Karlsruhe Institute of Technology

KASTEL

Privacy-Preserving Aggregation and Evaluation of Personally Identifiable Information from Private Mobile Systems

Master's Thesis of

Alexander von Heyden

At the KIT Department of Informatics
KASTEL – Institute of Information Security and Dependability

First examiner:	Prof. Dr. Jörn Müller-Quade
Second examiner:	Prof. Dr. Thorsten Strufe
First advisor:	M.Sc. Felix Dörre
Second advisor:	M.Sc. Valerie Fetzer

03. June 2024 – 14. February 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, 14.02.2025

. . . . . .

(Alexander von Heyden)

Abstract

Privacy-preserving data aggregation is the practice of collecting and aggregating user data while still guaranteeing their privacy, which we increasingly rely on in an era where access to personal data becomes more and more valuable. Secure aggregation requires that no adversary, even one with significant capabilities, can infer private user information.

We categorise existing approaches into three distinct architectures: *Client-Server architectures*, which rely on a central server; *(Client Cluster)-Server architectures*, which enhance privacy by inter-client communication before submitting data to a server; and *Client-(Distributed Server) architectures*, which distribute the aggregation across multiple servers, requiring only one honest server to maintain privacy. For each approach we analyse and compare key properties. Building on this analysis, we find that real-world scenarios exist, where current approaches fall short and can be improved upon to provide similar privacy guarantees while requiring less initial trust.

We propose a novel user-trust-based approach to privacy-preserving data aggregation which ensures, that user data remains secure, even against adversaries with control over all aggregation servers. By combining advanced cryptographic primitives - such as threshold and verifiable secret sharing schemes - with results from coding theory we can prevent dishonest clients from manipulating the aggregation result. Although our approach increases inter-user communication, it reduces the trust reliance on central servers while maintaining strong privacy guarantees.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronymes

CPA Chosen-plaintext attack

CPA2 Adaptive chosen-plaintext attack

CCA Chosen-ciphertext attack

IND-CPA Indistinguishability under chosen-plaintext attack

DCR decisional composite residuosity

DCRA decisional composite residuosity assumption

CSA Client-Server Architecture

CcSA (Client Cluster)-Server Architecture

CDsA Client-(Distributed Server) Architecture

SNIP secret-shared non-interactive proof

NIZK non-interactive zero-knowledge

SNARK succinct non-interactive argument of knowledge

1. Introduction

In our increasingly connected lives, data has become a cornerstone for our technological advancement. We use data to make innovations, drive decision-making processes across a broad spectrum of fields, including healthcare, finances and artificial intelligence on any scale imaginable. However, the collection, aggregation and evaluation of such data comes with a significant risk to our privacy. When exposed and misused, personal and sensitive information can lead to severe consequences, such as user profiling. User profiles enable discrimination against users based on their previous and predicted behaviour, such as gouging of goods, services or commodities. Staying ahead of these challenges requires the development of data aggregation schemes, that preserve our privacy while still providing the necessary aggregation of data to continue making new innovations and technological improvements.

This thesis aims to evaluate existing and introduce new privacy-preserving aggregation schemes that all aim to strike this delicate balance between generating meaningful insight into the aggregated data of individuals without leaking any sensitive data in the process. We examine schemes using only basic building blocks up to complex schemes that make use of advances cryptographic primitives. The problem at hand is complex and multifaceted and no known solution fits all the requirements at once. This requires us to take a close look at each use case individually to determine how to best approach it.

1.1. Motivation

Centralized aggregation schemes handling raw data often requires placing trust in entities that may not have the data owner's best interest in mind. Whether this is due to malicious intent, data breaches, or technical shortcomings, an aggregation scheme handling raw data always places this data at risk.

In recent years many legislative bodies have introduced regulations like the General Data Protection Regulation (GDPR) [20] from the European Union or the California Consumer Privacy Act (CCPA) [11] that aim to increase user privacy by mounting pressure on organisations to comply with increasingly stringent privacy standards. At the same time more organisations than ever continue competing to unlock new value behind user generated data. These inherent conflicts underscore the dire need for privacy-preserving solutions to data aggregation.

1.2. Contribution

This thesis makes several contributions to the topic of privacy-preserving data aggregation. It starts by introducing and explaining different existing approaches. We then notice, that we can describe all existing approaches using three main types of architectures, even though the

approaches themselves might be entirely unique. We outline each architecture and classify the existing schemes accordingly. Using common scenarios this thesis identifies if and where these existing approaches fail to meet certain requirements and provides improved approaches to counteract these shortcomings. As a result we create a new approach that performs better for some real-world applications. Building on these approaches it also examines and evaluates possible approaches to output privacy in connection with privacy-preserving aggregation schemes, further solidifying the privacy guarantees by the schemes.

1.3. Structure of this Thesis

The thesis is structured as follows:

Preliminaries Understanding the mathematical and cryptographic foundations is key to successfully construct privacy-preserving data aggregation schemes. Chapter two introduces these foundations in two stages. In the first half common definitions are established and illustrated where needed, whereas the second half lists the building blocks that make use or implement these definitions. These building blocks will later allow us to construct privacy-preserving aggregation schemes.

Existing Approaches Chapter three describes the distinct architectures of privacy-preserving aggregation schemes and introduces existing schemes for each of the architectures. It presents the assumptions needed and the guarantees given by the schemes and evaluates the schemes based on the metrics privacy and robustness.

Improved Approaches Building on the previous chapter, chapter four describes new and improved approaches to the aggregation problem. It takes the existing approaches, identifies problems and presents solutions to improve the schemes with regards to the identified problems.

Evaluation To evaluate how all the presented privacy-preserving data aggregation schemes hold up in the real-world, different scenarios are outlined, each providing unique challenges that need to be overcome. The aggregation schemes are compared based on how well they can be applied to each scenario, based on the scheme's guarantees and how difficult it is to meet the required assumptions. Concluding this chