additionally, we consider some of the honest clients to unintentionally fail (i.e., drop from the protocol) in some federated learning rounds. The failures can happen at any stage of the protocol. The adversary (controlling the server and the colluding clients) is interested in discovering any private information about the individual inputs of the honest clients.

We consider the two possible settings for the adversary as defined in Section 3.2. Namely, the honest-but-curious model and the malicious model. For each of these settings, we identify the minimum required security parameters to achieve *Aggregator Obliviousness* (see Definition 3.2.1). We rely on a trusted party only to generate the public parameters of our protocol. Attacks that aim to change the result of the aggregated data or to perform some denial of service are out of the scope. Additionally, we do not consider attacks where the adversary impersonates existing clients as these can be prevented by deploying a public key infrastructure with signed certificates. Note that this threat model is common among secure aggregation protocols and it is the same as the one adopted by [BIK+17] except for the dependency on a trusted party for the setup. We show later how to avoid this dependency.

6.3 Prior Work based on Masking

In this section, we discuss the previous works on fault-tolerant secure aggregation. All the existing work focus on masking-based secure aggregation. To achieve fault tolerance, these solutions integrate different types of secret sharing with masking.

6.3.1 SecAgg

Bonawitz et al. propose SecAgg [BIK+17] as a secure aggregation scheme based on masking. The solution is briefly described in Section 5.4. We here give a more detailed description of the protocol. Every two clients agree on a mutual mask using a key-agreement scheme and use these masks to protect their inputs. To recover from some potential drops of clients, the clients secretly share their secret keys such that any t online clients can help the aggregator reconstruct the missing client's masks and remove

them from the aggregate. In more detail, in the **SA**.**Setup** phase, the users agree on mutual seeds using Diffie-Hellman key-agreement scheme similar to the case in standard masking-based SA. Additionally, they use the t-out-of-n Shamir's Secret Sharing [Sha79] to share their key-agreement [DH06] private keys. Using this approach, masks of dropped users can be recovered as long as t users are still online. While this solves the problem of dropped users, it causes a new security problem. More specifically, the aggregator can claim that an online user actually dropped and thus ask for reconstructing its masks and hence reveal the user input. To solve this problem, a double masking technique is used in the **SA**.**Protect** phase. Each user adds another layer of masking using a randomly generated mask $b_{i,\tau}$:

$$c_{i,\tau} \leftarrow x_{i,\tau} + k_{i,\tau} + b_{i,\tau}$$

where $k_{i,\tau}$ is computed based on the masking construction discussed in Chapter 3 (Equation 3.1). This new mask $b_{i,\tau}$ is generated from a randomly generated seed which is also shared using Shamir's Secret Sharing with all other users. In **SA**.**Agg**, the aggregator can ask to either reconstruct the blinding mask $b_{i,\tau}$ or the Diffie-Hellman private key of a user. Therefore, the aggregator will not be able to reveal individual inputs. So, it first collects t shares of the seed of each mask $b_{i,\tau}$ of every online user U_i and reconstructs it. Then, it gets t shares of the Diffie-Hellman's secret key of the dropped users and thus reconstructs the missing masks. Consider $\mathcal{X}$ and $\mathcal{Y}$ as the set of remaining and dropped users, respectively, the aggregation operation is described as follows (all the operations are performed mod $R = nR_i$ where $\mathbb{Z}_{R_i}$ is the range for the input values of each user):

$$\begin{aligned}
&\sum_{\mathcal{U}_i \in \mathcal{X}} c_{i,\tau} - \sum_{\mathcal{U}_i \in \mathcal{X}} b_{i,\tau} + \sum_{\mathcal{U}_j \in \mathcal{Y}} k_{i,\tau} \\
&= \sum_{\mathcal{U}_i \in \mathcal{X}} x_{i,\tau} + \sum_{\mathcal{U}_i \in \mathcal{X}} k_{i,\tau} + \sum_{\mathcal{U}_i \in \mathcal{X}} b_{i,\tau} - \sum_{\mathcal{U}_i \in \mathcal{X}} b_{i,\tau} + \sum_{\mathcal{U}_j \in \mathcal{Y}} k_{i,\tau} \\
&= \sum_{\mathcal{U}_i \in \mathcal{X}} x_{i,\tau} + \cancelto{0}{\sum_{\mathcal{U}_i \in \mathcal{X} \cup \mathcal{Y}} k_{i,\tau}} \\
&= \sum_{1}^{n} x_{i,\tau}
\end{aligned}$$

6.3.2 SecAgg$^+$

Bell et al. propose a new technique [BBG+20] to increase the scalability of secure aggregation protocols. In their solution, the clients do not need to share secret keys with all other clients to achieve robustness against dropouts. The authors propose to build partially connected (as opposed to fully connected) graphs where secure aggregation runs between the connected graph nodes (representing clients) only. More specifically, each client sends and receives secret shares of its keys to/from its neighbor nodes (according to the graph). Since the graph is connected, the correctness of mask cancellation and the privacy of individual values are both guaranteed under certain thresholds. The authors further propose a method to build these graphs to achieve a good tradeoff between security

and efficiency. They apply this technique to SecAgg and show that their technique reduces the complexity in a logarithmic scale with respect to the number of users.

6.3.3 TurboAgg

So et al. propose TurboAgg [SGA21b] which uses additive secret sharing instead of Shamir's secret sharing. To achieve robustness against dropouts, Lagrange coding [YRSA18] is used. The main drawback of the scheme is that it divides the clients into groups and each client in a group needs to communicate with every other client in the next group. Therefore, the communication cost of TurboAgg is relatively high. Additionally, the protocol can fail to recover the aggregate value when some clients drop with a non-negligible probability.

6.3.4 FastSecAgg

Kadhe et al. propose FastSecAgg [KRKR20] which uses FastShare instead of Shamir's secret sharing. FastShare is a new cryptographic primitive proposed by the same authors. It relies on Fast Fourier Transfer (FFT) and it decreases the computation cost of sharing and reconstructing a secret. Using FastShare, the authors reduce the computational complexity of the fault-tolerant secure aggregation. However, this comes at a cost of lower security guarantees. Indeed, FastSecAgg can reconstruct the aggregate (with a good probability) if the failed clients are arbitrarily chosen. Therefore, an adversary may prevent the recovery of the aggregate by cherry-picking a few clients and isolating them.

6.4 FTSA - Overview

In this section, we present our fault-tolerant secure aggregation protocol (FTSA) by giving an overview of the main idea.

Joye-Libert (JL) scheme [JL13] is an AHE-based SA scheme (see the construction of the scheme in Section 3.4.2.2). The scheme allows n users to encrypt their private input with a unique long-term pre-distributed key. Its main interesting property is that it is homomorphic with respect to both the messages and the encryption keys. An aggregator holding the sum of the user keys can decrypt the aggregate of the messages. Since this secure aggregation scheme supports long-term keys, it features a significant advantage in terms of computation and communication cost as it allows the use of the same keys for all FL rounds. However, JL introduces the following challenges when used for federated learning:

Client failures It is common in federated learning that some clients drop in several FL rounds. When one or more clients do not provide a ciphertext for FL round τ, the aggregation fails. This is because the aggregation key used to aggregate the ciphertext is equal to the sum of the user keys. If a client fails, its encryption key will not be involved in the aggregation. We deal with this problem by designing a threshold version of JL to recover from dropouts.

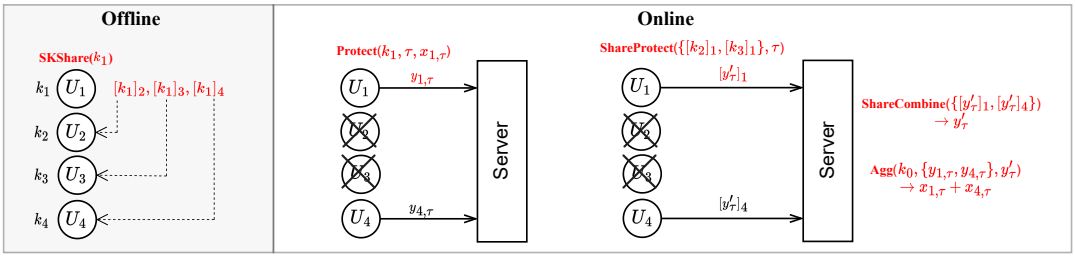


Figure 6.1: Demonstration of an execution of TJL scheme with four users ($n = 4$) and a reconstruction threshold $t = 2$.

Our Idea (Threshold-variant of JL scheme): We design a threshold JL scheme, whereby clients can secretly share their individual keys with each other so that when one or more clients fail, t out of n clients that are still online help provide a protected zero-value that is computed with the failed clients' individual keys. By computing the protected zero-value of the missing clients, the server can correctly compute the aggregated result.

Larger inputs In federated learning, the inputs are the parameters of the client's model and are of vector type. JL is originally designed to encrypt single integers. We extend JL to support vector inputs.
Our Idea (JL scheme with vector inputs): We propose to encode the vector elements in a single integer. The vector sum is decoded after aggregation.

No trusted key dealer JL requires a key dealer to distribute some keys to the users and the aggregator. However, a trusted key dealer may not be feasible in federated learning applications. Therefore, we propose a decentralised key setup phase to distribute the keys.
Our Idea (JL scheme with decentralized key setup): In order not to rely on a trusted key dealer, we propose a distributed key generation mechanism. We mainly use secure multi-party computation such that each of the n users and the aggregator get a random additive share of zero. Each user will use its share as a secret key so that the sum of the keys with the aggregator key equals zero.

6.5 Threshold Joye-Libert Scheme

We describe a threshold version of the Joye-Libert secure aggregation scheme (see Section 3.4.2.2 for the original scheme). The design of this scheme mainly transplants the design of the threshold version of the Paillier encryption scheme [DJN10] into this context. This extended solution mainly helps the aggregator recover failed users' inputs (which consists of the protection of the zero-value under each failed user's individual key) and hence compute the final aggregate value.

The goal is to distribute user key k_i to the n users such that any subset of at least t (online) users can produce a ciphertext on behalf of user U_i while less than t users learn

nothing. To secret share the keys, we use integer secret sharing `ISS` (see Section 2.3.1).

Let $\mathcal{U} = \{U_1, ..., U_n\}$ be the set of all users and $\mathcal{U}_{\texttt{on}} \subset \mathcal{U}$ be the set of online users, the threshold-variant Joye-Libert secure aggregation scheme, denoted as `TJL`, consists of the following PPT algorithms:

- $\texttt{TJL.Setup}(\lambda, \sigma) \rightarrow (pp, k_0, \{k_i\}_{\forall i \in [n]})$: Given some security parameter λ, this algorithm calls the original $\texttt{JL.Setup}(\lambda)$ and outputs the server key, one secret key per user and the public parameters $pp = (N, H^{\mathbb{Z}^*_{N^2}}, \mathbb{I})$ where $\mathbb{I} = [-2^l, 2^l]$ and l corresponds to the bit-size of the modulus N. Additionally, it sets the security parameter of the `ISS` scheme to σ.
- $\texttt{TJL.SKShare}(pp, k_i, t, n) \rightarrow \{(i, \langle k_i \rangle_j)\}_{\forall j \in [n]}$: On input of user U_i's secret key, this algorithm calls $\texttt{ISS.Share}(k_i, , t, n, \mathbb{I})$ (see Section 2.3.1). It outputs n shares of the key where each share is intended for different user $U_j \in \mathcal{U}$.
- $\texttt{TJL.ShareProtect}(pp, \{\langle k_j \rangle_i\}_{\forall j | U_j \in \mathcal{U} \setminus \mathcal{U}_{\texttt{on}}}, \tau) \rightarrow \langle y'_\tau \rangle_i$: This algorithm protects a zero-value with U_i's shares of all the secret keys corresponding to the failed users. It calls $\texttt{JL.Protect}(pp, \sum_{j \in \mathcal{U} \setminus \mathcal{U}_{\texttt{on}}} \langle k_j \rangle_i, \tau, 0)$ and outputs $\langle y'_\tau \rangle_i$
- $\texttt{TJL.ShareCombine}(pp, \{(i, \langle y'_\tau \rangle_i)\}_{\forall U_i \in \mathcal{U}^\tau_{\texttt{shares}}}) \rightarrow y'_\tau$: This algorithm computes the Lagrange interpolation on the exponent (the ν_i coefficients are defined in `ISS.Recon` in Section 2.3.1):

$$y'_\tau = \prod_{\forall U_i \in \mathcal{U}_{\texttt{shares}}} (\langle y'_\tau \rangle_i)^{\nu_i}$$

- $\texttt{TJL.Protect}(pp, k_i, \tau, x_{i,\tau}) \rightarrow y_{i,\tau}$: This algorithm calls $\texttt{JL.Protect}(pp, k_i, \tau, x_{i,\tau})$ and hence outputs cipher $y_{i,\tau}$.
- $\texttt{TJL.Agg}(pp, k_0, \tau, \{y_{i,\tau}\}_{U_i \in \mathcal{U}_{\texttt{on}}}, y'_\tau) \rightarrow X_\tau$: On input public parameters pp, the aggregation key k_0, the ciphertexts of online users, and the ciphertext of the zero-value corresponding to the failed users for timestamp τ, this algorithm computes:

$$y_\tau = (\prod_{U_i \in \mathcal{U}_{\texttt{on}}} y_{i,\tau})^{\Delta^2} \cdot y'_\tau \mod N^2 \tag{6.1}$$

To decrypt the final result, the algorithm proceeds as follows:

$$V_\tau = H(\tau)^{\Delta^2 k_0} \cdot y_\tau \qquad X_\tau = \frac{V_\tau - 1}{N\Delta^2} \mod N \tag{6.2}$$

Definition 6.5.1: Correctness

Given the set of users $\mathcal{U}$ and the set of online user $\mathcal{U}_{\texttt{on}} \subset \mathcal{U}$, the correctness of `TJL` scheme is defined as follows:

$$\forall \lambda, \forall \sigma, \forall t, \forall \tau, \forall x_{i,\tau}, \forall \tau, \forall \mathcal{U}_{\texttt{shares}} \text{ s.t. } (t \leq n) \wedge (\mathcal{U}_{\texttt{shares}} \subset \mathcal{U}_{\texttt{on}}) \wedge (|\mathcal{U}_{\texttt{shares}}| \geq t),$$

$$\Pr\left[\begin{array}{l|l} X_\tau \neq \sum\limits_{\forall U_i \in \mathcal{U}_{\text{on}}} x_{i,\tau} & \begin{array}{l} \texttt{TJL.Setup}(\lambda,\sigma)\to(pp,k_0,\{k_i\}_{\forall i\in[n]}),\\ \left\{\texttt{TJL.SKShare}(pp,k_i,t,n)\to\{(i,\langle k_i\rangle_j)\}_{\forall j\in[n]}\right\}_{\forall i\in[n]},\\ \left\{\texttt{TJL.ShareProtect}(pp,\{\langle k_j\rangle_i\}_{\forall U_j\in\mathcal{U}\setminus\mathcal{U}_{\text{on}}},\tau)\to\langle y'_\tau\rangle_i\right\}_{\forall U_i\in\mathcal{U}^\tau_{\texttt{shares}}},\\ \left\{\texttt{TJL.Protect}(pp,k_i,\tau,x_{i,\tau})\to y_{i,\tau}\right\}_{\forall U_i\in\mathcal{U}_{\text{on}}},\\ \texttt{TJL.ShareCombine}(pp,\{(i,\langle y'_\tau\rangle_i)\}_{\forall U_i\in\mathcal{U}_{\texttt{shares}}})\to y'_\tau,\\ \texttt{TJL.Agg}(pp,k_0,\tau,\{y_{i,\tau}\}_{\forall U_i\in\mathcal{U}_{\text{on}}},y'_\tau)\to X_\tau \end{array} \end{array}\right] = 0$$

Theorem 6.5.1

The scheme TJL is correct.

Proof.

To prove the correctness of TJL, observe that the following equalities hold:

$$\begin{aligned}
1)\ y_{i,\tau} &= (1+x_{i,\tau}N)H^{\mathbb{Z}^*_{N^2}}(\tau)^{k_i} \mod N^2\\
2)\ \langle y'_\tau\rangle_i &= H^{\mathbb{Z}^*_{N^2}}(\tau)^{\sum_{U_j\in\mathcal{U}\setminus\mathcal{U}_{\text{on}}}\langle k_j\rangle_i} \mod N^2\\
3)\ y'_\tau &= \prod_{U_i\in\mathcal{U}_{\texttt{shares}}}(\langle y'_\tau\rangle_i)^{\nu_i}\\
&= H^{\mathbb{Z}^*_{N^2}}(\tau)^{\sum_{U_i\in\mathcal{U}_{\texttt{shares}}}\nu_i\sum_{U_j\in\mathcal{U}\setminus\mathcal{U}_{\text{on}}}\langle k_j\rangle_i}\\
&= H^{\mathbb{Z}^*_{N^2}}(\tau)^{\sum_{U_j\in\mathcal{U}\setminus\mathcal{U}_{\text{on}}}\sum_{U_i\in\mathcal{U}_{\texttt{shares}}}\nu_i\langle k_j\rangle_i}\\
&= H^{\mathbb{Z}^*_{N^2}}(\tau)^{\Delta^2\sum_{U_j\in\mathcal{U}\setminus\mathcal{U}_{\text{on}}}k_j}\\
4)\ y_\tau &= (\prod_{U_i\in\mathcal{U}_{\text{on}}}y_{i,\tau})^{\Delta^2}\cdot y'_\tau \mod N^2\\
&= (1+\Delta^2\sum_{\forall U_i\in\mathcal{U}_{\text{on}}}x_{i,\tau}N)H^{\mathbb{Z}^*_{N^2}}(\tau)^{\Delta^2\sum\limits_{\forall U_i\in\mathcal{U}_{\text{on}}}k_i}\cdot H^{\mathbb{Z}^*_{N^2}}(\tau)^{\Delta^2\sum\limits_{\forall U_i\in\mathcal{U}\setminus\mathcal{U}_{\text{on}}}k_i}\\
&= (1+\Delta^2\sum_{\forall U_i\in\mathcal{U}_{\text{on}}}x_{i,\tau}N)H^{\mathbb{Z}^*_{N^2}}(\tau)^{\Delta^2\sum\limits_{\forall U_i\in\mathcal{U}}k_i}\\
&= (1+\Delta^2\sum_{\forall U_i\in\mathcal{U}_{\text{on}}}x_{i,\tau}N)H^{\mathbb{Z}^*_{N^2}}(\tau)^{-\Delta^2k_0}\\
5)\ V_\tau &= H(\tau)^{\Delta^2k_0}\cdot y_\tau\\
&= (1+\Delta^2\sum_{\forall U_i\in\mathcal{U}_{\text{on}}}x_{i,\tau}N)
\end{aligned}$$

Theorem 6.5.2

This scheme provides *Aggregator Obliviousness* security under the DCR assumption in the random oracle model if the number of corrupted users is less than threshold t.

Proof.

The security of this scheme mainly relies on the security of the JL secure aggregation scheme which is proved secure under the DCR assumption (Theorem 3.4.2). Similar to the original scheme, we assume that each user encrypts at most one input per timestamp. Additionally, we assume that failed users do not provide any encrypted values for that timestamp and the non-corrupted users construct at most one encrypted value on behalf of the failed clients. By relying on the security of the secret sharing scheme over integers (which is proved to be statistically secure in Theorem 1 in [VAS19]), we can show that less than t users cannot construct an encrypted value. Therefore, *Aggregator Obliviousness* is guaranteed in the random oracle model under the DCR assumption if less than t users are corrupted.

6.6 FTSA - Complete Specifications

We now describe the newly designed secure and fault-tolerant aggregation protocol based on the proposed `TJL` scheme. The protocol consists of a setup phase and an online phase each of them defined with two communication rounds. The setup phase is performed once, while the online phase is repeated for each federated learning round. We describe the details of each protocol phase.

6.6.1 The Setup Phase

The setup phase consists of the registration of the clients and the distribution of the security keys. In `TJL`, a trusted key dealer is needed to generate the keys. To avoid the dependence on a key dealer, we propose a distributed method to setup the `TJL` user keys.

Distribution of `TJL` keys We recall that the aggregator's key k_0 allows the aggregator to recover the sum of the inputs from the set of users' ciphertexts (see Equation 6.1). The goal of this key is to protect the final aggregate result. In the case of FL applications, the aggregated ML model is considered public and thus does not need to be hidden. Therefore, we set the aggregator's key to a publicly known value (for example zero). Indeed, the security proof in [JL13] considers the case where an adversary controls the key of the aggregator. In such case, the scheme cannot protect the result of the aggregation but it still protects the individual inputs of the users which is sufficient for FL. To generate the user keys, every two clients U_i and U_j agree using the `KA` scheme on a shared mutual key $k_{i,j}$. Then, client U_i computes its protection key $k_i \leftarrow \sum_{U_j \in \mathcal{U}} (\delta_{i,j} \cdot k_{i,j})$ where $\delta_{i,j} = 1$ when $i > j$, and $\delta_{i,j} = -1$ when $i < j$. The correctness of the protocol is preserved since:

$$\sum_{\forall U_i \in \mathcal{U}} k_i = \sum_{\forall U_i \in \mathcal{U}} \Big(\sum_{\forall U_j \in \mathcal{U}} \delta_{i,j} \cdot k_{i,j} \Big) = 0 = -k_0$$

FTSA - Setup Phase

- **Setup - Registration:**
  Trusted Dealer:
  - Choose security parameters λ, σ and runs $\texttt{KA.Param}(\lambda) \rightarrow (\mathbb{G}, g, q) = pp^{\texttt{KA}}$ and $\texttt{TJL.Setup}(\lambda, \sigma) \rightarrow ((N, H^{\mathbb{Z}^*_{N^2}}, \mathbb{I}), \perp, \perp)$. It sets the public parameters $pp = (pp^{\texttt{KA}}, pp^{\texttt{TJL}}, G^{\mathbb{G}}, t, n, m, R, \mathbb{F})$ such that t is the secret sharing threshold, n is the number of clients, $\mathbb{Z}_R^m$ is the space from which inputs are sampled, $G^{\mathbb{G}}$ is the hash function for key-derivation (see Section 2.4.3), and $\mathbb{F}$ is the field for SSS scheme. It sends them to the server and to all the clients.

  User u (pp):
  - Receive the public parameters from the trusted dealer.
  - Generate key pairs $(c_i^{PK}, c_i^{SK}) \leftarrow \texttt{KA.gen}(pp^{\texttt{KA}}), (s_i^{PK}, s_i^{SK}) \leftarrow \texttt{KA.gen}(pp^{\texttt{KA}})$
  - Send $(c_i^{PK} \,||\, s_i^{PK})$ to the server (through the private authenticated channel) and move to next round.

  Server(pp):
  - Receives public parameters pp from the trusted dealer.
  - Collect all public keys from the users (denote with $\mathcal{U}$ the set of registered users). Abort if $|\mathcal{U}| < t$ otherwise move to the next round.
  - Broadcast to all users the list $\{U_i, (c_i^{PK}, s_i^{PK})\}_{\forall U_i \in \mathcal{U}}$
- **Setup - Key Setup:**
  User u:
  - Receive the public keys of all registered users $\mathcal{U}$
  - For each registered user $U_j \in \mathcal{U} \setminus \{U_i\}$, compute channel keys $c_{i,j} \leftarrow \texttt{KA.agree}(pp^{\texttt{KA}}, c_i^{SK}, c_j^{PK}, G^{\mathbb{G}})$.
  - For each registered user $U_j \in \mathcal{U} \setminus \{U_i\}$, compute $k_{i,j} \leftarrow \texttt{KA.agree}(pp^{\texttt{KA}}, s_i^{SK}, s_j^{PK}, G^{\mathbb{G}})$ (set $k_{i,i} = 0$). Then compute the TJL secret key $k_i \leftarrow \sum_{U_j \in \mathcal{U}} \delta_{i,j} \cdot k_{i,j}$ where $\delta_{i,j} = 1$ when $i > j$, and $\delta_{i,j} = -1$ when $i < j$.
  - Generate t-out-of-n shares of the TJL secret key: $\{(j, \langle k_i \rangle_j)\}_{\forall U_j \in \mathcal{U}} \leftarrow \texttt{TJL.SKShare}(pp^{\texttt{TJL}}, k_i, t, n)$.
  - For each registered user $U_j \in \mathcal{U} \setminus \{U_i\}$, encrypt its corresponding shares: $\epsilon_{i,j} \leftarrow \mathsf{Enc}_{c_{i,j}}(U_i \,||\, U_j \,||\, \langle k_i \rangle_j)$.
  - If any of the above operations fails, abort.
  - Send all the encrypted shares $\{\epsilon_{i,j}\}_{\forall U_j \in \mathcal{U}}$ to the server (each implicitly containing addressing information U_i, U_j as metadata).
  - Store all messages received and values generated in the setup phase, and move to the online phase.

  Server:
  - Collect from each user U_i its encrypted shares $\{(U_i, U_j, \epsilon_{i,j})\}_{\forall U_j \in \mathcal{U}}$.
  - Forward to each user $U_j \in \mathcal{U}$ its corresponding encrypted shares: $\{(U_i, U_j, \epsilon_{i,j})\}_{\forall U_i \in \mathcal{U}}$ and move to the online phase.

Figure 6.2: Detailed description of the setup phase of FTSA

Note that this distributed method allows the distribution of the JL user keys but it still relies on a trusted third party to generate the public parameters. There exist techniques to distribute the computation of the public RSA modulus N ([BF01, NS11, VAS19] in the passive setting and [CDK+22] in the active setting). In this work, we assume the existence of a trusted offline third party that distributes the public parameters.

Protocol steps During *Registration*, clients register by sending their public keys to the aggregator who further broadcasts them to all clients. Notice that each client generates two public keys, one used to create secret communication channels among clients and the other used to compute the TJL secret key. Later, in the *Key Setup* step, each client U_i computes mutual channel keys $c_{i,j}$ and its TJL key k_i. It also creates secret shares of the TJL keys using TJL.SKShare and sends them to the corresponding clients (via the server through the authenticated channels). The specifications of the setup phase are given in Figure 6.2.

6.6.2 The Online Phase

The online phase consists of two communication rounds. In the first communication round, (i.e., *Encryption* step), the clients protect their inputs and send them to the aggregator. In the second communication round (i.e., *Aggregation* step), the clients and the aggregator construct the ciphertext of the failed clients.

Blinding inputs to ensure privacy One problem with the TJL scheme is that it guarantees privacy under the assumption that only one ciphertext of each user is computed for each timestamp. However, this assumption may break even in the honest-but-curious model. Let us assume the case where a client sends its protected input with some delay. The delay may cause the aggregator to request the online clients to construct the protected zero-value of the assumed failed client. In this case, two ciphertexts for the same timestamp are collected breaking the security assumption of the JL scheme. To deal with this problem, the client masks its input before encrypting it. The goal of the mask is to protect any leakage in case two ciphertexts of the same period are obtained. To remove these masks from the aggregated value, each client secretly shares its mask with all other clients. If the client survives the federated learning round, the online clients help construct its blinding mask. Otherwise, the clients construct the ciphertext of the zero-value using the TJL scheme.

Input vector encoding The TJL scheme is originally designed to work with integers. In federated learning applications, FL clients send a vector of parameters instead of a single value. We, therefore, propose a dedicated encoding solution to encode a vector into a long integer. Each element of the initial vector V is firstly expanded by $\log_2(n)$ bits of 0's at the beginning of the element. Then the elements of the vector are packed to form a large integer ω. The number of elements that ω represents corresponds to $ptsize$ which denotes the plaintext size of TJL divided by the actual size of the extended element (i.e

$\lfloor \frac{ptsize}{\log_2(R)+\log_2(n)} \rfloor$,$R$ being the maximum possible value for vector elements). Note that for `TJL` scheme, the plaintext size is equal to half of the size of the user key ($ptsize = \frac{|k_i|}{2}$).

The decoding operation simply consists of unpacking ω into bitmaps of $\log_2(R) + \log_2(n)$.

To execute `TJL.Protect` and `TJL.ShareProtect`, the user input is first encoded. In the case where the user input vectors are too large to be encoded in a single integer of size $ptsize$, the vectors are split into multiple vectors of length $\lfloor \frac{ptsize}{\log_2(R)+\log_2(n)} \rfloor$ and encoded separately. We use a counter to generate a unique timestamp for each encoded part.

Protocol steps In the *Encryption* step, the client generates random seed b_i which is further extended to blinding mask B_i using a `PRG`. The client first blinds its input with B_i then protects it with `TJL.Protect`. After that, the client secretly shares the seed b_i with other clients and sends its masked and protected input to the server.

In the *Aggregation* step, the client learns the list of failed clients and computes `TJL.ShareProtect` using the sum of their TJL keys' shares. Then, it sends to the server the blinding mask share of each online client and the share of the protected zero-value corresponding to all the failed clients. The server constructs the blinding masks and the protected zero-value using `TJL.ShareCombine`. Finally, it uses `TJL.Agg` to obtain the blinded sum which is unblinded by removing the clients' masks. The detailed specification of this phase is provided in Figure 6.3.

6.6.3 Deployment of FTSA on Semi-Connected Graphs (FTSA$^+$)

In the current description of the protocol, we assumed that all clients secretly share their keys with each other. This gives the maximum security guarantee. However, as shown in [BBG+20], the client doesn't need to share its key with all other clients (see Section 6.3). Indeed it is sufficient to build a connected graph where each node (representing a client) is connected to a subset of all clients and only shares its key with this subset. As we presented in Section 6.3, this technique is used to improve the scalability of SecAgg [BIK+17]. We observe that the same method can also be applied to FTSA. We refer to our protocol as FTSA$^+$ when deployed on such graphs. Given a connected graph as described in [BBG+20], FTSA$^+$, involves the same steps as FTSA, except that the sharing of the key is done only with the neighbor clients. By using this optimization, the complexity of our protocol in terms of the number of users is reduced into a logarithmic factor.

6.7 Security Analysis

In this section, we evaluate the security of our secure and fault-tolerant aggregation protocol (FTSA) and prove that it ensures *Aggregator Obliviousness* (see Definition 3.2.1) in both the honest-but-curious and malicious model settings, given a dedicated threshold t of honest clients.

FTSA - Online Phase

- **Online - Encryption (step τ):**
  User u:
  - Sample a random element $b_{i,\tau} \xleftarrow{R} \mathbb{F}$ (to be used as a seed for a `PRG`).
  - Extend $b_{i,\tau}$ using the `PRG`: $B_{i,\tau} \leftarrow \texttt{PRG}(b_{i,\tau})$.
  - Protect the private input $X_{i,\tau} \in \mathbb{Z}_R^m$ (after masking it with $B_{i,\tau}$) using `TJL` scheme: $Y_{i,\tau} \leftarrow \texttt{TJL.Protect}(pp, k_i, \tau, X_{i,\tau} + B_{i,\tau})$.
  - Generate t-out-of-n shares of $b_{i,\tau}$ using the `SSS` scheme: $\{(j, \langle b_{i,\tau}\rangle_j)\}_{\forall U_j \in \mathcal{U}} \leftarrow \texttt{SSS.Share}(b_{i,\tau}, t, n, \mathbb{F})$.
  - For each registered user $U_j \in \mathcal{U} \setminus \{U_i\}$, encrypt its corresponding shares $e_{(i,j),\tau} \leftarrow \mathsf{Enc}_{c_{i,j}}(U_i \,||\, U_j \,||\, \langle b_{i,\tau}\rangle_j)$
  - If any of the above operations fails, abort.
  - Send all the encrypted shares $\{e_{(i,j),\tau}\}_{\forall U_j \in \mathcal{U}}$ (with addressing information U_i, U_j as metadata) and the protected input $Y_{i,\tau}$ to the server .

  Server:
  - Collect from each user u its encrypted shares $\{(U_i, U_j, e_{(i,j),\tau})\}_{\forall U_j \in \mathcal{U}}$ and its protected input $Y_{i,\tau}$ (or time out).
  - Denote with $\mathcal{U}_{\text{on}}^{\tau} \subset \mathcal{U}$ the set of online users. Abort if $|\mathcal{U}_{\text{on}}^{\tau}| < t$.
  - Forward to each user $U_j \in \mathcal{U}_{\text{on}}^{\tau}$ its corresponding encrypted shares: $\{(i, j, e_{(i,j),\tau})\}_{\forall U_i \in \mathcal{U}_{\text{on}}^{\tau}}$.
- **Online - Aggregation (step τ):**
  User u:
  - Receive the encrypted shares and deduce the list of online users $\mathcal{U}_{\text{on}}^{\tau}$ from the received shares. Verify that $\mathcal{U}_{\text{on}}^{\tau} \subset \mathcal{U}$ and $|\mathcal{U}_{\text{on}}^{\tau}| >= t$.
  - Decrypt all the encrypted secret shares: $U_j' \,||\, U_i' \,||\, \langle b_{j,\tau}\rangle_i \leftarrow \mathsf{Dec}_{c_{i,j}}(e_{(j,i),\tau})$. Assert that $U_i = U_i' \wedge U_j = U_j'$
  - Compute the share of the zero-value corresponding to all failed users: $\langle Y_\tau'\rangle_i \leftarrow \texttt{TJL.ShareProtect}(pp^{\texttt{TJL}}, \{\langle k_j\rangle_i\}_{\forall U_j \in \mathcal{U} \setminus \mathcal{U}_{\text{on}}^{\tau}}, \tau)$.
  - Abort if any operation failed.
  - Send the secret shares of the blinding mask seeds $\{\langle b_{j,\tau}\rangle_i\}_{\forall U_j \in \mathcal{U}_{\text{on}}^{\tau}}$ and of the share of the protected zero-value $\langle Y_\tau'\rangle_i$ to the server.

  Server:
  - Collect shares from at least t honest users. Denote with $\mathcal{U}_{\texttt{shares}}^{\tau} \subset \mathcal{U}_{\text{on}}^{\tau}$ the set of users. Abort if $|\mathcal{U}_{\texttt{shares}}^{\tau}| < t$.
  - Construct the blinding mask seed of all users $b_{i,\tau}$ $\forall U_i \in \mathcal{U}_{\text{on}}^{\tau}$: $b_{i,\tau} \leftarrow \texttt{SSS.Recon}(\{\langle b_{i,\tau}\rangle_j\}_{\forall U_j \in \mathcal{U}_{\texttt{shares}}^{\tau}}, \mathbb{F})$
  - Recompute the blinding mask: $B_{i,\tau} \leftarrow PRG(b_{i,\tau})$
  - Construct the protected zero-value corresponding to all failed users: $Y_\tau' \leftarrow \texttt{TJL.ShareCombine}(pp, \{\langle Y_\tau'\rangle_j\}_{\forall U_j \in \mathcal{U}_{\texttt{shares}}^{\tau}})$
  - Aggregate all the protected inputs of the online clients and the protected zero-value: $C_\tau \leftarrow \texttt{TJL.Agg}(pp, 0, \tau, \{Y_{i,\tau}\}_{\forall U_i \in \mathcal{U}_{\text{on}}^{\tau}}, Y_\tau')$
  - Remove the blinding masks $C_\tau - \sum_{\forall U_i \in \mathcal{U}_{\text{on}}^{\tau}} B_{i,\tau} = \sum_{\forall U_i \in \mathcal{U}_{\text{on}}^{\tau}} X_{i,\tau}$

Figure 6.3: Detailed description of the online phase of FTSA

6.7.1 Security in the honest-but-curious model

As explained in Section 3.2, in the honest-but-curious model, the adversary, who corrupts the aggregator and a subset of the clients, correctly follows the protocol but colludes with up to $n - t$ clients. Let $\mathcal{U}_{\texttt{corr}}$ be the set of corrupted clients and $\mathcal{C} = \mathcal{U}_{\texttt{corr}} \cup A$ (A represents the aggregator).

Theorem 6.7.1: Security in the honest-but-curious model

In the honest-but-curious model, the protocol FTSA achieves *Aggregator Obliviousness* if the number of corrupted clients is less than threshold t ($|\mathcal{U}_{\texttt{corr}}| < t$).

Proof.

To prove this argument, we show that the view of $\mathcal{C}$ is computationally indistinguishable from a simulated view where all inputs of the honest clients are replaced with random values (given that the sum of the random inputs is equal to the sum in the real view). For the offline phase, the proof is trivial as the offline phase is considered as a distribution of a secret sharing a zero value between n parties using Diffie-Hellman. Thus, we focus on proving security in the online phase using a hybrid argument that relies on the security of the underlying cryptographic primitives.

- $\boldsymbol{\mathcal{H}_0}$: This hybrid view is the same as the real view.
- $\boldsymbol{\mathcal{H}_1}$: In this hybrid view, we replace all channel keys established by `KA.agree` between honest clients with uniformly random keys. The Decisional Diffie-Hellman assumption guarantees that this hybrid view is indistinguishable from the previous one.
- $\boldsymbol{\mathcal{H}_2}$: In this hybrid view, we replace all ciphertexts generated from Enc between honest clients with encryptions of zeros (padded to the right length). The IND-CPA security of the encryption scheme (see Definition 2.3.5) guarantees that this hybrid view is indistinguishable from the previous one.
- $\boldsymbol{\mathcal{H}_3}$: In this hybrid view, we replace all shares of b_i sent in the online phase from honest clients $U_i \in \mathcal{U} \setminus \mathcal{U}^{\tau}_{\texttt{on}} \setminus \mathcal{U}_{\texttt{corr}}$ to corrupted ones with a random share of zero. Notice that the adversary does not receive additional shares of b_i in the construction phase since the honest clients do not provide shares of b_i for $U_i \in \mathcal{U} \setminus \mathcal{U}^{\tau}_{\texttt{on}}$. Hence, if $|\mathcal{U}_{\texttt{corr}}| < t$, this hybrid view is indistinguishable from the previous one due to the security of the `ISS` scheme (see Definition 2.3.2).
- $\boldsymbol{\mathcal{H}_4}$: In this hybrid view, we replace all honest clients' ($\mathcal{U} \setminus \mathcal{U}_{\texttt{corr}}$) inputs with uniformly random values such that the sum of the online clients' random inputs is equal to the aggregate. The simulator masks these random values and then protects them with `TJL.Protect` and gives the resulting ciphertexts to the adversary. Notice that we allow the adversary to receive `TJL.Protect` ciphertexts from offline

clients as this may happen in benign scenarios where the aggregator receives the delayed ciphertext after it had considered the user offline.

- For the online honest clients ($U_i \in \mathcal{U}^{\tau}_{\texttt{on}} \setminus \mathcal{U}_{\texttt{corr}}$) the adversary does not receive the shares of the zero-protected value from the honest clients. Thus, the adversary cannot construct the zero-protected value of the user U_i if $|\mathcal{U}_{\texttt{corr}}| < t$. Hence, the adversary only sees one ciphertext of `TJL.Protect` of the user U_i. The security of TJL states that the ciphertext of two different values cannot be distinguished under the DCR assumption if only one ciphertext per timestamp is provided (see Theorem 6.5.2). Therefore, the ciphertext of the uniformly random input in this hybrid is indistinguishable from the ciphertext of the real input from the real view.
- For the offline honest clients ($U_i \in \mathcal{U} \setminus \mathcal{U}^{\tau}_{\texttt{on}} \setminus \mathcal{U}_{\texttt{corr}}$) the adversary does not receive the shares of the mask b_i from honest clients. Thus, the adversary cannot construct the mask b_i of the user U_i if $|\mathcal{U}_{\texttt{corr}}| < t$. Hence, the masked random input in this hybrid view cannot be distinguished from the masked real input in the real view.

Since the adversary cannot distinguish the messages in the hybrid view from that in the real view for both online and offline clients, we can deduce that this hybrid view is indistinguishable from the previous one if $|\mathcal{U}_{\texttt{corr}}| < t$.

We showed that we can define a simulator that can simulate the view of parties in $\mathcal{C}$, where for any adversary in polynomial time the output of the simulation is indistinguishable from the real view of $\mathcal{C}$ if $|\mathcal{U}_{\texttt{corr}}| < t$. Thus, the aggregator learns nothing more than the sum of the inputs of the online honest clients. This proves *Aggregator Obliviousness*.

Corollary 6.7.1

In the honest-but-curious model, security is guaranteed if the minimum number of honest clients t should be strictly larger than half of the number of clients in the protocol ($t > \frac{n}{2}$) and the protocol can recover from up to $\frac{n}{2} - 1$ client failures.

Proof.

From Theorem 6.7.1, $|\mathcal{U}_{\texttt{corr}}| < t$. We also have the number of honest users set to the threshold t (i.e, $|\mathcal{U}_{\texttt{corr}}| = n - t$). Therefore, $t > \frac{n}{2}$.

6.7.2 Security in the malicious model

As explained in Section 3.2, in the malicious model, the adversary, who corrupts the aggregator and a subset of users, can additionally manipulate the messages of the corrupted parties.

Theorem 6.7.2: Security in the malicious model

In the malicious model, the protocol FTSA achieves *Aggregator Obliviousness* if the number of corrupted clients is $|\mathcal{U}_{\texttt{corr}}| < 2t - n$.

Proof.

The only messages that the aggregator distributes, are the encrypted shares. The aggregator cannot modify the values of these encrypted shares thanks to the underlying authenticated encryption Enc scheme (see Definition 2.3.6). Therefore, the aggregator's power in the protocol is limited to not forwarding some of the shares. This will make clients reach some false conclusions about the set of online clients in the protocol. Note, importantly, that the aggregator can give different views to different clients about which clients have failed. Therefore, the aggregator can convince a set of honest clients to send the protected zero-value Y'_τ corresponding to client U_i (i.e., $\mathcal{U} \setminus \mathcal{U}^\tau_{\texttt{on}} = \{U_i\}$) while asking another set of honest clients to send the shares of the blinding mask $b_{i,\tau}$. The aggregator should not obtain the protected zero-value Y'_τ corresponding to a client U_i and its blinding mask $b_{i,\tau}$ for the same FL round τ. If that happens, the aggregator can recover the masked input from Y'_τ and $Y_{i,\tau}$, then remove the mask using $b_{i,\tau}$.

Knowing that the aggregator may collude with $|\mathcal{U}_{\texttt{corr}}|$ corrupted clients, The adversary controlling $\mathcal{C}$ obtains $|\mathcal{U}_{\texttt{corr}}|$ shares of Y'_τ s.t. $\mathcal{U} \setminus \mathcal{U}^\tau_{\texttt{on}} = \{U_i\}$ and $|\mathcal{U}_{\texttt{corr}}|$ shares of $b_{i,\tau}$. Additionally, for the remaining $\mathcal{U} \setminus \mathcal{U}_{\texttt{corr}}$ honest clients, the adversary's best strategy is to set U_i as online for half of them and set it as offline for the other half. It thus receives shares of Y'_τ of U_i from $\frac{n-|\mathcal{U}_{\texttt{corr}}|}{2}$ honest clients and shares of $b_{i,\tau}$ from the other $\frac{n-|\mathcal{U}_{\texttt{corr}}|}{2}$ honest clients. Hence, in total, the adversary can learn a maximum number of $|\mathcal{U}_{\texttt{corr}}| + \frac{n-|\mathcal{U}_{\texttt{corr}}|}{2} = \frac{n+|\mathcal{U}_{\texttt{corr}}|}{2}$ shares of Y'_τ and $b_{i,\tau}$ of the same client.

Finally, we can use the same hybrid argument used in the proof of security in the honest-but-curious model except that in the active model the adversary chooses to receive the share $\langle Y'_\tau \rangle_j$ or the share $\langle b_{i,\tau} \rangle_j$ from each honest client U_j (instead of being fixed according to $\mathcal{U}^\tau_{\texttt{on}}$). Hence, the indistinguishablity can be ensured as long as the maximum number of shares of both values obtained by the adversary is less than the reconstruction threshold ($\frac{n+|\mathcal{U}_{\texttt{corr}}|}{2} < t$). Therefore, we can prove *Aggregator Obliviousness* if the number of corrupted clients is $|\mathcal{U}_{\texttt{corr}}| < 2t - n$.

Corollary 6.7.2

In the active model, security is guaranteed if the minimum number of honest clients t should be strictly larger than two third of the number of clients in the protocol ($t > \frac{2n}{3}$) and the protocol can recover from up to $\frac{n}{3} - 1$ client failures.

Proof.

From Theorem 6.7.2, $|\mathcal{U}_{\texttt{corr}}| < 2t - n$. We also have the number of honest users set to the threshold t (i.e, $|\mathcal{U}_{\texttt{corr}}| = n - t$). Therefore, the reconstruction threshold is $t > \frac{2n}{3}$.

6.8 Performance Analysis

In this section, we analyze the performance of FTSA in terms of computation, communication, and storage costs at both the client and the server and compare it with the state-of-the-art. We first perform the complexity analysis with respect to the number of clients n and the dimension of the client's input m. Then, we implement the protocol and conduct an experimental evaluation of it. For completeness, we analyze both the offline and online phases. We only compare the online phases of different protocols since the offline setup phase is a one-time process.

Table 6.1: Complexity analysis of the online phase of FTSA and the state-of-the-art protocols. The values represent the order of complexity. n is the number of clients (order of hundreds) and m is the dimension of the client's input (order of hundreds of thousands).

	Communication		**Computation**	
	Client	Server	Client	Server
SecAgg [BIK+17]	$m+n$	$nm+n^2$	$nm+n^2$	n^2m
SecAgg+ [BBG+20]	$\boldsymbol{m+\log(n)}$	$\boldsymbol{nm+n\log(n)}$	$\log(n)m+\log^2(n)$	$n\log(n)m+n\log^2(n)$
TurboAgg [SGA21b]	$\log(n)m$	$n\log(n)m$	$\log(n)m$	$\boldsymbol{nm}$
FastSecAgg [KRKR20]	$m+n$	$nm+\log^2(n)$	$\log(n)m$	$\boldsymbol{nm}$
FTSA	$m+n$	$nm+n^2$	$m+n^2$	$nm+n^2$
FTSA+	$\boldsymbol{m+\log(n)}$	$\boldsymbol{nm+n\log(n)}$	$\boldsymbol{m+\log^2(n)}$	$nm+n\log^2(n)$

6.8.1 Scalability Analysis of The Offline Phase

In terms of communication cost, the offline phase consists of each client receiving n public keys and sending n shares of its secret keys. Thus, the complexity is $O(n)$ for the client and $O(n^2)$ for the server. In terms of computation cost, the server only relays the messages between the clients so it does not perform cryptographic operations. However, the client's cost is dominated with creating t shares of its secret keys which is $O(n^2)$. Finally, the storage requirement of the client depends on the number of shares which is $O(n)$ while it is $O(n^2)$ for the server since it has to collect and forward the encrypted shares of all clients.

6.8.2 Scalability Analysis of The Online Phase at the Client

Communication The communication cost consists of:

During the *Encryption* step:

1. Sending $O(n)$ shares of $b_{i,\tau}$ and receiving $O(n)$ shares.
2. Sending the encrypted client input which is $O(m)$.

During the *Aggregation* step:

3. Sending $O(n)$ shares of $b_{i,\tau}$.
4. If at least one client failed, sending the share of the protected zero-value $\langle Y'_\tau \rangle_i$ which is $O(m)$.

During the total, the complexity is $O(n+m)$ which is the **same** as in [BIK+17].

Computation The computation cost consists of:
During the *Encryption* step:

1. Generating t out of n shares of $b_{i,\tau}$ which is $O(n^2)$.
2. Encrypting the client's message $X_{i,\tau}$ which is $O(m)$.

During the *Aggregation* step:

3. Encrypting the zero-value using the secret shares which is $O(m)$.

Therefore, the total complexity is $O(n^2+m)$ which is **lower** than in [BIK+17] ($O(n^2+nm)$). The higher complexity in [BIK+17] is due to the one-to-one key agreement on the shared masking seeds then extending each of them to the dimension of the client's input which adds an extra $O(nm)$.

Storage The client must store the keys and shares of all clients as well as the data vector which results in a storage overhead of $O(n+m)$ which is the **same** as the one in [BIK+17].

6.8.3 Scalability Analysis of The Online Phase at the Server

Communication The server's communication cost is n times the client's communication cost. Thus, a complexity of $O(n^2+nm)$ which is the **same** as the one in [BIK+17].

Computation The computation cost consists of:
During the *Encryption* step, the server's computation cost is not significant since the server only forwards messages.
During the *Aggregation* step:

1. Computing the product of the received ciphertexts which corresponds to $O(nm)$.
2. Reconstructing t out of n shares of $b_{i,\tau}$ for each online client U_i which is $O(n^2)$.
3. Extending the seed $b_{i,\tau}$ to the dimension of the client's input (using **PRG**) which is $O(nm)$.
4. If at least one client failed, constructing the protected zero-value Y'_τ from its t shares which is $O(nm)$.

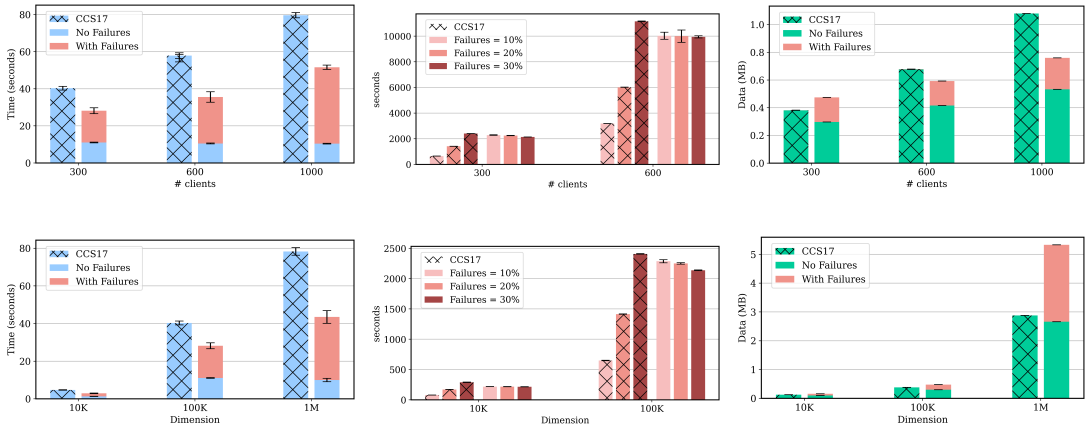
5. Aggregating the ciphertexts and unmasking the result which is $O(nm)$.

The total computation cost equals to $O(n^2+nm)$ which is **lower** than the one in [BIK+17] ($O(n^2m)$).

Note that the reconstruction of t out of n shares normally costs $O(n^2)$ since it consists of computing the Lagrange coefficient $O(n^2)$ and then applying the Lagrange formula $O(n)$. Thus, the computation of t out of n shares for n clients should cost $O(n^3)$. However, we optimize the reconstruction by computing the Lagrange coefficients only one time per FL round and use them for all the reconstructions which result in $O(n^2+n) \equiv O(n^2)$.

Storage $O(n^2+nm)$. The server storage consists of: t shares of $b_{i,\tau}$ for each client $b_{i,\tau}$ which is $O(n^2)$ and t shares of Y'_τ which is $O(nm)$.

6.8.4 Experimental Evaluation



(a) Client Runtime (b) Server Runtime (c) Bandwidth

Figure 6.4: The wall-clock running time (a,b) and the total data transfer sent and received (c). The measurements are performed using a single-threaded python implementation of our solution and the solution in [BIK+17] (only the online phase time is shown). When varying the number of clients, we fix the input dimension to $m = 100$K and when varying the dimension we fix the number of clients to $n = 300$. Bars represent the average value based on 10 runs and the error margins represent the standard deviation.

We have implemented a prototype of the TJL scheme and our fault-tolerant secure aggregation protocol. Additionally, we have also implemented a prototype of the protocol presented in [BIK+17]. Our implementation is done using Python programing language and is available on GitHub[1]. We first benchmark our new proposed TJL secure aggregation scheme. Then, we run several experiments with our secure aggregation protocol and the one in [BIK+17] while varying the number of clients $n = \{100, 300, 600, 1000\}$. We also use different dimensions for clients' inputs $m = \{10^3, 10^4, 0^5, 10^6\}$ and different client

[1] https://github.com/MohamadMansouri/fault-tolerant-secure-agg

failure rates $f = \{0\%, 10\%, 20\%, 30\%\}$. In each experiment, we run all the clients' and the server's processes and measure their execution time. We also measure the size of the data transmitted in the network. The measurements are performed using a single-threaded process on a machine with an Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz processor and 32 GB of RAM.

6.8.4.1 Implementation Details

We use the same implementation and parameters for the building blocks of our protocol and the protocol proposed in [BIK+17].

- *Pseudo-Random Generator (PRG)*: We use AES encryption [DBN+01] in the counter mode with 128 bits key size. Thus, the blinding mask seed (b_i) is 128-bit long.
- *Key Agreement (KA)*: We use Elliptic-Curve Diffie-Hellman over the NIST P-256 curve. For the hash function, we use SHA256.
- *Secret Sharing (SS)*: We use the finite fields $\mathbb{F}_p$ (integers modulo p where p is prime) in the implementation of Shamir's secret sharing. To share the seed of the blinding mask (b_i) (128 bits) we set $p = 2^{129} - 1365$. To share the DH secret key (256 bits) we set $p = 2^{257} - 2233$.
- *Authenticated Encryption (*Enc*)*: We use AES-GCM [Dwo07] with a key size of 256 bits.
- *Threshold Joye-Libert Scheme (TJL)*: We use 1024 bits for the public parameter N. Thus, the user keys are of size 2048 bits. For the hash function $H : \mathbb{Z} \to \mathbb{Z}^*_{N^2}$, we implement it as a Full-Domain Hash using a chain of eight SHA256 hashes. Additionally, for the secret sharing over the integers scheme `ISS` used by `TJL.SKShare`, we set the security parameter σ to 128 bits.

6.8.4.2 Benchmarks for `JL` and `TJL` schemes

We show in Table 6.2 the execution time of all the algorithms of `JL` and `TJL` schemes.

6.8.4.3 Benchmarks for FTSA and SecAgg

Running time for clients We plot the wall-clock running time for a client in both protocols (i.e., our protocol and the one in [BIK+17]) in Figure 6.4a. Our protocol shows better running time in most of the scenarios. Additionally, our protocol scales better with respect to the increasing number of clients (i.e., our solution is $\times 1.5$ faster with 100 clients, $\times 5.5$ faster with 600 clients, and $\times 8$ faster with 1000 clients). This confirms the study in Section 6.8.2. We also show the setup time for our protocol in Table 6.4. The results show that the offline time becomes negligible after running a sufficient number of FL rounds.

Running time for the server We plot the wall-clock running time for the server in both protocols (i.e., our protocol and the one in [BIK+17]) in Figure 6.4b. We show the

Table 6.2: Benchmarks of JL and TJL schemes. The time measurements are in milliseconds such that x / y refers to x ms runtime for JL scheme and y ms runtime for TJL scheme. The current benchmarks are run with 30% user failures and 70% threshold. Gray distinguishes the algorithms that are executed only once for the setup.

		$n = 100$	$n = 300$	$n = 600$
TP	**Setup**	12 / 10	11 / 15	13 / 13
User	**SkShare**	- / 2	- / 23	- / 149
	ShareProtect	- / 7	- / 14	- / 26
	Protect	4 / 4	4 / 4	4 / 4
Agg	**ShareCombine**	- / 88	- / 946	- / 4220
	Agg	4 / 8	5 / 21	7 / 42

Table 6.3: Wall-clock running time per client in the online phase for one FL round with different % of client failures. The number of clients varies $n = \{100, 300, 600, 1000\}$ and the dimension is fixed to $m = 100K$.

# Clients	**% Failures**			
	0%	10%	20%	30%
100	9.8 sec	20.9 sec	20.8 sec	20.8 sec
300	10.9 sec	28.1 sec	27.9 sec	26.6 sec
600	10.5 sec	35.5 sec	35.6 sec	35.8 sec
1000	10.7 sec	51.6 sec	51.2 sec	52.1 sec

running time with different failure rates of clients (with 0% failure rate the runtime is negligible since there is no need for a recovery phase which is the most costly operation). In general, the cost is higher in our protocol since the reconstruction operation (i.e., Lagrange formula) is computed on the exponent. However, this overhead is constant in our protocol w.r.t. the number of failed clients (i.e., construction of single value Y'_τ that represents all failed clients). Therefore, the scalability of our protocol with respect to the number of failures is better. We show the detailed running time (per protocol round) in Table A.1 in Appendix A.1.

Data transfer To measure the data transfer, we measure the size of the data sent by the client to the server, and the data received by the client from the server. We plot the total data transfer (sent and received) per client in both protocols (i.e., our protocol and the protocol in [BIK+17]) in Figure 6.4c. Our protocol has larger data transfer in scenarios with a low number of clients ($n = 300$) but is more efficient when the number of clients is increased to $n = 1000$. This shows that our protocol scales better with an increasing number of clients. On the other hand, our protocol involves larger data transfer when the input dimension increases. This is mainly because of the larger size of the ciphertext (two

Table 6.4: Wall-clock running time per client for $T = \{100, 500, 1000\}$ FL rounds. The number of clients, dimension and % of failures are fixed to $n = 600, m = 100K, f = 10\%$.

# FL rounds	Total Time	Setup Time
100	451.3 sec	9.6 sec (2.1%)
500	2218.3 sec	9.6 sec (0.4%)
1000	4427.0 sec	9.6 sec (0.2%)

times the size of the plaintext) with respect to masking. We show detailed measurements per protocol round in Table A.2 in Appendix A.2.

Summary In summary, the major improvement of our protocol over SecAgg [BIK+17] is in the running time at the client. Our protocol reduces the overhead on the client's processor up to 8 times when running in large networks. For the server running time, the improvement is only achieved in networks with significantly high failure rates ($\sim 30\%$). Finally, the data transfer is better in our protocol in the case of a large number of clients but worse in the case of huge input dimensions.

6.9 FTSA*: An Optimized Fault-Tolerant SA

We present an optimized version of our protocol FTSA presented in Section 6.6. The optimization can be considered as a major change in the way how the TJL scheme is used to achieve fault-tolerant secure aggregation. We first discuss the main idea of this modification and its consequences on the protocol. Second, we present the new version of our protocol (we call it FTSA*). Finally, we present the new performance results of the protocol.

6.9.1 The Idea

The main limitation of FTSA is that it requires masking the user input $X_{i,\tau}$ with random mask $B_{i,\tau}$ before applying the `TJL.Protect` algorithm. The masks are secretly shared with all users which results in a quadratic complexity in terms of computation and communication. The reason for masking the input $(X_{i,\tau} + B_{i,\tau})$ is to prevent a malicious aggregator to leak the user input. This attack is performed by a malicious aggregator claiming that an actually online user is dropped. It allows the aggregator to obtain two ciphertexts (for the same timestamp) produced by the same user key (the first ciphertext is received from the online user and the second is the zero-protected value computed using the shares of the key). Masking guarantees that the user input is protected when such an attack happens.

We propose a different solution to mitigate this attack. Our solution is to mask the encryption key of the user instead of masking the message itself. The advantage of this approach is that it is performed in the offline phase when the keys are created. Hence, the sharing of the masking key is also performed in the offline phase. Consequently, the

computation and communication complexity of the user in the online phase becomes constant with respect to the number of users.

6.9.2 The Protocol

Similar to FTSA, our optimized secure and fault-tolerant aggregation protocol is composed of an offline and an online phase. In the setup phase, the users compute their TJL secret keys similar to FTSA protocol. In addition, each user generates a masking key $bk_i \leftarrow\$ \{0,1\}^{2l}$ and shares it with the other user using the t-out-of-n secret sharing scheme. Thus, each user U_i will get a share $\{(j, \langle bk_j \rangle_i)\}_{\forall U_j \in \mathcal{U}}$.

In the encryption step of the online phase, the users do not need to mask their input with $B_{i,\tau}$ which is generated from secretly shared seed $b_{i,\tau}$. Instead, they compute the `TJL.Protect` of their input with the masked key and send it to the server. Next, in the construction step, the online users construct the zero-protected values of the offline users similar to FTSA protocol. In addition, they help the aggregator remove the masking keys from the ciphertexts of the online users. The details of the online phase of the new protocol are shown in Figure 6.5.

6.9.3 Security Analysis

Theorem 6.9.1

The protocol FTSA* achieves *Aggregator Obliviousness* in the passive model if the number of corrupted clients is less than threshold t ($|\mathcal{U}_{\texttt{corr}}| < t$) and it achieves *Aggregator Obliviousness* in the active model if the number of corrupted clients is $|\mathcal{U}_{\texttt{corr}}| < 2t - n$.

Proof.

FTSA* uses `TJL` scheme which is secure under the DCR assumption if the users encrypt no more than one ciphertext for each timestamp. Encrypting with a masked key $(k_i + bk_i)$ is equivalent to masking the ciphertext of each user at each timestamp with a fresh random mask $H(\tau)^{bk_i}$ (i.e., $\texttt{TJL.Protect}(k_i + bk_i, \tau, x_{i,\tau}) = (1 + x_{i,\tau}N)H^{\mathbb{Z}^*_{N^2}}(\tau)^{k_i}H^{\mathbb{Z}^*_{N^2}}(\tau)^{bk_i}$). Similar to FTSA, removing the mask requires at least t users to collaborate where t is the threshold of the secret sharing scheme. Hence, similar to the security analysis in Section 6.7, and based on Theorems 6.7.1 and 6.7.2, FTSA* achieves *Aggregator Obliviousness* when ($|\mathcal{U}_{\texttt{corr}}| < t$) in the case of a passive adversary and it achieves *Aggregator Obliviousness* when $|\mathcal{U}_{\texttt{corr}}| < 2t - n$ in the active.

Corollary 6.9.1

In the passive (active) model, security is guaranteed if the minimum number of honest clients $t > \frac{n}{2}$ ($t > \frac{2n}{3}$) and the protocol can recover from up to $\frac{n}{2} - 1$ ($\frac{n}{3} - 1$) client failures.

Secure Aggregation Protocol - Online Phase

- **Online - Encryption (step τ):**
  User u:
  - Protect the private input $X_{i,\tau} \in \mathbb{Z}_R^m$ with the masked key $(k_i + bk_i)$ using TJL scheme: $Y_{i,\tau} \leftarrow \texttt{TJL.Protect}(pp^{\text{TJL}}, k_i + bk_i, \tau, X_{i,\tau})$.
  - Send the protected input $Y_{i,\tau}$ to the server .

  Server:
  - Collect from each user its protected input $Y_{i,\tau}$ (or time out).
  - Denote with $\mathcal{U}_{\text{on}}^{\tau} \subset \mathcal{U}$ the set of online users. Abort if $|\mathcal{U}_{\text{on}}^{\tau}| < t$.
  - Send to each user $U_j \in \mathcal{U}_{\text{on}}^{\tau}$ the list $|\mathcal{U}_{\text{on}}^{\tau}|$.
- **Online - Aggregation (step τ):**
  User u:
  - Receive the list of online users $\mathcal{U}_{\text{on}}^{\tau}$.
  - Compute the share of the zero-value corresponding to all failed users: $\langle Y'_{\tau} \rangle_i \leftarrow \texttt{TJL.ShareProtect}(pp^{\text{TJL}}, \{\langle k_j \rangle_i\}_{\forall U_j \in \mathcal{U} \setminus \mathcal{U}_{\text{on}}^{\tau}}, \tau)$.
  - Compute the share of the masking value corresponding to all online users: $\langle \tilde{Y}_{\tau} \rangle_i \leftarrow \texttt{TJL.ShareProtect}(pp^{\text{TJL}}, \{\langle bk_j \rangle_i\}_{\forall U_j \in \mathcal{U}_{\text{on}}^{\tau}}, \tau)$.
  - Send the share of the protected zero-value $\langle Y'_{\tau} \rangle_i$ and the masking value $\langle \tilde{Y}_{\tau} \rangle_i$ to the server.

  Server:
  - Collect shares from at least t honest users. Denote with $\mathcal{U}_{\texttt{shares}}^{\tau} \subset \mathcal{U}_{\text{on}}^{\tau}$ the set of users. Abort if $|\mathcal{U}_{\texttt{shares}}^{\tau}| < t$.
  - Construct the protected zero-value corresponding to all failed users: $Y'_{\tau} \leftarrow \texttt{TJL.ShareCombine}(\{\langle Y'_{\tau} \rangle_j\}_{\forall U_j \in \mathcal{U}_{\texttt{shares}}^{\tau}}, t)$
  - Construct the masking value corresponding to all online users: $\tilde{Y}_{\tau} \leftarrow \texttt{TJL.ShareCombine}(\{\langle \tilde{Y}_{\tau} \rangle_j\}_{\forall U_j \in \mathcal{U}_{\texttt{shares}}^{\tau}}, t)$
  - Aggregate all the protected inputs of the online clients and the protected zero-value: $C_{\tau} \leftarrow \texttt{TJL.Agg}(pp^{\text{TJL}}, 0, \tau, \{Y_{i,\tau}\}_{\forall U_i \in \mathcal{U}_{\text{on}}^{\tau}}, Y'_{\tau})$
  - Remove the blinding masks:

$$\frac{C_{\tau}}{\tilde{Y}_{\tau}} = \sum_{\forall U_i \in \mathcal{U}_{\text{on}}^{\tau}} X_{i,\tau}$$

Figure 6.5: Detailed description of the online phase of our optimized secure and fault-tolerant aggregation protocol. The text in blue shows the new modifications.

Proof.

The proof follows the proof for Collaries 6.7.1 and 6.7.2

6.9.4 Performance Analysis

Table 6.5: Complexity analysis of the online phase of FTSA* scheme and the state-of-the-art protocols. The values represent the order of complexity. n is the number of clients (order of hundreds) and m is the dimension of the client's input (order of hundreds of thousands).

	Communication		**Computation**	
	Client	Server	Client	Server
SecAgg [BIK+17]	$m+n$	$nm+n^2$	$nm+n^2$	n^2m
SecAgg+ [BBG+20]	$m+\log(n)$	$nm+n\log(n)$	$\log(n)m+\log^2(n)$	$n\log(n)m+n\log^2(n)$
TurboAgg [SGA21b]	$\log(n)m$	$n\log(n)m$	$\log(n)m$	$\boldsymbol{nm}$
FastSecAgg [KRKR20]	$m+n$	$nm+\log^2(n)$	$\log(n)m$	$\boldsymbol{nm}$
FTSA	$m+n$	$nm+n^2$	$m+n^2$	$nm+n^2$
FTSA+	$m+\log(n)$	$nm+n\log(n)$	$m+\log^2(n)$	$nm+n\log^2(n)$
FTSA*	$\boldsymbol{m}$	$\boldsymbol{nm}$	$\boldsymbol{m}$	$\boldsymbol{nm}$

FTSA* shows a significant improvement in the performance of the user in the online phase. More specifically, the computation cost of the user is one call of the algorithm `TJL.Protect` and two calls to `TJL.ShareProtect`. Consequently, the communication cost is sending three ciphertexts. Therefore, both costs depend on the size of the user input (i.e., only $O(m)$). On the other hand, the server computation cost involves computing `TJL.ShareCombine` twice to reconstruct Y'_τ and $\tilde{Y}_\tau$ from their shares. These operations are $O(nm)$. It also involves aggregating the ciphertexts and unmasking the result which is $O(nm)$. Therefore the asymptotic costs of the user and the server are reduced to $O(m)$ and $O(nm)$ respectively which correspond to the theoretical bound since this is equivalent to the cost of federated learning without using secure aggregation. Table 6.5 summarizes the result.

6.10 Conclusions

We presented two versions of a new secure and fault-tolerant aggregation protocol (FTSA and FTSA*). Our solutions use the additively homomorphic encryption scheme of Joye-Libert [JL13] with Shamir's secret sharing to achieve fault tolerance. FTSA protocol outperforms the state-of-the-art protocols in terms of the computational cost of the federated learning clients. On the other hand, our optimized version FTSA* allows performing all secret-sharing operations during the setup phase. Hence, The online phase of FTSA* has only a constant factor of overhead compared to a base federated learning protocol (without secure aggregation). Consequently, FTSA* outperforms all the existing fault-tolerant secure aggregation protocols in terms of the asymptotic cost on both the clients and the server. We show that FTSA* indeed achieves optimal scalability.

Part III

Remote Attestation Protocols

Chapter 7

Fairness-Driven Remote-Attestation

In this chapter, we investigate remote attestation protocols when the IoT network is large, dynamic, and heterogeneous. We first study state-of-the-art solutions for collaborative remote attestation and we identify challenges due to the large size of IoT networks; We focus on two particular problems: the heterogeneity of the devices used in the network and the dynamic nature of IoT networks. To deal with these challenges, we propose FADIA, the first lightweight collaborative remote attestation protocol that is designed with fairness in mind: FADIA enables the fair distribution of attestation load/tasks to achieve better performance. We implement our solution on heterogeneous embedded devices and evaluate it in real scenarios. The evaluation shows that FADIA can (i) increase the lifetime of a network by an order of magnitude and (ii) decrease the remote attestation runtime by a factor of 1.6.

7.1 Remote Attestation

7.1.1 Definition

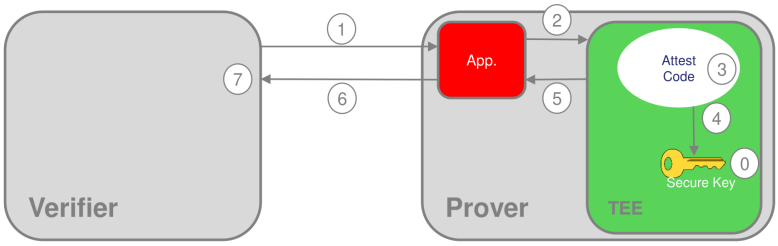


Figure 7.1: Illustration of an remote attestation execution between a prover and verifier .

Remote attestation (RA) is a technique that enables a device called *prover* to prove the integrity of its software configuration to another remote device called *verifier*.

- The *prover*: It is the device that needs to prove its software integrity. The device executes some internal procedure that checks its software configuration and

consequently generates a cryptographic proof. The procedure is usually run inside a trusted execution environment in the device.

- The *verifier*: The device that verifies the integrity of the prover's software. It receives the proof from the prover and validates it. The verifier relies on a root of trust such as a trusted execution environment installed on the prover.

7.1.2 Root of Trust for Remote Attestation Protocols

In remote attestation protocols, the verifier in an attestation protocol relies on a root of trust existing on the prover device. Based on the kind of the root of trust, there are three types of attestations:

Software Root of Trust [GGR09, SLP08, SPvK04, LMP11, SLS+05, KJ03] Software-based attestation relies on assumptions concerning the software implementation of the attestation code. An example of software-based attestations is Pioneer [SLS+05]. Pioneer computes a checksum of device memory using a function that includes side effects in its computation, such that any emulation of this function incurs a timing overhead sufficiently long to detect cheating. Software-based attestation is useful only if the verifier and the prover are directly connected (without passing through intermediate hops) [CFPS09], thus impractical as a remote attestation technique.

Hardware Root of Trust [AFS97, Gro08, KSA+09, SZJvD04] Hardware-based attestation relies on assumptions concerning the hardware that hosts the attestation code. This involves hardware components such as Trusted Platform Modules (TPM) [Gro08] and secure co-processors [Smi02]. These extra hardware components perform verification tasks such as secure booting. The high cost of these techniques limits their use for IoT devices.

Hybrid Root of Trust [FNRT14, KPS+14, EFPT12, BES+15] More recently, hybrid approaches for the root of trust have been proposed. These approaches rely on a combination of hardware and software assumptions. They propose to design a minimal secure hardware architecture on the prover which can be used to perform integrity checks on the prover's running software. It then creates a cryptographic proof and sends it to the verifier. The verifier upon validating this proof authenticates the prover and thus trusts its software.

7.1.3 Collaborative Remote Attestation

Lately, to adhere to the increasing number of embedded devices in the network, collaborative remote attestation protocols are proposed. The idea of collaborative remote attestation is to allow the IoT devices to collaboratively collect attestations from each other and finally produce a single proof for the IoT platform administrators. Hence, it improves the scalability of the protocol.

In this thesis, we are interesting on studying collaborative remote attestation techniques since they are very practical in IoT networks.

7.2 Previous Work on Collaborative Remote Attestation

In this section, we present previous work on collaborative remote attestation. We first regroup them into three categories based on the type of connection topology between provers. Then, we categorize them based on the type of key management. Finally, we identify the main limitations of the previous work in the literature.

7.2.1 Categorization based on Connection Topology

In collaborative remote attestation, provers communicate with each other to collect the proofs. We identify three main types of collaborative remote attestation based on the topology of the connection between provers.

Remote Attestation using Tree Topology Collaborative remote attestation such as SANA [ACI+16], LISA [CERT17], SEDA [ABI+15], and SHeLA [RVW+19] are designed for static or quasi-static networks. These approaches run on devices deployed in a static tree topology. Tree structures provide an efficient and scalable method for verifying a large number of devices. In general, there exist three roles in the tree based on the position of the nodes:

- *Initiator node:* This is the root node of the tree. It initiates the tree construction (attestation) process.
- *Intermediate node:* This corresponds to any node in the tree that has a parent node and children nodes. It forwards messages from its parent to its children and vice versa. It usually also aggregates the attestations received from the children nodes.
- *Leaf node:* This is any node in the tree that has no children nodes. The message forwarding stops at this node. It sends its attestation to its parent node.

Using this tree structure, provers can collaboratively attest by aggregating their proofs. Depending on the type of cryptographic tools used to generate the proof, different aggregation algorithms may be used. For example, SANA [ACI+16] uses a combination of Boldyreva's multi-signature scheme [Bol02] and Boneh et al.'s aggregate signature scheme [BGLS03b] to generate a proof. Other solutions like LISA [CERT17] uses simple message authentication code (MAC) to generate proofs.

Remote Attestation using Mesh Topology Another line of research proposes RA schemes that use a gossip-based mechanism and run in dynamic mesh networks. Examples of such an approach are DARPA [ISTZ16], PADS [ACL+18], SALAD [KBK18], and SARA [DRC+20]. In these protocols, provers share their attestations with all their neighbors, then the received attestations are aggregated and forwarded. The process is

repeated until convergence is reached about the state of all provers. Although gossip-based protocols offer high tolerance against failures and a high degree of autonomy, they suffer from high bandwidth overhead and long runtime.

Remote Attestation using Hybrid Topology Recently, Kohnhauser et al. proposed PASTA [KBK19], an autonomous RA protocol in which provers create multiple spanning trees in the network and generate the so-called tokens to verify the attestations of the provers participating in the tree. Each token embeds the attestations of all the nodes in the tree. Later, these tokens are distributed among all the provers in a gossip-like approach. The parallelized tree generation process in PASTA allows for a relatively low runtime for the protocol. Also, the token distribution offers autonomy (decentralization) to the RA protocol. However, PASTA does not scale well in terms of communication due to the broadcasting of large-size tokens. Also, PASTA requires provers with high storage and computation capabilities. This is because it uses asymmetric cryptography for token generation.

7.2.2 Categorization based on Key Management

In remote attestation protocols, two types of secret keys are defined: the *communication keys* which are used to establish a secure channel between provers and the *attestation keys* which are used for attestation, i.e., to generate a proof of the integrity of a prover's software. We identify two main types of collaborative remote attestation with regard to the underlying key management mechanism.

Remote Attestation using a Single Shared Key In such RA solutions, the network operator defines a single key shared among all provers. This key is used as both a *communication key* and an *attestation key* (e.g., PADS [ACL+18]). Such an approach enables the efficient addition of new devices to the network at runtime. Nevertheless, solutions become inefficient if a single node is compromised (as all nodes need to update their keying material).

Remote Attestation using Pairwise Unique Keys In such RA solutions, each prover holds a unique *attestation key* and a pair-wise mutual *communication key* with each other prover. An example of this type of RA is SALADS [KBK18], PASTA [KBK19], and DARPA [ISTZ16]. This approach ensures better security guarantees and enables an efficient revocation mechanism. However, it does not scale with a high number of provers. Moreover, the addition of new devices to the network becomes costly because each newly joined device requires a key distribution mechanism to run with all existing provers.

7.2.3 Limitation of Previous Work

The aforementioned collaborative remote attestation solutions tackle different challenges such as scalability, security and robustness. Unfortunately, these solutions may become inefficient and sometimes even impractical for some applications of IoT networks. This is

mainly because the current state-of-the-art solutions disregard two common characteristics of IoT networks: (i) heterogeneity of IoT networks and (ii) continuous increase in the size of the network (i.e., devices are added to the network gradually over time).

Heterogeneity of devices: Large-scale RA protocols involve collaborative tasks across devices. These tasks include the generation and forwarding of attestations. Existing solutions perform the distribution of this load (i.e., RA tasks) randomly or uniformly. This may imply a significant performance decrease in heterogeneous networks because it results, usually, in creating bottlenecks, resulting in a degradation of the overall performance. We define heterogeneity in the IoT network as the diversity in the (hardware and/or software) characteristics of the IoT devices. For instance in Industry 4.0 IoT applications [RMK16], sensor devices in the network (ex. Tmote Sky [Cor16]) have Microcontroller Units (MCU) with low computational capabilities compared to a Raspberry Pi operating in robots. The quality of the MCU directly affects the speed of processing the attestation messages from peer devices. Therefore, a RA collaborative protocol that does not consider this gap in the hardware capabilities between devices will end up putting either an equal attestation load on different devices or a higher load on less capable devices. The number of proofs (i.e., attestations) that a node can receive and forward should thus be depending on its capabilities. The heterogeneity of the network can also threaten the lifetime of the services. Running a RA protocol on a device consumes a significant amount of its battery due to the frequent participation in the transmission and reception of attestation messages. Thus, if sensors with lower battery levels engage in many energy-consuming operations, this can end up with battery depletion of some devices causing potential disruption in the service. To this end, we see heterogeneity as a problem that can have a strong impact on both the performance and lifetime of a collaborative RA protocol.

Dynamic nature of IoT networks: With the continuous advancement of IoT applications, IoT networks gradually increase in size (i.e., new devices are added). In collaborative remote attestation, existing devices need to agree on keys with new devices to communicate. These keys ensures secure commination between devices to transmit their attestations. The existing solutions lack scalability and add a high overhead to the runtime of the RA protocol. Based on that, we identify the need for dynamic management and distribution of the key materials as a missing requirement for a practical remote attestation protocol.

7.3 FADIA - Overview

We present our approach to solving the problems mentioned in Section 7.2.3 (namely, heterogeneity and device addition). We design a lightweight collaborative RA protocol (FADIA).

To solve the heterogeneity problem, FADIA is designed with fairness in mind. In a collaborative RA protocol, we define fairness as the ability to distribute the load of the protocol according to the capabilities of the provers. The goal is to increase the performance of the protocol and to reach a better lifetime for the network. In a fair RA

Table 7.1: Comparison between previous work on collaborative remote attestation and FADIA. The big O notation represents the complexity of the cost on a prover with respect to n (number of provers in the network).

	Scalability		*Supported Features*				
	Communication Cost	**Storage cost**	**Mobility**	**Detects hw attacks**	**Autonomous**	**Adding Devices**	**Fairness**
SEDA [ABI+15]	$O(1)$	$O(1)$	○	○	○	○	○
SENA [ACI+16]	$O(1)$	$O(1)$	○	○	○	○	○
LISA$_\alpha$ [CERT17]	$O(\log n)$	$O(1)$	○	○	○	○	○
LISA$_s$ [CERT17]	$O(1)$	$O(1)$	○	○	○	○	○
DARPA [ISTZ16]	$O(n)$	$O(n)$	○	●	○	○	○
SALAD [KBK18]	$O(n)$	$O(n)$	●	○	○	○	○
PADS [ACL+18]	$O(n)$	$O(n)$	●	○	●	○	○
PASTA [KBK19]	$O(n)$	$O(n)$	●	●	●	○	○
FADIA	$O(1)$	$O(1)$	●	●	○	●	●

protocol, provers in FADIA can be assigned a score depending on their capabilities and behave accordingly. This score will be frequently computed and the protocol should adapt to any change. In FADIA, similar to PASTA [KBK19], a group of provers collaborate and create a spanning tree in which parent nodes collect attestations from children nodes. However, in contrast to PASTA, the choice of the position and the number of children of a prover in the tree are adaptively regulated. These are determined by the scoring function which is computed based on the hardware capabilities (e.g., CPU) of the prover and its current residual battery. We, therefore, define a *score* function that outputs a score value between 0 and 1: when the score is closer to 1, the prover can assign more tasks with respect to the RA protocol (for example, the node can have a higher number of children).

To cope with the dynamic nature of the network, the protocol should support the addition of new devices at runtime. We propose to rely on an exisiting Eschenauer-Gligor's (E-G) scheme [EG02] for the distribution of *communication keys*. In FADIA, each node is pre-loaded with a random keyring (a set of communication keys) randomly selected from the key pool. Devices that share at least one key in their keyrings can directly establish a secure communication channel. Thanks to this scheme, FADIA easily addresses the trade-off between the connectivity of the provers and the security of the communication. Furthermore, the addition of a new prover to the protocol does not require any modification of the other provers.

7.4 Building Block: Eschenauer and Gligor's Scheme

We present Eschenauer and Gligor's (E-G) [EG02] scheme that is used in the design of our protocol. E-G key distribution scheme follows a probabilistic approach to efficiently distribute the keys over a large number of devices. It facilitates the addition and the revocation of nodes (and the corresponding keys) in the network without substantial computation and communication overhead on the end devices. The scheme defines a main key pool as a set of p keys of size. The participating devices pick a key ring as a random subset of size r from the key pool. Devices having at least one shared key from their key rings can communicate securely. This scheme has been shown to be simple and highly scalable and is, therefore, suitable for resource-constrained devices. In this work, we utilize E-G scheme to distribute the keys to the provers. Only provers who share common keys can establish secure communication channels.

Connectivity of the Provers Connectivity between provers is defined as the average percentage of provers, a prover can connect to (i.e., communicate with). The connectivity between provers using E-G scheme is equivalent to the probability of two provers sharing at least one key in their key rings (P_s).

$$P_s = 1 - \frac{((p-r)!)^2}{(p-2r)!p!} \tag{7.1}$$

For example, with a key ring of size $r = 300$ and a key pool of size $p = 100000$ we obtain a connectivity of $P_s \approx 0.6$.

7.5 Assumptions and Threat Model

In this section, we describe the assumptions on the network. Then we define our security model.

7.5.1 Network Assumptions

We consider a mesh network topology where devices acting as provers, communicate within their communication range. Additionally, all provers are connected to a more powerful device (the controller $\mathcal{C}$) acting as the verifier. For example, this can be an edge router. The network may contain more than one controller such that these controllers share their information and synchronize their data on a different layer. For the sake of simplicity, in this paper, we consider a single controller that connects to all provers in the network. Both, the provers and the verifier are managed by the network operator $\mathcal{O}$. The participating provers can have heterogeneous characteristics. Moreover, they can be static or mobile within the network. New provers may be added to the network at any point in time.

7.5.2 Security Model

Security assumptions: We assume that provers have the minimal secure hardware features to perform RA [FNRT14]. Additionally, provers are equipped with loosely synchronized real-time clocks. The minimal secure hardware can be implemented using a secure Read-Only-Memory (ROM) to store the keys and a Memory Protection Unit (MPU) that stores FADIA's attestation code. The aforementioned execution space on each prover is referenced as the Trusted Anchor ($\mathcal{T}_\mathcal{A}$). Additionally, we assume that the controller is not compromised and is fully trusted. Furthermore, similar to previous research, FADIA only considers invasive and semi-invasive physical attacks. Thus, non-invasive attacks such as side-channel attacks are out of the scope of this paper. In this context, we rely on a common assumption that for an attacker to successfully bypass $\mathcal{T}_\mathcal{A}$, it needs to take the devices offline for more than some δ_h time which is predefined and known to be non-negligible [CDPG+10, CDPMM08]. This is because such attacks require expensive and complex laboratory equipment and require the full possession of the target for a significant amount of time (from hours to weeks) [Sko12, Sko11]. This becomes even more expensive especially when devices are equipped with tamper-resistant mechanisms [RRC04, ION+18, MGGA17]. We finally assume that the implementation of FADIA and its cryptographic components do not suffer from any security bugs.

Adversarial Model: The main objective of an Adversary is to perform malicious activities by corrupting the memory of a prover and also damaging the network communication while being undetected. We consider two types of adversaries, Software Adversary $\mathcal{A}_s$ and Hardware Adversary $\mathcal{A}_h$.

- $\mathcal{A}_s$ has full control of the execution of a prover apart from the $\mathcal{T}_\mathcal{A}$. It also has full access to the prover's memory except the memory protected by the MPU. Thus, $\mathcal{A}_s$ can launch attacks like spoofing attacks, Man-in-the-middle-attacks, replay attacks.

- $\mathcal{A}_h$ has, in addition to $\mathcal{A}_s$ capabilities, physical access to the devices in the network. This provides it with the ability to leak any secret or modify FADIA's code on the targeted prover. However, this is only possible after turning the prover off for more than δ_h time, as stated previously. δ_h is defined by $\mathcal{C}$ for all provers participating in the protocol.

Definition 7.5.1: Security

FADIA is considered secure if an adversary (under the aforementioned assumptions) cannot forge a "healthy" state of a compromised prover.

In line with other RA schemes [ABI+15, ACI+16, KBK19, ACL+18] we do not consider Distributed Denial of Service (DDoS) attacks. Nevertheless, in Section 7.7 we mention possible ways to detect DDoS attacks in FADIA.

Table 7.2: Notations

Entities	
$\mathcal{P}_i$, $\mathcal{C}$, $\mathcal{O}$	A prover of index i, Collector, and Network Operator
$\mathcal{A}_h$, $\mathcal{A}_s$	Hardware adversary and software adversary
Parameters	
uid	Unique id of a prover
$cntr$	Counter for the number of attestation of a prover
α_g	Max size of the set of uids in an attestation message.
δ_h	Minimum time required by $\mathcal{A}_h$ to compromise a prover
δ_c	Time a prover waits to receive $invite$ before it calls $generateTree()$
c_{max}	Maximum number of children for a node in a tree
c_{limit}	Maximum number of children a prover can accept in a tree
K_{ic}	Secret key shared between $\mathcal{P}_i$ and the controller
K_s	Secret key shared between $\mathcal{C}$ and $\mathcal{O}$
K_{ij}^{ID} , K_{ij}	Secret key (and it corresponding key id) shared between $\mathcal{P}_i$ and $\mathcal{P}_j$
sch	The hash of the software configuration on a prover

7.6 FADIA - Complete Specifications

In this section, we describe the design of FADIA. The protocol is composed of four different phases: the *initialization*, *joining*, *attestation* and *revocation* phases. The first *initialization* phase consists of an offline setup phase where the keying material is installed at all involved provers. During the *joining* phase, a prover identifies itself to the controller. The prover further starts the attestation phase. During the *attestation* phase (which is the core phase of FADIA), the active prover periodically participates in virtual attestation trees to send its attestation report and forward others'. The active prover keeps running this phase until it is dropped from the network (either intentionally because it is detected as malicious or incidentally because it left the network). On such an event, the *revocation* phase starts whereby the dropped prover becomes offline and its keys are revoked. In the following, we describe each phase in more detail.

7.6.1 Initialization Phase

Provers, before participating in FADIA, are considered offline. In order for a prover $\mathcal{P}_i$ to enter the protocol, it has first to be set by the network operator $\mathcal{O}$. $\mathcal{O}$ defines a key pool ($\mathcal{O}.Pool$) according to [EG02], (see § 7.4). This key pool is stored in a safe location (offline). The key pool contains p symmetric keys together with their key ids $\{(\ K^{ID}\ ,\ K\ )\}$. $\mathcal{P}_i$ randomly receives a key ring of size r from the key pool:

$$\mathcal{P}_i.Ring \leftarrow \{(K^{ID}, K)\} \subset \mathcal{O}.Pool$$

$\mathcal{O}$ also assigns a unique id (uid) for $\mathcal{P}_i$ and a cryptographic hash of the current software configuration (sch). $\mathcal{P}_i$ then obtains a unique key K_{ic} (the *attestation key*) derived from the chosen keyring ids and the prover's unique id. This key is computed by $\mathcal{O}$ using a key

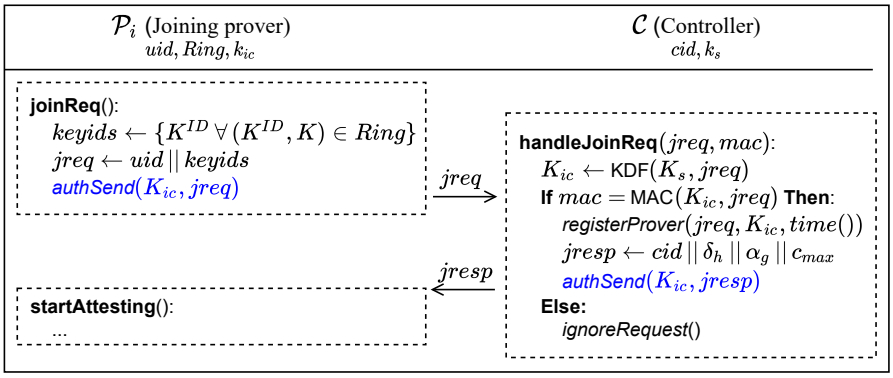


Figure 7.2: FADIA's joining phase

derivation function (KDF) and a secret key (K_s) known only by $\mathcal{O}$ and the controller. More formally,

$$K_{ic} \leftarrow KDF(K_s, \{K^{ID} \forall (K^{ID}, K) \in \mathcal{P}_i.Ring\} \cup \{uid\}) \tag{7.2}$$

K_{ic} is used in the later phases of the protocol to provide secure communication between a $\mathcal{P}_i$ and the controllers. Moreover, $\mathcal{O}$ also defines a function $score()$ which evaluates at the runtime the required load $\mathcal{P}_i$ should take based on its hardware capabilities and its current capacity. This function outputs a value between 0 and 1 that indicates the amount of load $\mathcal{P}_i$ can take (0 indicating that this device should take the least load possible, and 1 indicating that it should take the maximum load that can be given to one device). The implementation of $score()$ depends on the underlying environment and application. For example, in the case of a network of wireless devices with batteries, the function $score()$ will evaluate the battery percentage level of a prover; For a network of devices with microcontrollers of different computational speeds, $score()$ will categorize different types of microcontrollers into different classes, and output a higher value for more powerful classes.

7.6.2 Joining Phase

Once $\mathcal{P}_i$ is initialized, it can join the network. $\mathcal{P}_i$ sends a join request message ($jreq$) to the controller ($\mathcal{C}$) in the network. The join message contains its unique ID (uid) and the set of key ids in its key ring. This message is authenticated using a message authentication code (MAC) computed with K_{ic}. Based on the received uid and the key ids, $\mathcal{C}$ computes K_{ic} (see equation 7.2) and authenticates $\mathcal{P}_i$. Upon validation, $\mathcal{C}$ registers $\mathcal{P}_i$ in the table of provers currently participating in the protocol. The table of registered provers records the current state of each prover (being either *healthy* or *unhealthy*) along with the last time the prover attested. $\mathcal{P}_i$ is first registered as "healthy". $\mathcal{C}$ sends back a response message $jresp$ (authenticated using K_{ic}) to confirm the joining process. $jresp$ includes (i) cid: the controller's id (ii) δ_h which corresponds to the maximum time a prover can stay active without attesting in the network, (iii) α_g which defines the maximum number of attestations that can be aggregated, and (iv) c_{max} that is the maximum number of

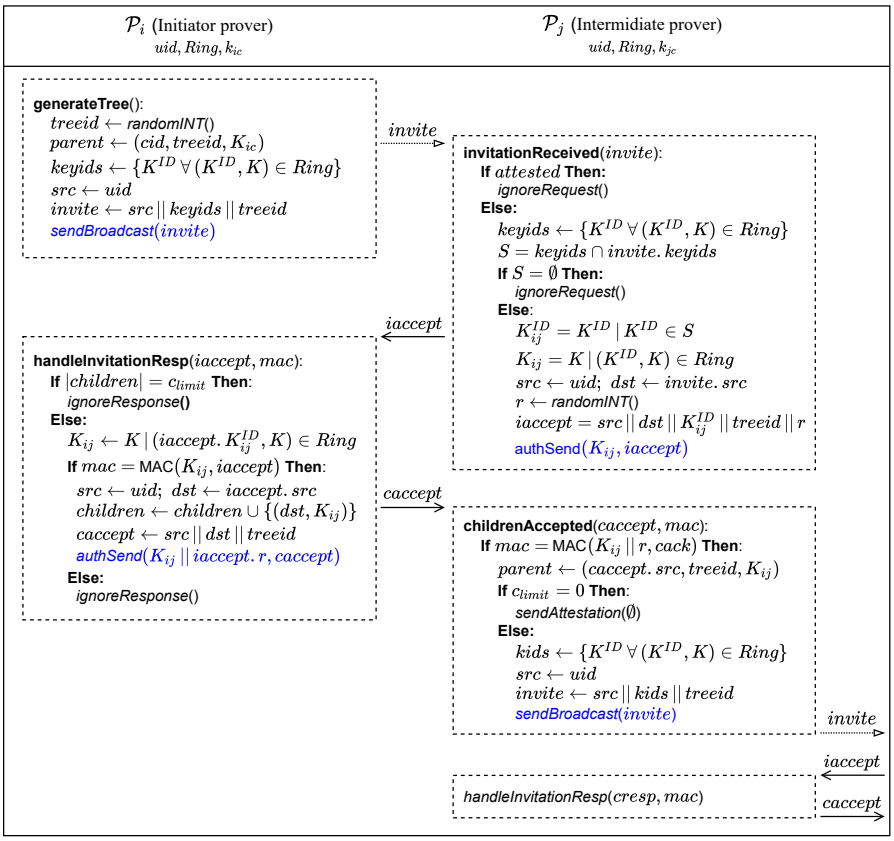


Figure 7.3: Tree construction in FADIA's attestation phase.

children a prover can have. After $\mathcal{P}_i$ is registered, $\mathcal{C}$ regularly checks its status and if $\mathcal{P}_i$ does not attest in δ_h time, its status becomes "unhealthy". Figure 7.2 provides the specification of the joining phase.

7.6.3 Attestation Phase

Provers start running this phase immediately after completing the joining process. $\mathcal{P}_i$ enters a new attestation period every δ_h /2 time. (δ_h /2 is chosen to guarantee that two consequent attestations are always received within no longer than δ_h time.) At each attestation period, a $\mathcal{P}_i$ starts by performing an integrity check on its software configuration. The check is performed against the software configuration hash (sch) stored in the prover. The method of this check and the format of sch is out of the scope of this paper. This can be a simple technique based on computing the hash of the firmware or more complex techniques such as the ones proposed in [AAD+16, DZN+17, ZDA+17, CTR18]. If the integrity check fails, then $\mathcal{P}_i$ quits the attestation phase and will be eventually dropped from the protocol. After succeeding in the integrity check, $\mathcal{P}_i$ participates in the construction of a tree in which it will attest. First, $\mathcal{P}_i$ generates a unique proof of its attestation using the function $generateProof()$. It further evaluates

$score()$ and uses the result to decide on its role in an attestation tree. More specifically, $\mathcal{P}_i$ updates two parameters namely, c_{limit} and δ_c:

- c_{limit} is the maximum number of children $\mathcal{P}_i$ can accept in the upcoming attestation tree.

$$c_{limit} \leftarrow score() \times c_{max}$$

- δ_c (which is less than $\delta_h/2$) is the amount of time $\mathcal{P}_i$ waits to receive a participation invitation to an attestation tree. When δ_c is reached and $\mathcal{P}_i$ did not receive any invitation, it starts the tree construction protocol.

$$\delta_c \leftarrow score() \times \delta_h/2$$

Neighboring provers that share common keys and are ready to provide their attestations construct a tree. Note that if a prover does not share any key with its neighbors, it directly sends the attestation to the controller. However, such cases appear with a low probability in realistic networks. For example, if the average number of neighbors for a prover is 5, and the connectivity $P_s = 0.6$, then the probability of a prover being isolated is $(1 - P_s)^5 \approx 0.01$. The tree construction and the attestation collection processes are described next.

7.6.3.1 Tree Construction

A tree construction starts by an initiator prover ($\mathcal{P}_i$) after waiting for δ_c time. The latter broadcasts an invitation message (*invite*) to its neighbors. The invitation message includes the unique id (*uid*) of $\mathcal{P}_i$ as well as a tree id (*treeid*) (generated from a timestamp to guarantee freshness) and the set of key ids which $\mathcal{P}_i$ holds in its keyring. When $\mathcal{P}_j$ receives *invite*, it first checks if it did not attest in the last $\delta_h/2$ time. Then it checks if it shares at least one key with $\mathcal{P}_i$. Let K_{ij} denote the shared key between $\mathcal{P}_i$ and $\mathcal{P}_j$, and K_{ij}^{ID} be its corresponding key id. $\mathcal{P}_j$ responds with an invitation acceptance message (*iaccept*) containing the key id K_{ij}^{ID} . The message also includes the unique ids of the source and the destination, the tree id sent by $\mathcal{P}_i$, and a random value (r) and a Message Authentication Code (MAC) computed using the shared key K_{ij} . $\mathcal{P}_i$ accepts $\mathcal{P}_j$ as a child only if it has not acquired c_{limit} children yet and responds with a child acceptance message *caccept* authenticated with the shared key concatenated with the random value (r). To this end, both provers established a secure channel with the shared key K_{ij} . Next, $\mathcal{P}_j$ either extends the tree and thus acts as an intermediate prover, or it finishes the tree construction process thus acts as a leaf prover. A prover acts as a leaf prover in two cases: either it does not accept any children (i.e., $c_{limit} = 0$) or its *invite* message is timed out without receiving any response from its neighbors. The details of the tree construction messages are shown in Figure 7.3. The secure channels established between parent nodes and children nodes in the tree are used next to transmit the attestation messages.

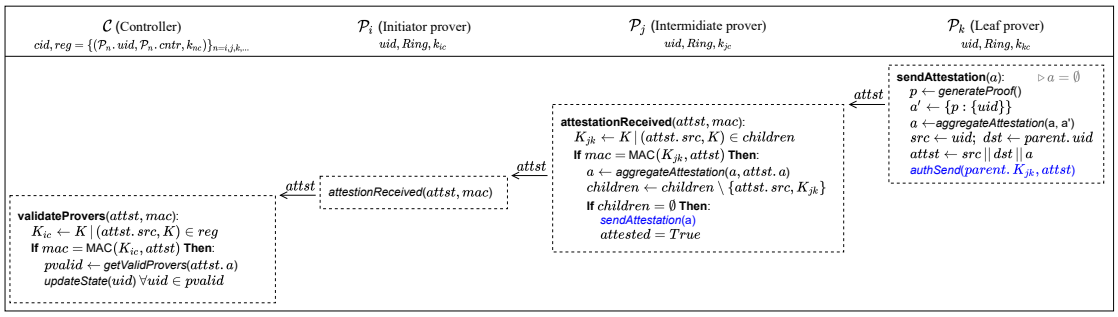


Figure 7.4: Attestation collection in FADIA's attestation phase.

Algorithm 1 Aggregation of two attestations.

$a_x = \{proof_a : uids_a, proof_b : uids_b, ...\}$
$uids_x \subset \{uid_1, uid_2, ..., uid_n\}$
Algorithm `aggregateAttestation`$(a_1,\ a_2)$
  $res \leftarrow a_1$
  for $proof, uids$ **in** a_2 **do**
    if $|uids| = \alpha_g$ **then**
      $a_2 \leftarrow a_2 \setminus \{proof : uids\}$
      $a_2 \leftarrow a_2 \setminus \{proof : uids\}$
    else
      for $proof', uids'$ **in** res **do**
        if $|uids| + |uids'| < \alpha_g$ **then**
          $a_2 \leftarrow a_2 \setminus \{proof : uids\}$
          $res \leftarrow res \setminus \{proof' : uids'\}$
          $res \leftarrow res \cup \{proof \oplus proof' : uids \cup uids'\}$
          break
  for $proof, uids$ **in** a_2 **do**
    $res \leftarrow res \cup \{proof : uids\}$

7.6.3.2 Attestations collection

The leaf nodes send attestation messages ($attst$) to their parents. The attestation message of $\mathcal{P}_i$ contains the generated proof which is computed as follows:

$$generateProof() : proof \leftarrow MAC(K_{ic}, uid \,||\, cntr \,||\, treeid) \tag{7.3}$$

where $cntr$ is a counter that is incremented each time $\mathcal{P}_i$ attests. Parent nodes aggregate the attestation messages from their children using the function $aggregateAttestation()$. Similarly, the new attestation message is sent to the parent and processed. This is repeated until the initiator prover receives all the aggregated attestation messages. It then sends the final attestation message to the controller. An attestation message is composed of multiple sets of prover ids. The number of provers in a set is controlled by the parameter α_g. Each set is linked with a single proof which is the XOR of all proofs provided by the provers in that set. The algorithm that describes $aggregateAttestation()$ is depicted in Algorithm 1.

The granularity of the aggregation of the attestation is parameterized by α_g. If α_g is 0, then none of the proofs are XORed, and thus, all individual proofs are transmitted to the controller. If α_g is larger than the number of provers participating in the tree, all proofs are XORed forming a single proof for all provers.

When $\mathcal{C}$ receives the aggregated attestation it validates the proofs. For each set, it computes the MAC according to equation 7.3 using the unique keys (K_{jc}) of each prover $\mathcal{P}_j$. For each verified aggregated proof, the status of all provers in the set is updated. More specifically, $\mathcal{C}$ updates the time of the last proof received from these provers by the current time. We show the attestations collection details in Figure 7.4.

7.6.4 Key Revocation Phase

When prover $\mathcal{P}_i$ becomes "unhealthy", it cannot be trusted anymore. So it is required by $\mathcal{C}$ to revoke all the shared keys between $\mathcal{P}_i$ and the other provers. Since $\mathcal{C}$ knows the ids of all the keys in the keyrings of all provers, it can find out the affected devices (i.e., the ones which share at least a key with the "unhealthy" device). $\mathcal{C}$ sends a revocation message ($revk$) to each of them. The message contains the uid of the receiving prover, and the set of key ids that should be revoked. $revk$ is authenticated with a MAC using K_{ic} of the corresponding prover.

$$revk \leftarrow cid \,||\, uid \,||\, \text{affectedKeys}() \tag{7.4}$$

When a prover receives $revk$, it removes the affected keys from its keyring. When the size of its keyring goes under a certain threshold θ, the prover goes to the offline state and requires reinitializing to go back online.

7.6.5 Role of the *score* function

The fairness by design approach can be achieved thanks to the tuning of mainly two parameters set with the help of the *score* function: c_{limit} and δ_c. this allows configuring the behavior and the computation load of each prover during the attestation phase and their correct setting hence ensuring a fair distribution of the load caused by FADIA on the active devices.

The first parameter c_{limit} represents the number of children a prover can hold during one attestation round in the virtual attestation tree. The number of children has a direct impact on the amount of load put on the prover. The load involves the use of cryptographic tools (MACs) to establish a secure channel with each child and forward the attestation proofs.

The second parameter δ_c is the time at which a prover waits to receive the invitation message ($invite$) before it decides to start its own tree and sends $invite$ itself. It is important to mention here that when a prover finishes an attestation round, it can switch to a sleeping state since it has completed its attestation for this round. Research on wireless ad-hoc networks has studied the scheduling of the nodes sleeping state to optimize the network lifetime [MM08]. Inlighted by these studies we control the sleeping time of a prover based on a fair policy. In FADIA for each attestation round, a prover is first actively waiting for invitation messages, then performs the requested attestation operation in the tree and finally switches to a sleep state until the next round. Consequently, δ_c is a critical parameter that influences directly the average amount of time a prover spends in the sleep state and hence optimizes its resource consumption. Research has shown

that being in an active state (listening or sending) can be intensively resource-consuming compared to being in a sleep state [KJ19]. Therefore, δ_c parameter is also used to control the amount of load on a prover in FADIA.

In FADIA, these two parameters are calculated at runtime by each prover. Their corresponding values are updated at each round of the attestation phase using the *score* function. We present examples of how to build this function depending on the scenario of deployment in section § 7.8.3.

7.7 Security Analysis

The main goal of an adversary is to perform malicious activities and evade detection. However, a remote attestation scheme should identify the presence of malicious actors in the network to safeguard the network's operations. We consider the system secure if an attacker cannot forge a "healthy" state of a "non-healthy" prover. The remainder of this section informally discusses the security of FADIA w.r.t. adversarial assumptions mentioned in 7.5.

Attacks Performed by $\mathcal{A}_s$:

- Spoofing attacks: FADIA is immune to attackers trying to spoof a prover's identity. Since all message exchanges are protected by keys stored in an inaccessible location for software attackers and can only be accessed through $\mathcal{T}_\mathcal{A}$, an attacker will not be able to produce authenticated messages without the keys from the key ring. Please note that the *invite* message is an exception, as this message is not authenticated. However, this does not affect the security of FADIA since attackers spoofing this message will not be able to complete the 3-way handshake, (i.e., respond with a valid *caccept*).

- MITM attacks: FADIA will identify this attack as it suffers the same limitations as spoofing attacks. $\mathcal{A}_s$ using this attack technique will not be able to manipulate messages without being detected since the integrity of these messages is ensured using the keys from the key rings that are protected by $\mathcal{T}_\mathcal{A}$.

- Replay attacks: FADIA prevents replay attacks since all messages are unique and cannot be used twice without being detected. Specifically, freshness is guaranteed in *invite* messages thanks to using a timestamp in the *treeid*. Similarly, *iaccept* and *caccept* includes randomness for each prover-to-prover communication. Further, *attst* messages are also resilient to replay attacks since they contain a counter which is incremented at each attestation round.

- DoS Attacks: Although FADIA does not include these types of attacks in its threat model, it is worth noting that FADIA can detect the effect of such attacks. This is because $\mathcal{A}_s$ performing DoS attacks on a prover (or prover group) will prevent the attestation of these provers. The controller will, therefore, inevitably discover a missing attestation from the provers under attack.

Table 7.3: Benchmarks and energy consumption measurements of cryptographic functionalities of FADIA when implemented on Tmote Sky and Raspberry PI 2 devices.

		HMAC-SHA256		SHA256	
		Time (ms)	**Energy (mJ)**	**Time (ms)**	**Energy (mJ)**
PI 2	**32 B**	0.068	-	0.025	-
	4 KB	1.075	-	1.049	-
	8 KB	2.083	-	2.032	-
	32 KB	8.131	-	8.079	-
SKY	**32 B**	63.28	0.3384	15.54	0.0862
	4 KB	1035	5.5908	988	5.3388
	8 KB	1998	10.7892	1960	10.6128

Attacks Performed by $\mathcal{A}_h$ In addition to software adversaries, hardware adversaries have the ability to tamper with a device to extract the keys from its keyring or alter the attestation code. Under the assumption that a hardware adversary needs to take the system offline for at least a δ_h duration, FADIA provides resilience against these attackers. This is because FADIA requires that within every δ_h period, each prover provides evidence of its "healthiness". This helps the network owner to ensure that the provers are not taken offline and thus not corrupted. On the other hand, a hardware adversary may use leaked keys to attack other provers sharing the same key. However, the key revocation process ensures that if a prover no longer participates in the protocol, these keys are revoked. Additionally, even if an attacker leaked all the key materials from a prover, the attacker will not be able to forge legitimate proof as this forgery involves the targeted prover's K_{ic}. The state of a "unhealthy" device will thus not be forged by $\mathcal{A}_h$.

7.8 Performance Analysis

In this section, we evaluate the performance of FADIA in a heterogeneous network and demonstrate the advantages originating from its fairness-driven approach by evaluating the energy consumption, the computational cost and the bandwidth. We further study its performance in a homogeneous network and show that even in this case, FADIA outperforms the relevant state-of-the-art solutions, namely PASTA [KBK19] and SALAD [KBK18].

7.8.1 Implementation of FADIA on Tmote Sky and Raspberry PI 2

To illustrate heterogeneity, we implement FADIA on two types of devices: Tmote Sky [Cor16] and Raspberry PI 2. The Tmote Sky which represents a resource-constrained device is equipped with an 16-bits 8 MHz msp430 MCU, 10 KB of RAM, and 48 KB of non-volatile memory. On the other hand, the Raspberry PI 2 is more powerful as it is equipped with a $900MHz$ quad-core ARM Cortex-A7 CPU, 1 GB of RAM, and 32 GB of non-volatile memory. Both types of devices are equipped with CC2420 RF transceivers.

The CC2420 chip operates on 2.4 GHz and is compliant with IEEE 802.15.4 standards. We do not follow a certain security architecture for implementing FADIA on the devices. However, FADIA can be implemented based on any security architecture. Previous research has shown that achieving a security architecture on sensor devices is indeed possible [EFPT12, BES+15, DRT17, NBM+17]. In our implementation, and without loss of generality, an ARM TEE can be used for the Raspberry PI, and TYTAN [BES+15] can be used for the Tmote Sky. We use HMAC-SHA256 for authenticating the messages and creating the proofs. We use the SHA256 of the device's firmware as the software configuration hash (*sch*).

In order to conduct our study, we first evaluate the cost of one attestation round for a prover in terms of the execution time and the energy consumption of FADIA. Since one round mainly involves MAC and hash computation operations, we have obtained some benchmarks on HMAC-SHA256 and SHA-256 respectively. Results are shown in Table 7.3. As expected, Tmote Sky takes more time to generate attestation proofs. We also benchmark the throughput and the round-trip time (RTT) of the CC2420 transceiver in a real environment. The throughput on the application layer is 25.2 Kbps and the RTT is 61.4 ms.

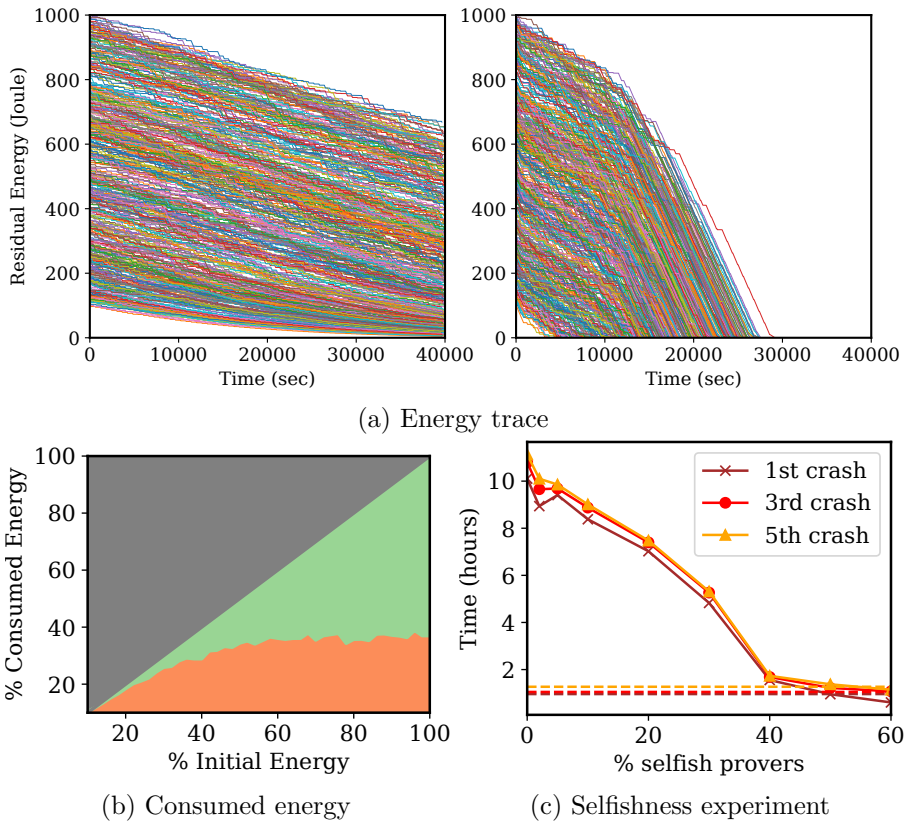
7.8.2 Environment for Experimental Evaluation

We first evaluate FADIA on a heterogeneous network to measure the influence of fairness on the performance in terms of energy consumption and computational cost. To evaluate the benefits of fairness, we consider two variants of FADIA. Variant (1) with fairness activated and variant (2) without fairness. Our results show that fairness improves the lifetime of the network and the runtime of the RA protocol. We then evaluate the performance of FADIA on a homogeneous network in terms of storage, computation and communication cost. Our results show that FADIA outperforms the state-of-the-art solutions. Additionally, we evaluate the robustness of FADIA against selfish provers. We also evaluate the efficiency of the revocation phase. We perform our simulations using the Omnet++ simulator [VH08]. We implement FADIA on the application layer of the devices [1]. For the lower layers, we use a simplified medium access control version which makes sure that no device within the communication range transmits at the same time. Provers communicate in a half-duplex fashion and store messages in queues when the medium is busy.

7.8.3 Evaluation in Heterogeneous Networks

Test Case 1: Optimizing the energy consumption To measure the optimization of energy consumption, we simulate FADIA on 500 Tmote Sky sensors acting as provers. *sch* is set as the SHA256 hash of the firmware of size 30 KB. The network and cryptographic delays are set according to the values measured in Table 7.3. Additionally, we consider a simple energy consumption model: The model updates the current energy consumption based

[1] The C++ implementation of FADIA for Omnet++ can be found at https://github.com/MohamadMansouri/FADIA



(a) Energy trace

(b) Consumed energy

(c) Selfishness experiment

Figure 7.5: Evaluation of the energy consumption. (a) the residual energy over time. w/ (left side) and w/o (right side) activating fairness. (b) the average consumed energy w.r.t. initial energy after 16.6 hours of running FADIA with fairness activated. The green-colored area represents the residual energy and the orange-colored area represents the consumed energy at the end of the experiment. (c) the time taken until 1st, 3rd, and 5th crash (energy depletion at a prover) with different percentages of selfish provers in the network. Dotted lines represent the results when fairness is not activated.

on the status of the transceiver and the microcontroller of the prover. The transceiver can either be transmitting, listening, or OFF. Similarly, the microcontroller can either be ON or idle. For each of the following statuses, the energy is computed according to the energy measurements shown in Table 7.4. The provers move in a random waypoint model at a linear speed uniform between $1mps$ and $2mps$ in a $300m \times 300m$ area. Each of the provers is equipped with a battery of $1000J$ max capacity. The initial energy level a prover starts with is chosen randomly (uniformly $[100J, 1000J]$). Evaluation of different random distribution functions for the initial energy are shown in appendix B.1. Note that the maximum capacity of an alkaline AA battery is around $13,000J$. But we use $1000J$ as the maximum capacity for the seek of the feasibility of the experiment. We run FADIA for $60,000$ seconds. For variant (1) of FADIA we implement the $score()$ to return the current battery level of a device. Alternatively, for variant (2) $score()$ always

STATE	Energy Consumption
MCU ON	0.0054 J
TX	0.0585 J
RX	0.0654 J
IDLE	0.00016 J

Table 7.4: Energy consumption of Tmote Sky devices while in different states. MCU ON represents a state where the microcontroller performing computations. TX and RX represents the CC2420 state while transmitting or receiving respectively. IDLE represents and idle state where the transceiver is off and the MCU is in low power mode. Data taken from Tmote Sky datasheet [Cor16].

returns 0.5.

We measure the consumption of energy of each prover with respect to time. Figure 7.5a shows the energy traces of the provers in both variants of FADIA. As expected, variant (2) of FADIA (i.e. fairness is not activated), shows a fast depletion of the energy of all the provers while for variant (1), most of the provers remain active after $40,000$ seconds. The reason for the fast depletion of energy in the "unfair" protocol (i.e. variant 2) is that provers with low energy are treated indifferently from high energy provers. This leads to putting a significant load on these devices due to the high (i.e. unfair) number of children they need to collect attestations from. Moreover, since δ_c is not adapted to the energy level of the device, provers may spend more time waiting to be invited to a tree construction process. This keeps the transceivers of these devices in the listening state for a longer time instead of switching sooner to the OFF/idle state. This brings a serious problem since the part that mostly consumes energy in sensor devices is the transceiver. This is the case for Tmote Sky as shown in Table 7.4. Notice that, when provers with low battery crash, this decreases the number of provers in the network and causes fewer tree constructions to appear, causing a domino effect and faster depletion for other devices. Differently, in the case of FADIA variant (1), the low battery provers preserve their the energy which prevents the early loss of provers.

We also look into the consumed energy of the provers at the end of the experiment (i.e., after 16.6 hrs). We group provers that had similar initial battery levels at initialization and we measure their average consumed energy level after the experiment. Figure 7.5b shows the relation between the consumed energy with respect to the initial energy. The graph shows that provers with high initial battery levels (more than 50%) consumed more energy than provers with fewer initial battery levels thus providing a longer lifetime to the network. Additionally, we measure the time taken until we detect 1st, 3rd and 5th crash of a prover (i.e., its energy is completely depleted). We observe that the lifetime of the provers in a fair protocol is an order of magnitude longer.

Test Case 2: Optimizing the runtime To measure the optimization of the runtime achieved thanks to fairness, we evaluate the runtime of FADIA on heterogeneous static tree topology. In this scenario, we use two types of devices such that 50% of provers

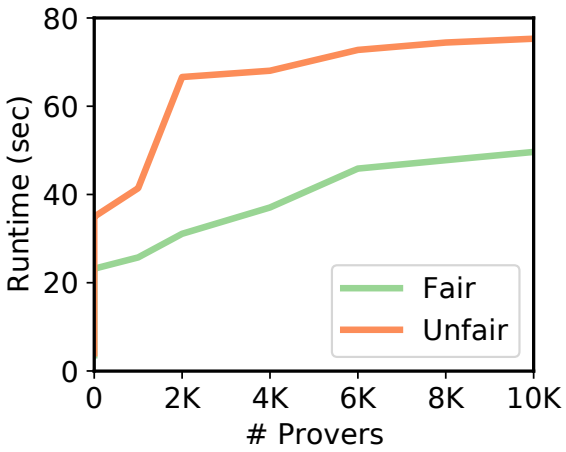


Figure 7.6: Runtime of FADIA with (in green) and without (in orange) fairness integrated in dynamic network.

are Tmote Sky (MSP430) devices and the other 50% are Raspberry PI 2 devices. Both types of devices use the same transceiver (CC2420). We use the throughput, network delay, and the cryptographic delays for each type of device according to the benchmarks measured in table 7.3. We measure the time taken from the start of the tree construction until the final attestation is sent from the root node to the collector. The function $score()$ is defined such that it returns 0.05 for the MSP430 provers and 1 for Raspberry PI provers. Accordingly, the number of accepted children (c_{limit}) will be 1 for the less powerful provers and 20 for the powerful ones. On the contrary, we simulate FADIA variant (2) which assigns 10 children for each prover regardless of its type. Figure 7.6 shows the runtime results of both approaches with respect to a varying number of provers in the network. The results show that FADIA with fairness option can run 1.6 faster in static topologies.

7.8.4 Evaluation in Homogeneous Networks

Memory consumption of FADIA Each prover in FADIA stores one key ring. The storage consumption derived from the key ring is $(4B + 32B) \times R$, where R is the size of the keyring. The prover also stores uid $(4B)$, the controller key k_{ic} $(32B)$, the attestation counter $cntr$ $(4B)$, and other FADIA parameters $(20B)$. The total memory consumption is $56 + 36 \times R$ Bytes. For example, with a key ring of size 300, this results in $10.6KB$ of memory. It is worth noticing that the memory consumption depends only on the key ring size and is independent of the number of provers on the network. This provides very high scalability compared to most of the state-of-the-art solutions that incur a cost linear to the number of provers. Our experimental study shows that FADIA can run on $10,000$ provers while each of them consumes only 10.6 KB which is significantly low compared to PASTA with $780KB$ of memory usage and SALAD with $365KB$.

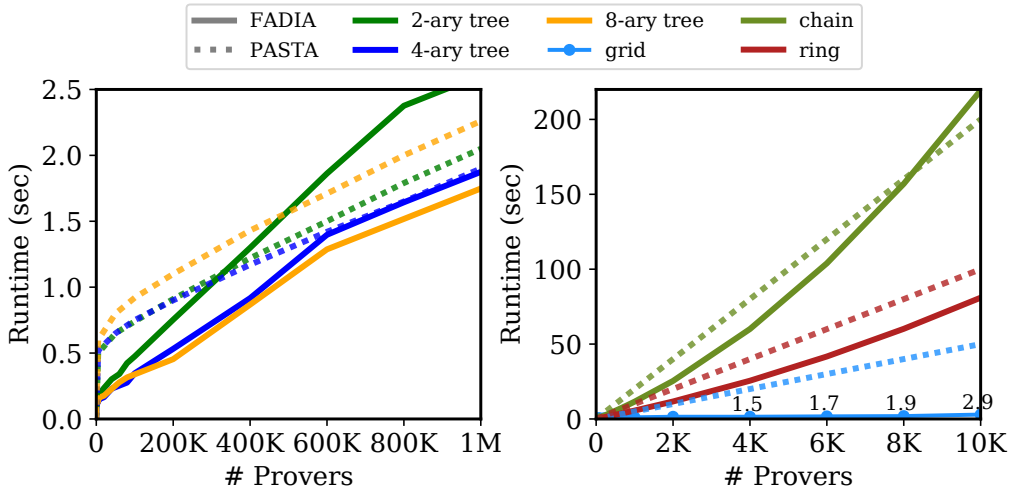


Figure 7.7: Runtime of FADIA and PASTA with a different number of provers in tree, grid, chain, and ring topologies.

Runtime of FADIA To measure the runtime for FADIA, we have implemented the protocol in a static network defined under four common topologies: the tree, chain, ring and grid topologies. We consider the construction of a single attestation tree where all provers participate in it. The running time is evaluated by measuring the time it takes until the report is collected by the controller. For a fair comparison with the state-of-the-art, the scenario, the types of devices, the network delays and the cryptographic benchmarks are all set as the ones used in the evaluation of PASTA protocol in [KBK19]. We use ESP32-PICO-D4 devices in the simulation as provers and set the size of their firmware to $50KB$. The throughput of the provers on the application layer is 12.51 MB/s and the round trip time is 4.63 ms. The time a device takes to generate the proof is set according to Table B.1 in the appendix. The provers perform 10 attestation rounds and the runtime of the attestation is averaged. The position of the nodes is randomized between rounds to force reinitialization of the tree construction. We set the keyring size $r = 300$, and $\alpha_g = inf$. Figure 7.7 shows the runtime results of FADIA and PASTA.

We observe that FADIA shows a low runtime in tree topologies. It can attest $1,000,000$ provers in less than 2 seconds in a 4-ary tree. The runtime of FADIA and PASTA are close to each other in a tree topology since most of the attestation time corresponds to network delays. Moreover, FADIA shows better performance at higher degree trees because messages are broadcasted during the construction of the attestation tree, whereas for PASTA, one-to-one tree commitment requests are sent from the prover to its neighbors. Additionally, FADIA is faster than PASTA by approximately 17 times for grid topologies and 1.3 times for ring topologies. In grid topologies, the tree construction results in an unbalanced tree. This creates a problem for PASTA since the tree construction happens in two steps. In the first step, all provers first commit to the tree. Then in the second step, all provers receive the aggregated commitment. This requires all provers to wait for the tree construction to finish before they start attesting. FADIA does not have this

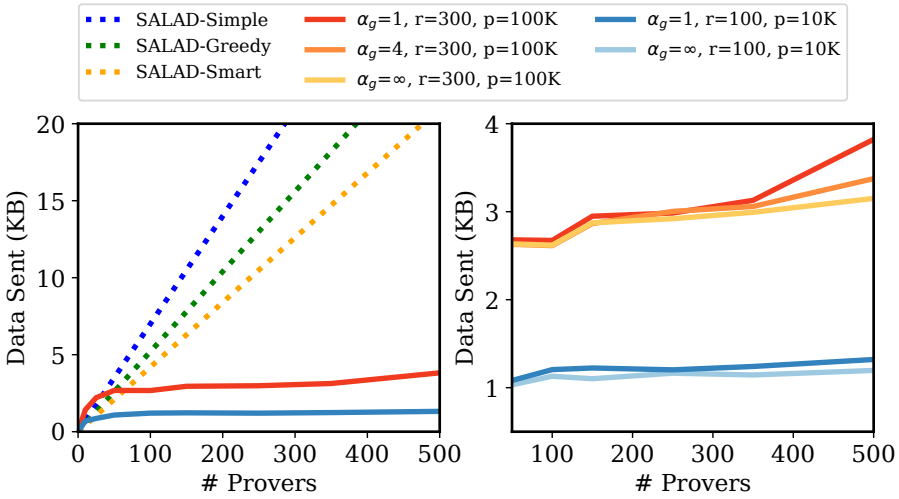


Figure 7.8: Average amount of data sent by a prover during one round of attestation in FADIA and SALAD protocols.

problem since the leaf nodes can immediately send their attestations without waiting for the tree construction to finish. On the other hand, PASTA outperforms FADIA in chain topologies with a large number of provers (i.e., > 8000). This is explained by the fact that in FADIA, every prover sends all the key ids in the keyring to its neighbors which turns to be not effective in large chain topologies. Fortunately, such topologies do not often exist in real applications.

Bandwidth consumption of FADIA We evaluate the bandwidth consumption of FADIA in a dynamic network. We consider devices moving in a random waypoint model (i.e., provers choose random destinations and move toward them) at a linear speed uniform between $1mps$ and $2mps$. The provers move in an area of $500m \times 500m$. We measure the average amount of data sent and received by a prover in an attestation round (i.e., for all provers to attest). Notice that the scenario, the device type, the network delays and the cryptographic benchmarks are all set the same as the ones used in the evaluation of SALAD protocol in [KBK18]. We use Stellaris LM4F120H5QR devices in the simulation as provers and set the size of their firmware to $30KB$. The throughput of the provers on the application layer is set to 35.0 kbps and the round trip time is set to $15ms$. The cryptographic delays are set according to Table B.1 in the appendix. We choose the value of α_g as 1, 10, and infinity. We also use different values for the keyring and pool sizes: more specifically we set $r = 100, p = 10,000$ and $r = 300, p = 100,000$. Both cases give the same connectivity of the graph being $P_s = 0.6$ (see 7.4). Figure 7.8 shows the results. In particular, the results show that FADIA has highly scalable bandwidth consumption since the data consumption is nearly constant with respect to the number of provers. It also shows that the bandwidth consumption depends mostly on the size of the keyring. However, this cost always remains less than the average consumption of SALAD increases linearly with the number of provers in the network.

7.8.5 Evaluation with Selfish Provers

We evaluate the impact of selfish provers on the lifetime of the network (time till 1st, 3rd and 5th crash of a prover). A selfish prover is a prover that does not will to participate in the tree-generation process. It thus greedily attests individually to the controller and goes to sleep state as early as possible. Note that such extreme selfish behavior can be easily detected. We consider this extreme case to evaluate the worst-case scenario. A more careful selfish prover will still collaborate however less than it is supposed to. We consider the same energy consumption scenario in *Test Case 1* (see 7.8.3). However, selfish provers are chosen with an initial battery level greater than $250J$. Figure 7.5c shows the results with different percentages of selfish provers. We observe that FADIA is robust against selfishness. This is because the lifetime drops significantly only when more than 40% of the provers are selfish. With a high number of selfish provers, the performance degrades since there is a sort of race toward the collector to provide attestation. This creates too much contention between provers accessing the wireless medium leading to the attestations being delayed.

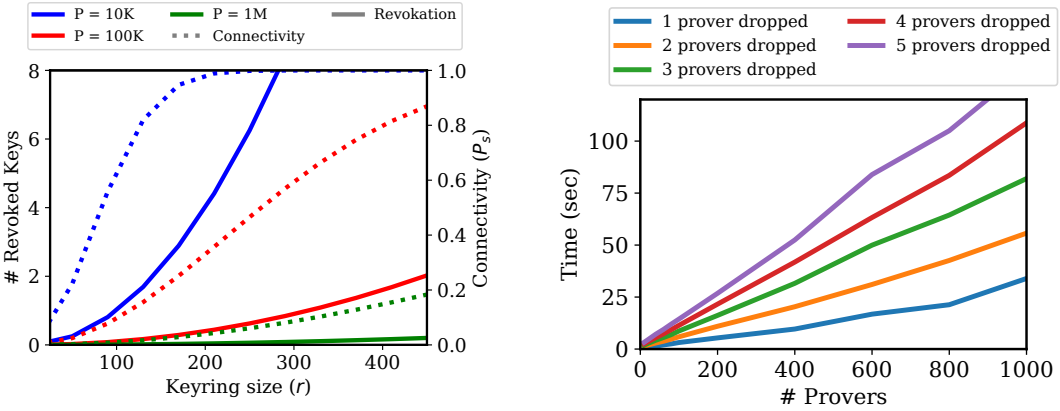
7.8.6 Evaluation of the Revocation Phase in FADIA

The revocation phase starts when a prover is dropped from the network. The controller detects the event during the next δ_h time and sends a revocation message *revk* to all the affected provers. To measure its efficiency, we evaluate the following:

- ($\mathbb{E}_1$) *The number of provers affected when a device is dropped.*
- ($\mathbb{E}_2$) *The number of connections affected.*
- ($\mathbb{E}_3$) *The number of keys revoked for each prover.*
- ($\mathbb{E}_4$) *Time taken until all affected provers receive the revocation.*

$\mathbb{E}_1$ is estimated as the number of provers in the network multiplied by the probability of two provers being connected ($n \times P_s$). This means that the average number of affected devices is proportionally tied to the connectivity of the provers in the network. An example of a keyring of size 300 and a key pool of size 100,000 gives $P_s = 0.6$, thus in this example 60% of the provers are affected by the revocation process. However, this is not a problem since most of the keys revoked at the prover are not actually used. Therefore, a more important metric to look into is the average number of connections affected (i.e., connections established using a key that is revoked). This is equal to $c \times r/p$ which estimates $\mathbb{E}_2$ (c is the total number of connections established at the time of the revocation). Following our example, the averaged affected connections will be only $0.003 \times c$ thus only 0.3% of the current connections need to be re-established. Third, $\mathbb{E}_3$ estimates the expected amount of decrease in the size of a keyring on each revocation process and it is equal to the following:

$$\sum_{i=0}^{K} \frac{\prod_{j=0}^{i-1}(K-j) \times \prod_{j=0}^{K-i-1}(P-K-j)}{\prod_{j=0}^{K-1}(P-j)} \times i \times \binom{i}{K} \tag{7.5}$$



(a) Number of revoked keys when one proved is removed

(b) Revocation time

Figure 7.9: Revocation efficiency. (a) shows the number of revoked keys per prover with respect to different key ring sizes. (b) shows the average amount of time taken till the revocation process is completed with respect to a variable number of provers in the network.

We demonstrate this equation in Figure 7.9a. The figure shows the relation between the average number of keys revoked per prover and the connectivity of the graph with respect to different keyring sizes and key pool sizes. We can see that for $r = 300$ and $p = 100,000$, 0.8 keys are revoked per prover on average. It thus requires 190 provers to drop for an active prover to revoke half of its keys. Finally, to evaluate $\mathbb{E}_4$, we run a simulation in which we deploy FADIA in a network of provers and we set $r = 300$ and $p = 100,000$. During the run of the protocol, we start the revocation process at a random point in time. We measure the time it takes till all provers receive the revocation message. We also consider cases where multiple revocations start simultaneously. Results are shown in Figure 7.9b. The revocation process increases linearly with the number of provers in the network and is performed within 34 seconds for 1,000 provers. When multiple provers are dropped simultaneously, the revocation time scales linearly with the number of dropped provers.

7.9 Conclusion on Fairness-Driven Remote-Attestation

In this chapter, we proposed FADIA, a lightweight collaborative attestation protocol that can be deployed on heterogeneous networks of IoT devices. FADIA is the first RA protocol that integrates fairness in its design and deploys a scalable key management mechanism based on E-G scheme. We show that fairness is an important feature of remote attestation protocols. It can increase the performance of the protocol by a factor of 1.6 in a network where Tmote Sky sensors and Raspbery PIs coexist. Additionally, the lifetime of the network can increase by an order of magnitude, thus achieving fewer failures. We also show that FADIA outperforms the state-of-art solutions in terms of scalability on

networks of a large number of provers. Finally, we evaluated the efficiency of the key revocation in FADIA. Our results show that the key management in FADIA offers a good tradeoff between the efficiency of adding and removing provers to the network.

Chapter 8

Final Conclusion and Future Work

8.1 Conclusion

In this thesis, we have studied the design of security protocols that are suitable for IoT. We focused on two security protocols: Secure Aggregation and Remote Attestation. We observe that the existing secure aggregation protocols suffer from several limitations when used in IoT context. Namely, the existing SA protocols do not consider malicious users. We argued that this type of solution does not fit IoT use cases since the devices may be compromised. Moreover, we identified that existing SA protocols do not cope with the dropouts of users. This poses a limitation for these solutions due to their large scale where failures of some devices are inevitable. On the other hand, we observe that existing remote attestation protocols are also not suitable for IoT. The major challenges facing the efficient design of RA protocols are the heterogeneity of IoT devices and the dynamic nature of the IoT network. Therefore, this thesis proposed solutions that fix the limitations of the existing work for secure aggregation and remote attestation.

For secure aggregation protocols, we proposed VSA as a new secure aggregation protocol that considers a stronger threat model than existing SA protocols. VSA preserves the privacy of the users' input and guarantees the correct computations of the aggregation result even when the aggregator and few users are acting maliciously. To design VSA, we built over an existing work (PUDA [LEÖM15]) which considers only a malicious aggregator and uses user-generated tags to verify the aggregation result. To cover the case of malicious users, we integrated a newly designed tagging protocol into PUDA. Our tagging protocol (which is secure in the malicious model) involves new entities called taggers that issue the tags. The tagging protocol guarantees that only honest users receive valid tags for their inputs. Therefore, thanks to the tagging protocol, VSA ensures that neither the malicious users nor the malicious aggregator can produce an incorrect aggregation result.

Additionally, we proposed FTSA as a new secure and fault-tolerant aggregation protocol dedicated to federated learning application. FTSA allows a robust computation of the aggregation result even when some users drop during the aggregation process. To build our new solution, we improved the Joye-Libert secure aggregation scheme (`JL` [JL13]) and proposed a threshold variant of it (`TJL`). Our new `TJL` uses an Integer

Secret Sharing scheme (ISS see Section 2.3.1) and it allows a threshold number of users to protect a zero-value on behalf of other users. FTSA leverages TJL scheme to recover the aggregation result by enabling online users to submit protected zero-value on behalf of the offline ones. We deployed FTSA in the context of federated learning and compared it with the state-of-the-art. Our evaluation showed that FTSA can scale better than all existing solutions and it achieves the theoretical limit in terms of scalability with respect to both the number of users and the size of the machine-learning models.

Last and not least, we proposed FADIA as a new collaborative remote attestation protocol with fairness integrated into its design. FADIA enables the fair distribution of attestation load/tasks to achieve better performance. We show that fairness is an important feature for remote attestation protocols. Fairness can increase the performance of the protocol by a factor of 1.6 in a network where Tmote Sky sensors and Raspbery PIs coexist. The lifetime of the network can increase by an order of magnitude, thus achieving fewer failures. We also show that FADIA outperforms the state-of-art-solutions in terms of scalability.

8.2 Future Work

In this thesis, we studied secure aggregation and remote attestation protocols. Possible future research directions to investigate for each of the solutions can be listed below:

- The verifiable secure aggregation protocol (VSA) proposed in this thesis is secure under static corruptions. To achieve security under adaptive corruptions we need to improve the design of our protocol to cover adversaries that may corrupt honest users during the execution of the protocol. In the current design, we build our tagging protocol over an OT protocol [CO15] which is secure in the static corruption model. To replace this protocol with an OT protocol secure under adaptive corruptions, we need a technique to still be able to compute the tags from the OT messages as done in the current design.

- The verifiable secure aggregation protocol (VSA) proposed in this thesis verifies the users' inputs by checking that they are less than an upper bound. In some applications, it might be required to check a more complex condition on the inputs. Fortunately, our solution uses garbled circuit to implement this condition which can be generalized to a circuit that checks any predicate. It is interesting to study what predicates are useful for secure aggregation and their implementations.

- The fault-tolerant secure aggregation protocol (FTSA) proposed in this thesis presents a method to compute protected inputs on behalf of offline users. To achieve fault tolerance in the case of verifiable secure aggregation solutions, it is possible to use the same approach to compute the tags on behalf of offline users. However, we need some distributed zero-knowledge proof techniques to prove that the computed ciphertext is an actual protection of zero-value.

- In this thesis, we studied secure aggregation as a solution to preserve the privacy of clients in federated learning. However, as discussed in Chapter 5, secure aggregation

should be used with differential privacy to achieve better privacy guarantees. Thus, studying the suitability of differential privacy techniques with secure aggregation is an interesting area of research that complements our work.

- The collaborative remote attestation protocol (FADIA) proposed in this thesis improves the performance of the RA by optimizing the management of the attestation between provers. Research work on the trusted execution environment which runs the attestation code and the integrity-checking algorithm can also lead to a significant improvement of RA protocols for IoT.

Appendices

Appendix A

Appendices for Chapter 6

A.1 Detailed Measurements of the Running Time

We show in Table A.1 the running time of each of the phases of our protocol for the clients and the server.

Table A.1: Wall-clock running time for the clients and the server for our protocol. The dimension is fixed to $m = 10000$.

	# Clients	Failures	Registeration	KeySetup	Encryption	Aggregation
Client	100	0%	1.08 ms	1607 ms	976 ms	18.6 ms
		30%	0.86 ms	1583 ms	966 ms	1071 ms
	300	0%	0.90 ms	4730 ms	1234 ms	56.8 ms
		30%	0.94 ms	4718 ms	1233 ms	1642 ms
	600	0%	0.89 ms	9202 ms	1733 ms	109 ms
		30%	0.92 ms	8754 ms	1630 ms	2346 ms
Server	100	0%	0.005 ms	1.98 ms	1.97 ms	1235 ms
		10%	0.007 ms	1.75 ms	1.7 ms	25196 ms
		30%	0.009 ms	1.63 ms	1.65 ms	19176 ms
	300	0%	0.018 ms	16.6 ms	16.2 ms	7887 ms
		10%	0.009 ms	16.9 ms	15.9 ms	294910 ms
		30%	0.009 ms	17.4 ms	16.1 ms	226290 ms
	600	0%	0.016 ms	109 ms	102 ms	41057 ms
		10%	0.017 ms	116 ms	105 ms	1357778 ms
		30%	0.014 ms	96.7 ms	100 ms	962070 ms

A.2 Detailed Measurements of the Data Transfer

We show in Table A.2 the size of the transferred data in each of the phases of our protocol.

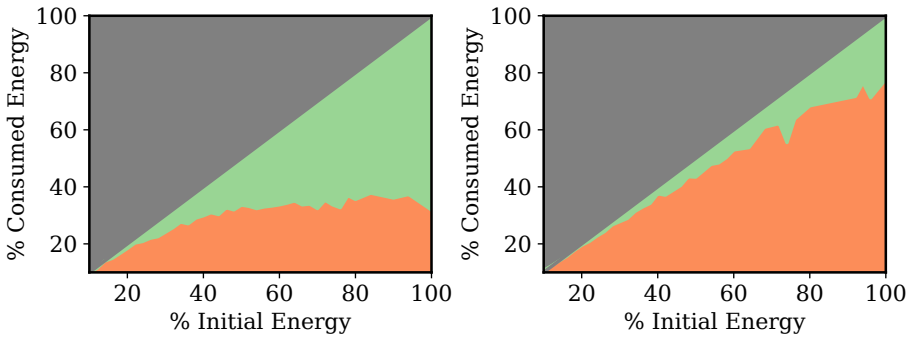
Table A.2: Data transfer per client for our protocol. The dimension is fixed to $m = 10000$.

# Clients	Failures		Registration	KeySetup	Encryption	Aggregation
100	0%	*sent*	0.13 KB	32.84 KB	62.47 KB	1.96 KB
		rcvd	– KB	45.73 KB	– KB	5.46 KB
		total	0.13 KB	78.57 KB	62.47 KB	7.42 KB
	30%	*sent*	0.13 KB	32.84 KB	62.46 KB	58.81 KB
		rcvd	– KB	45.73 KB	– KB	3.81 KB
		total	0.13 KB	78.57 KB	62.46 KB	62.62 KB
300	0%	*sent*	0.13 KB	135.95 KB	79.00 KB	5.86 KB
		rcvd	– KB	174.62 KB	– KB	16.50 KB
		total	0.13 KB	310.57 KB	79.00 KB	22.36 KB
	30%	*sent*	0.13 KB	135.95 KB	79.00 KB	67.09 KB
		rcvd	– KB	174.62 KB	– KB	11.53 KB
		total	0.13 KB	310.57 KB	79.00 KB	78.62 KB
600	0%	*sent*	0.13 KB	400.79 KB	97.30 KB	11.72 KB
		rcvd	– KB	478.13 KB	– KB	33.05 KB
		total	0.13 KB	878.92 KB	97.30 KB	44.77 KB
	30%	*sent*	0.13 KB	400.78 KB	97.30 KB	72.96 KB
		rcvd	– KB	478.13 KB	– KB	23.12 KB
		total	0.13 KB	878.91 KB	97.30 KB	96.08 KB

Appendix B

Appendix for Chapter 7

B.1 Energy consumption evaluation



(a) Normal distribution (b) Pareto distribution

Figure B.1: Evaluation of the fairness in energy consumption for 500 provers. Figures show the average percentage of consumed energy with respect to the percentage of initial energy. Details of the experiment are shown in section 7.8.3 (a) the initial energy of the provers is chosen following a normal distribution ($\mu = 500, \sigma = 200$). (b) the initial energy of the provers is chosen following a shifted Pareto distribution (80% of the provers has an energy between 100J and 300J. The rest has an energy between 300J and 1000J).

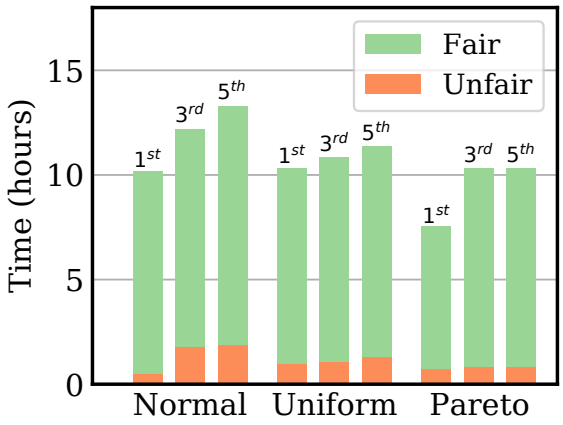


Figure B.2: Time taken until 1st, 3rd, and 5th crash is seen in a network of provers running FADIA with (in green) and without (in orange) fairness integrated.

B.2 Benchmarks of ESP32-PICO-D4 Devices and Stellaris LM4F120H5QR Microcontrollers

	ESP32-PICO-D4		LM4F120H5QR	
Function	Size	Time (ms)	Size	Time (ms)
SHA256	5 KB	13.171	32 KB	40.02
HMAC-SHA256	16 B	0.042	32 B	0.23
	1024 B	0.301	32 kB	39.86

Table B.1: Benchmarks of SHA256 and HMAC-SHA256 with different input sizes on ESP32-PICO-D4 devices and Stellaris LM4F120H5QR MCUs (data from [KBK19] and [KBK18] respectively).

Bibliography

[AAB+17] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, pages 1093–1110, USA, 2017. USENIX Association.

[AAD+16] T. Abera, N. Asokan, L. Davi, J. E. Ekberg, T. Nyman, A. Paverd, A. R. Sadeghi, and G. Tsudik. C-flat: Control-flow attestation for embedded systems software. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16. Association for Computing Machinery, 2016.

[ABI+15] N. Asokan, F. Brasser, A. Ibrahim, A. R. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann. Seda: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15. Association for Computing Machinery, 2015.

[ABM+20] Michel Abdalla, Florian Bourse, Hugo Marival, David Pointcheval, Azam Soleimanian, and Hendrik Waldner. Multi-client inner-product functional encryption in the random-oracle model. In *Security and Cryptography for Networks*. Springer International Publishing, 2020.

[ÁC11] Gergely Ács and Claude Castelluccia. I have a dream! (differentially private smart metering). In *Information Hiding*. Springer Berlin Heidelberg, 2011.

[ACF+18a] Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In *Advances in Cryptology – CRYPTO 2018*. Springer International Publishing, 2018.

[ACF+18b] Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In *Advances in Cryptology – CRYPTO 2018*. Springer International Publishing, 2018.

[ACI+16] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A. R. Sadeghi, and M. Schunter. Sana: Secure and scalable aggregate network attestation. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16. Association for Computing Machinery, 2016.

[ACL+18] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise. Pads: Practical attestation for highly dynamic swarm topologies. In *2018 International Workshop on Secure Internet of Things (SIoT)*, 2018.

[ADMC17] Muhammad Rizwan Asghar, György Dán, Daniele Miorandi, and Imrich Chlamtac. Smart meter data privacy: A survey. *IEEE Communications Surveys Tutorials*, 2017.

[AFS97] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*, 1997.

[AGRW17] Michel Abdalla, Romain Gay, Mariana Raykova, and Hoeteck Wee. Multi-input inner-product functional encryption from pairings. In *Advances in Cryptology – EUROCRYPT 2017*. Springer International Publishing, 2017.

[AMMK20] Sébastien Andreina, Giorgia Azzurra Marson, Helen Möllering, and Ghassan Karame. Baffle: Backdoor detection via feedback-based federated learning. *CoRR*, abs/2011.02167, 2020.

[AMS+15] Giuseppe Ateniese, Luigi V. Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *Int. J. Secur. Netw.*, 10(3), sep 2015.

[ASY+18] Naman Agarwal, Ananda Theertha Suresh, Felix Yu, Sanjiv Kumar, and H. Brendan McMahan. Cpsgd: Communication-efficient and differentially-private distributed sgd. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18. Curran Associates Inc., 2018.

[BBG+20] 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*, CCS '20. Association for Computing Machinery, 2020.

[BEG+19] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. *CoRR*, abs/1902.01046, 2019.

[BEMGS17] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17. Curran Associates Inc., 2017.

[BES+15] F. Brasser, B. El Mahjoub, A. Sadeghi, C. Wachsmann, and P. Koeberl. Tytan: Tiny trust anchor for tiny devices. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[BF01] Dan Boneh and Matthew Franklin. Efficient generation of shared rsa keys. *J. ACM*, 48(4):702–722, jul 2001.

[BGLS03a] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology — EUROCRYPT 2003*. Springer Berlin Heidelberg, 2003.

[BGLS03b] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*. Springer Berlin Heidelberg, 2003.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. CCS '17, pages 1175–1191, New York, NY, USA, 2017. Association for Computing Machinery.

[BJL16] Fabrice Benhamouda, Marc Joye, and BenoîT Libert. A new framework for privacy-preserving aggregation of time-series data. *ACM Trans. Inf. Syst. Secur.*, 18(3), mar 2016.

[BLV+21] Lukas Burkhalter, Hidde Lycklama, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. Rofl: Attestable robustness for secure federated learning. *CoRR*, 2021.

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, Oct 2008.

[Bol02] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Yvo G. Desmedt, editor, *Public Key Cryptography — PKC 2003*. Springer Berlin Heidelberg, 2002.

[BSK+19] Keith Bonawitz, Fariborz Salehi, Jakub Konečný, Brendan McMahan, and Marco Gruteser. Federated learning with autotuned communication-efficient secure aggregation. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, 2019.

[BSMD10] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, page 15. USENIX Association, 2010.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *Theory of Cryptography*. Springer Berlin Heidelberg, 2011.

[BT20] Constance Beguier and Eric W. Tramel. Safer: Sparse secure aggregation for federated learning, 2020.

[BTL+21] Carlo Brunetta, Georgia Tsaloli, Bei Liang, Gustavo Banegas, and Aikaterini Mitrokotsa. Non-interactive, secure verifiable aggregation for decentralized, privacy-preserving learning. In *Information Security and Privacy*, Cham, 2021. Springer International Publishing.

[BVH+20a] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, 2020.

[BVH+20b] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*. PMLR, 2020.

[BWAA18] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed optimisation for non-convex problems. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 560–569. PMLR, 10–15 Jul 2018.

[Can20] Ran Canetti. Universally composable security. *J. ACM*, 67(5), sep 2020.

[CCMT09] Claude Castelluccia, Aldar C-F. Chan, Einar Mykletun, and Gene Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Trans. Sen. Netw.*, 5, 2009.

[CDE+18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. Spd $\mathbb{Z}_{2^k}$: Efficient mpc mod 2^k for dishonest majority. In *Advances in Cryptology – CRYPTO 2018*. Springer International Publishing, 2018.

[CDK+22] Megan Chen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, Abhi Shelat, and Ran Cohen. Multiparty generation of an rsa modulus. *Journal of Cryptology*, 35(2), 2022.

[CDPG+10] M. Conti, R. Di Pietro, A. Gabrielli, L. V. Mancini, and A. Mei. The smallville effect: Social ties make mobile networks more secure against node capture attack. In *Proceedings of the 8th ACM International Workshop on Mobility Management and Wireless Access*, MobiWac '10. Association for Computing Machinery, 2010.

[CDPMM08] M. Conti, R. Di Pietro, L. Vincenzo Mancini, and A. Mei. Emergent properties: Detection of the node-capture attack in mobile wireless sensor networks. In *Proceedings of the First ACM Conference on Wireless Network Security*, WiSec '08. Association for Computing Machinery, 2008.

[CERT17] X. Carpent, K. ElDefrawy, N. Rattanavipanon, and G. Tsudik. Lightweight swarm attestation: A tale of two lisa-s. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17. Association for Computing Machinery, 2017.

[CFPS09] Claude Castelluccia, Aurélien Francillon, Daniele Perito, and Claudio Soriente. On the difficulty of software-based attestation of embedded devices. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09. Association for Computing Machinery, 2009.

[CGB17] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17. USENIX Association, 2017.

[Cha88] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1988.

[CMS13] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining*. Springer Berlin Heidelberg, 2013.

[CMT05] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, 2005.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In *Proceedings of the 4th International Conference on Progress in Cryptology – LATINCRYPT 2015 - Volume 9230*, Berlin, Heidelberg, 2015. Springer-Verlag.

[CO18] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *Security and Cryptography for Networks*. Springer International Publishing, 2018.

[Cor16] Moteiv Corporation. Tmote sky details. "http://www.snm.ethz.ch/snmwiki/pub/uploads/Projects/tmote_sky_datasheet.pdf", 2016.

[Cou18] Geoffroy Couteau. New protocols for secure equality test and comparison. In *Applied Cryptography and Network Security*. Springer International Publishing, 2018.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In *Advances in Cryptology - CRYPTO 2003*, pages 265–281, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[CTR18] X. Carpent, G. Tsudik, and N. Rattanavipanon. Erasmus: Efficient remote attestation via self-measurement for unattended settings. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018.

[CYD20] Zhou Chuanxin, Sun Yi, and Wang Degang. Federated learning with gaussian differential privacy. In *Proceedings of the 2020 2nd International Conference on Robotics, Intelligent Control and Artificial Intelligence*, RICAI 2020, New York, NY, USA, 2020. Association for Computing Machinery.

[DA16] Tassos Dimitriou and Mohamad Khattar Awad. Secure and scalable aggregation in the smart grid resilient against malicious entities. *Ad Hoc Networks*, 50, 2016.

[DBN+01] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. Advanced encryption standard (aes), 2001-11-26 2001.

[DCSW20] Ye Dong, Xiaojun Chen, Liyan Shen, and Dakui Wang. Eastfly: Efficient and secure ternary federated learning. *Computers & Security*, 94:101824, 2020.

[DGBL+16] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16. JMLR.org, 2016.

[DH06] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 2006.

[DHCP21] Phan The Duy, Huynh Nhat Hao, Huynh Minh Chu, and Van-Hau Pham. A secure and privacy preserving federated learning approach for iot intrusion detection system. In *Network and System Security*, Cham, 2021. Springer International Publishing.

[Dig19] Larry Dignan. Iot devices to generate 79.4zb of data in 2025, says idc. https://www.zdnet.com/article/iot-devices-to-generate-79-4zb-of-data-in-2025-says-idc/, 2019. [Online; accessed 14-Octobor-2022].

[DJN10] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6), Dec 2010.

[DN03] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '03. Association for Computing Machinery, 2003.

[DOS18] Ivan Damgård, Claudio Orlandi, and Mark Simkin. Yet another compiler for active security or: Efficient mpc over arbitrary rings. In *Advances in Cryptology – CRYPTO 2018*, volume 10992 of *Lecture Notes in Computer Science*. Springer, 2018.

[DOT18] Pratish Datta, Tatsuaki Okamoto, and Junichi Tomida. Full-hiding (unbounded) multi-input inner product functional encryption from the k-linear assumption. In *Public-Key Cryptography – PKC 2018*. Springer International Publishing, 2018.

[DP21] Anirban Das and Stacy Patterson. Multi-tier federated learning for vertically partitioned data. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021.

[DR14] Cynthia Dwork and Aaron Roth. 2014.

[DRC+20] E. Dushku, M. M. Rabbani, M. Conti, L. V. Mancini, and S. Ranise. Sara: Secure asynchronous remote attestation for iot systems. *IEEE Transactions on Information Forensics and Security*, 2020.

[DRT17] K. M. El Defrawy, N. Rattanavipanon, and G. Tsudik. Hydra: hybrid design for remote attestation (using a formally verified microkernel). *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2017.

[Dwo06] Cynthia Dwork. Differential privacy. In *Automa, Languages and Programming*. Springer Berlin Heidelberg, 2006.

[Dwo07] Morris J. Dworkin. Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. Technical report, Gaithersburg, MD, USA, 2007.

[DZN+17] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A. R. Sadeghi. Lo-fat: Low-overhead control flow attestation in hardware. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17. Association for Computing Machinery, 2017.

[EA20] Ahmed Roushdy Elkordy and Amir Salman Avestimehr. Secure aggregation with heterogeneous quantization in federated learning. *CoRR*, abs/2009.14388, 2020.

[EFPT12] K. Eldefrawy, A. Francillon, D. Perito, and G. Tsudik. SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust. In *NDSS 2012, 19th Annual Network and Distributed System Security Symposium, February 5-8, San Diego, USA*, 2012.

[EG02] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02. Association for Computing Machinery, 2002.

[ET12] Zekeriya Erkin and Gene Tsudik. Private computation of spatial and temporal power consumption with smart meters. In *Applied Cryptography and Network Security*, pages 561–577. Springer Berlin Heidelberg, 2012.

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, New York, NY, USA, 2015. Association for Computing Machinery.

[FMLF21] Joaquín Delgado Fernández, Sergio Potenciano Menci, Charles Lee, and Gilbert Fridgen. Secure federated learning for residential short term load forecasting. *CoRR*, abs/2111.09248, 2021.

[FMM+21] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, Thomas Schneider, Hossein Yalame, and Shaza Zeitouni. Safelearn: Secure aggregation for private federated learning. In *2021 IEEE Security and Privacy Workshops (SPW)*, 2021.

[FNRT14] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik. A minimalist approach to remote attestation. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014.

[Fre12a] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public Key Cryptography – PKC 2012*, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Fre12b] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *Public Key Cryptography – PKC 2012*. Springer Berlin Heidelberg, 2012.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[FYB18] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. *CoRR*, abs/1808.04866, 2018.

[GGG+14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Advances in Cryptology – EUROCRYPT 2014*. Springer Berlin Heidelberg, 2014.

[GGR09] R. W. Gardner, S. Garera, and A. D. Rubin. Detecting code alteration by creating a temporary memory bottleneck. *IEEE Transactions on Information Forensics and Security*, 2009.

[GHG+21] Jiqiang Gao, Boyu Hou, Xiaojie Guo, Zheli Liu, Ying Zhang, Kai Chen, and Jin Li. Secure aggregation is insecure: Category inference attack on federated learning. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[GJ11] Flavio D. Garcia and Bart Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In *Security and Trust Management*. Springer Berlin Heidelberg, 2011.

[GLL+21] Xiaojie Guo, Zheli Liu, Jin Li, Jiqiang Gao, Boyu Hou, Changyu Dong, and Thar Baker. Verifl: Communication-efficient and fast verifiable aggregation for federated learning. *IEEE Transactions on Information Forensics and Security*, 16, 2021.

[Gro08] Trusted Computing Group. Trusted platform module (tpm) summary. https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-Summary_04292008.pdf, 2008.

[GWY+18] Karan Ganju, Qi Wang, Wei Yang, Carl A. Gunter, and Nikita Borisov. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, New York, NY, USA, 2018. Association for Computing Machinery.

[HKKH21] Changhee Hahn, Hodong Kim, Minjae Kim, and Junbeom Hur. Versa: Verifiable secure aggregation for cross-device federated learning. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2021.

[HKR+18] Andrew Hard, Chloé M Kiddon, Daniel Ramage, Francoise Beaufays, Hubert Eichner, Kanishka Rao, Rajiv Mathews, and Sean Augenstein. Federated learning for mobile keyboard prediction, 2018.

[HL17] Eduard Hauck and Julian Loss. Efficient and universally composable protocols for oblivious transfer from the cdh assumption. Cryptology ePrint Archive, Paper 2017/1011, 2017. https://eprint.iacr.org/2017/1011.

[HTGW18] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N. Wright. Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies*, 2018, 2018.

[iA93] Shun ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5, 1993.

[IHPS11] R. S. H. Istepanian, S. Hu, N. Y. Philip, and A. Sungoor. The potential of internet of m-health things "m-iot" for non-invasive glucose level sensing. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 5264–5266, 2011.

[IKOS06] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from anonymity. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 239–248, 2006.

[ION+18] V. Immler, J. Obermaier, K. Kuan Ng, F. Xiang Ke, J. Lee, Y. Peng Lim, W. Koon Oh, K. Hoong Wee, and G. Sigl. Secure physical enclosures from covers with tamper-resistance. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018.

[ISTZ16] A. Ibrahim, A. R. Sadeghi, G. Tsudik, and S. Zeitouni. Darpa: Device attestation resilient to physical attacks. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '16. Association for Computing Machinery, 2016.

[JGP+21] Zoe L. Jiang, Hui Guo, Yijian Pan, Yang Liu, Xuan Wang, and Jun Zhang. Secure neural network in federated learning with model aggregation under multiple keys. In *2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2021.

[JL13] Marc Joye and Benoît Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2013.

[JNMALC22] Tayyebeh Jahani-Nezhad, Mohammad Ali Maddah-Ali, Songze Li, and Giuseppe Caire. Swiftagg: Communication-efficient and dropout-resistant secure aggregation for federated learning with worst-case security guarantees, 2022.

[KBK18] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser. Salad: Secure and lightweight attestation of highly dynamic and disruptive networks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18. Association for Computing Machinery, 2018.

[KBK19] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser. A practical attestation protocol for autonomous embedded systems. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[KFM04] M.N. Krohn, M.J. Freedman, and D. Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 226–240, 2004.

[KJ03] R. Kennell and L. H. Jamieson. Establishing the genuinity of remote computer systems. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03. USENIX Association, 2003.

[KJ19] W. Kim and I. Jung. Smart sensing period for efficient energy consumption in iot network. *Sensors*, 2019.

[KKR+23] Aditya Pribadi Kalapaaking, Ibrahim Khalil, Mohammad Saidur Rahman, Mohammed Atiquzzaman, Xun Yi, and Mahathir Almashor. Blockchain-based federated learning with secure aggregation in trusted execution environment for internet-of-things. *IEEE Transactions on Industrial Informatics*, 19(2):1703–1714, 2023.

[KLS21] Peter Kairouz, Ziyu Liu, and Thomas Steinke. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*. PMLR, 2021.

[KMA+19] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *CoRR*, abs/1912.04977, 2019.

[KOB21] Ferhat Karakoç, Melek Önen, and Zeki Bilgin. Secure aggregation against malicious users. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, SACMAT '21. Association for Computing Machinery, 2021.

[KPS+14] P. Koeberl, S. Patrick, S. Schulz, A. R. Sadeghi, and V. Varadharajan. Trustlite: A security architecture for tiny embedded devices. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14. Association for Computing Machinery, 2014.

[KRKR20] Swanand Kadhe, Nived Rajaraman, Onur Ozan Koyluoglu, and Kannan Ramchandran. Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning. *CoRR*, abs/2009.11248, 2020.

[KSA+09] C. Kil, E. C. Sezer, A. M. Azab, P. Ning, and X. Zhang. Remote attestation to dynamic system properties: Towards providing complete system integrity evidence. In *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, 2009.

[KShS12] Benjamin Kreuter, Abhi Shelat, and Chih hao Shen. Billion-gate secure computation with malicious adversaries. In *21st USENIX Security Symposium (USENIX Security 12)*, 2012.

[KTC20] Youssef Khazbak, Tianxiang Tan, and Guohong Cao. Mlguard: Mitigating poisoning attacks in privacy preserving distributed collaborative learning. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, 2020.

[Kus13] David Kushner. The real story of stuxnet, Feb 2013.

[Lar17] Selena Larson. Fda confirms that st. jude's cardiac devices can be hacked, Jan 2017.

[LCV19] Changchang Liu, Supriyo Chakraborty, and Dinesh Verma. *Secure Model Fusion for Distributed Learning Using Partial Homomorphic Encryption.* Springer International Publishing, 2019.

[LEM14a] Iraklis Leontiadis, Kaoutar Elkhiyaoui, and Refik Molva. Private and dynamic time-series data aggregation with trust relaxation. In *Cryptology and Network Security*, pages 305–320. Springer International Publishing, 2014.

[LEM14b] Iraklis Leontiadis, Kaoutar Elkhiyaoui, and Refik Molva. Private and dynamic time-series data aggregation with trust relaxation. In *Cryptology and Network Security.* Springer International Publishing, 2014.

[LEÖM15] Iraklis Leontiadis, Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva. Puda – privacy and unforgeability for data aggregation. In *Cryptology and Network Security*, pages 3–18. Springer International Publishing, 2015.

[LHCH20] Zhaorui Li, Zhicong Huang, Chaochao Chen, and Cheng Hong. Quantification of the leakage in federated learning, 2020.

[LMP11] Y. Li, J. M. McCune, and A. Perrig. Viper: Verifying the integrity of peripherals' firmware. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11. Association for Computing Machinery, 2011.

[LŢ19] Benoît Libert and Radu ŢiŢiu. Multi-client functional encryption for linear functions in the standard model from lwe. In *Advances in Cryptology – ASIACRYPT 2019*, pages 520–551. Springer International Publishing, 2019.

[LWH19] Qinbin Li, Zeyi Wen, and Bingsheng He. Federated learning systems: Vision, hype and reality for data privacy and protection. *CoRR*, abs/1907.09693, 2019.

[MASN21] Ahmed Moustafa, Muhammad Asad, Saima Shaukat, and Alexander Norta. Ppcsa: Partial participation-based compressed and secure aggregation in federated learning. In *Advanced Information Networking and Applications*. Springer International Publishing, 2021.

[Mei02] E. Meijering. A chronology of interpolation: from ancient astronomy to modern signal and image processing. *Proceedings of the IEEE*, 2002.

[MGGA17] S. Moein, T. Aaron Gulliver, F. Gebali, and A. Alkandari. Hardware attack mitigation techniques analysis. *International Journal on Cryptography and Information Security*, 2017.

[MHK+21] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. Ppfl: Privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '21, New York, NY, USA, 2021. Association for Computing Machinery.

[Mit97] Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[MM08] S. Mahfoudh and P. Minet. Survey of energy efficient strategies in wireless ad hoc and sensor networks. In *Seventh International Conference on Networking (icn 2008)*, 2008.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*. PMLR, 2017.

[MRT12] Eoghan McKenna, Ian Richardson, and Murray Thomson. Smart meter data: Balancing consumer privacy concerns with legitimate applications. *Energy Policy*, 41:807–814, 2012. Modeling Transport (Energy) Demand and Policies.

[MSDCS19] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706, 2019.

[MTMH18] Khizir Mahmud, Graham E. Town, Sayidul Morsalin, and M.J. Hossain. Integration of electric vehicles and management in the internet of energy. *Renewable and Sustainable Energy Reviews*, 82:4179–4203, 2018.

[NBM+17] J. Noorman, J. Van Bulck, J. Tobias Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling. Sancus 2.0: A low-cost security architecture for iot devices. *ACM Trans. Priv. Secur.*, 2017.

[NMZ+21] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Michael Rabbat, Mani Malek Esmaeili, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. *CoRR*, abs/2106.06639, 2021.

[NP05] Moni Naor and Benny Pinkas. Computationally secure oblivious transfer. *Journal of Cryptology*, 18, 2005.

[NRY+21] Thien Duc Nguyen, Phillip Rieger, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Ahmad-Reza Sadeghi, Thomas Schneider, and Shaza Zeitouni. FLGUARD: secure and private federated learning. *CoRR*, abs/2101.02281, 2021.

[NS11] Takashi Nishide and Kouichi Sakurai. Distributed paillier cryptosystem without trusted dealer. In Yongwha Chung and Moti Yung, editors, *Information Security Applications*, pages 44–60, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[NSH19] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[Ödé22] Johannes Ödén. Deanonymizing onion services byintroducing packet delay, 2022.

[ÖM07] Melek Önen and Refik Molva. Secure data aggregation with multiple encryption. In *Wireless Sensor Networks*, pages 117–132. Springer Berlin Heidelberg, 2007.

[PAH+18] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13, 2018.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology — EUROCRYPT '99*, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology — CRYPTO '91*. Springer Berlin Heidelberg, 1992.

[PFA21] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. Eluding secure aggregation in federated learning via model inconsistency. *CoRR*, abs/2111.07380, 2021.

[Pol78] J. Pollard. Monte carlo methods for index computation (). *Mathematics of Computation*, 32:918–924, 1978.

[Poo18] Linda Poon. Sleepy in songdo, korea's smartest city, Jun 2018.

[Pro22] Mike Prospero. Best smart home hubs of 2022. https://www.tomsguide.com/us/best-smart-home-hubs,review-3200.html. Retrieved 8 December 2022, October 21, 2022.

[PSJH+20] Ravi Pratap Singh, Mohd Javaid, Abid Haleem, Raju Vaishya, and Shokat Ali. Internet of medical things (IoMT) for orthopaedic in COVID-19 pandemic: Roles, challenges, and applications. *J Clin Orthop Trauma*, 11(4):713–717, May 2020.

[QPG+21] Youyang Qu, Shiva Raj Pokhrel, Sahil Garg, Longxiang Gao, and Yong Xiang. A blockchained federated learning framework for cognitive computing in industry 4.0 networks. *IEEE Transactions on Industrial Informatics*, 17(4):2964–2973, 2021.

[Rab98] Tal Rabin. A simplified approach to threshold and proactive rsa. In *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '98, Berlin, Heidelberg, 1998. Springer-Verlag.

[Riv97] Ronald L. Rivest. All-or-nothing encryption and the package transform. In *Fast Software Encryption*. Springer Berlin Heidelberg, 1997.

[RL89] R.M. Roth and A. Lempel. On mds codes via cauchy matrices. *IEEE Transactions on Information Theory*, 35(6), 1989.

[RMK16] V. Roblek, M. MeÅ¡ko, and A. KrapeÅ¾. A complex view of industry 4.0. *SAGE Open*, (2), 2016.

[RRC04] S. Ravi, A. Raghunathan, and S. Chakradhar. Tamper resistance mechanisms for secure embedded systems. In *17th International Conference on VLSI Design. Proceedings.*, 2004.

[Rub96] Frank Rubin. One-time pad cryptography. *Cryptologia*, 1996.

[RVW+19] M. M. Rabbani, J. Vliegen, J. Winderickx, M. Conti, and N. Mentens. Shela: Scalable heterogeneous layered attestation. *IEEE Internet of Things Journal*, 2019.

[Sab16] Paul Sabanal. Thingbots: The future of botnets in the internet of things, Feb 2016.

[SAG+21] Jinhyun So, Ramy E. Ali, Basak Guler, Jiantao Jiao, and Salman Avestimehr. Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning. *CoRR*, abs/2106.03328, 2021.

[SAGA21] Jinhyun So, Ramy E. Ali, Basak Güler, and Amir Salman Avestimehr. Secure aggregation for buffered asynchronous federated learning. *CoRR*, abs/2110.02177, 2021.

[Sch90] C. P. Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York.

[SCJ18] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[SCR+11] Elaine Shi, T.-H Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. volume 2, 01 2011.

[SG17] K. Sayed and H.A. Gabbar. Chapter 18 - scada and smart energy grid control automation. In Hossam A. Gabbar, editor, *Smart Energy Grid Engineering*, pages 481–514. Academic Press, 2017.

[SGA21a] Jinhyun So, Başak Göler, and A. Salman Avestimehr. Byzantine-resilient secure federated learning. *IEEE Journal on Selected Areas in Communications*, 39, 2021.

[SGA21b] Jinhyun So, Başak Güler, and A. Salman Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, 2, 2021.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 1979.

[Sko11] S. Skorobogatov. Physical attacks on tamper resistance: Progress and lessons. 2nd ARO Special Workshop on HW Assurance, Washington DC, 2011.

[Sko12] S. Skorobogatov. *Physical Attacks and Tamper Resistance*. Springer New York, 2012.

[SLP08] A. Seshadri, M. Luk, and A. Perrig. Sake: Software attestation for key establishment in sensor networks. In *Distributed Computing in Sensor Systems*. Springer Berlin Heidelberg, 2008.

[SLS+05] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05. Association for Computing Machinery, 2005.

[SMH21] Thomas Sandholm, Sayandev Mukherjee, and Bernardo A. Huberman. SAFE: secure aggregation with failover and encryption. *CoRR*, abs/2108.05475, 2021.

[Smi02] Sean W. Smith. Outbound authentication for programmable secure coprocessors. In *Proceedings of the 7th European Symposium on Research in Computer Security*, ESORICS '02. Springer-Verlag, 2002.

[SPvK04] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Swatt: software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, 2004.

[SS11] Abhi Shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. *IACR Cryptol. ePrint Arch.*, 2011:533, 2011.

[SSSS17a] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[SSSS17b] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[SSV+21] Timothy Stevens, Christian Skalka, Christelle Vincent, John Ring, Samuel Clark, and Joseph Near. Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. *CoRR*, abs/2112.06872, 2021.

[STL20] Mohamed Seif, Ravi Tandon, and Ming Li. Wireless federated learning with local differential privacy. In *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020.

[STS16] Shiqi Shen, Shruti Tople, and Prateek Saxena. Auror: Defending against poisoning attacks in collaborative deep learning systems. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16. Association for Computing Machinery, 2016.

[SZJvD04] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04. USENIX Association, 2004.

[TBA+19] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, AISec'19. Association for Computing Machinery, 2019.

[TF19] Aleksei Triastcyn and Boi Faltings. Federated learning with bayesian differential privacy. In *2019 IEEE International Conference on Big Data (Big Data)*, 2019.

[TLB+21] Georgia Tsaloli, Bei Liang, Carlo Brunetta, Gustavo Banegas, and Aikaterini Mitrokotsa. DEVA: Decentralized, verifiable secure aggregation for privacy-preserving learning. In *Information Security*, Cham, 2021. Springer International Publishing.

[TMTQ20] Lo'ai Tawalbeh, Fadi Muheidat, Mais Tawalbeh, and Muhannad Quwaider. Iot privacy and security: Challenges and solutions. *Applied Sciences*, 10(12), 2020.

[TNW+21] Minxue Tang, Xuefei Ning, Yitu Wang, Yu Wang, and Yiran Chen. Fedgp: Correlation-based active client selection for heterogeneous federated learning. abs/2103.13822, 2021.

[Tru23] TrustedFirmware.org. Op-tee documentation. https://optee.readthedocs.io/en/latest/, 2023. [Accessed 08-May-2023].

[VAS19] Thijs Veugen, Thomas Attema, and Gabriele Spini. An implementation of the paillier crypto system with threshold decryption without a trusted dealer. Cryptology ePrint Archive, Report 2019/1136, 2019. https://ia.cr/2019/1136.

[VH08] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[VNP+20] Anamaria Vizitiu, Cosmin Ioan Nită, Andrei Puiu, Constantin Suciu, and Lucian Mihai Itu. Applying deep neural networks over homomorphic encrypted medical data. *Computational and Mathematical Methods in Medicine*, 2020, 2020.

[VXK21] Raj Kiriti Velicheti, Derek Xia, and Oluwasanmi Koyejo. Secure byzantine-robust distributed learning via clustering. *CoRR*, abs/2110.02940, 2021.

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019, 2019.

[WLW+09] Rui Wang, Yong Fuga Li, XiaoFeng Wang, Haixu Tang, and Xiaoyong Zhou. Learning your identity and disease from research papers: Information leaks in genome wide association study. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09. Association for Computing Machinery, 2009.

[WPX+20] Danye Wu, Miao Pan, Zhiwei Xu, Yujun Zhang, and Zhu Han. Towards efficient secure aggregation for model update in federated learning. In *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020.

[WTB+20] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. FALCON: honest-majority maliciously secure framework for private deep learning. *CoRR*, abs/2004.02229, 2020.

[XBZ+19] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. Hybridalpha: An efficient approach for privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, AISec'19. Association for Computing Machinery, 2019.

[XHCL20] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2020.

[XLL+20] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. Verifynet: Secure and verifiable federated learning. *IEEE Transactions on Information Forensics and Security*, 15, 2020.

[YAE+18] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *CoRR*, abs/1812.02903, 2018.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.

[YLCT19] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10, 2019.

[YRSA18] Qian Yu, Netanel Raviv, Jinhyun So, and Amir Salman Avestimehr. Lagrange coded computing: Optimal design for resiliency, security and privacy. *CoRR*, abs/1806.00939, 2018.

[YSH+21] Chien-Sheng Yang, Jinhyun So, Chaoyang He, Songze Li, Qian Yu, and Salman Avestimehr. Lightsecagg: Rethinking secure aggregation in federated learning. *CoRR*, abs/2109.14236, 2021.

[YSW18] Chen Yang, Weiming Shen, and Xianbin Wang. The internet of things in manufacturing: Key issues and potential applications. *IEEE Systems, Man, and Cybernetics Magazine*, 4(1):6–15, 2018.

[YWHZ18] Hailong Yao, Caifen Wang, Bo Hai, and Shiqiang Zhu. Homomorphic hash and blockchain based authentication key exchange protocol for strangers. In *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, pages 243–248, 2018.

[YWW21] Ge Yang, Shaowei Wang, and Haijie Wang. Federated learning with personalized local differential privacy. In *2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*, 2021.

[ZDA+17] S. Zeitouni, G. Dessouky, O. Arias, D. Sullivan, A. Ibrahim, Y. Jin, and A. Sadeghi. Atrium: Runtime attestation resilient under memory attacks. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017.

[ZFW+20] Xianglong Zhang, Anmin Fu, Huaqun Wang, Chunyi Zhou, and Zhenzhu Chen. A privacy-preserving and verifiable federated learning scheme. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020.

[ZJF+21] Lingchen Zhao, Jianlin Jiang, Bo Feng, Qian Wang, Chao Shen, and Qi Li. Sear: Secure and efficient aggregation for byzantine-robust federated learning. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2021.

[ZLH19] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[ZLX+20] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 2020.

[ZLYM21] Zhuosheng Zhang, Jiarui Li, Shucheng Yu, and Christian Makaya. Safelearning: Enable backdoor detectability in federated learning with secure aggregation. *CoRR*, abs/2102.02402, 2021.

[ZM21] Wensheng Zhang and Trent Muhr. Tee-based selective testing of local workers in federated learning systems. In *2021 18th International Conference on Privacy, Security and Trust (PST)*, pages 1–6, 2021.

[ZWC+21] Yuhui Zhang, Zhiwei Wang, Jiangfeng Cao, Rui Hou, and Dan Meng. Shufflefl: Gradient-preserving federated learning using trusted execution environment. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*, CF '21, New York, NY, USA, 2021. Association for Computing Machinery.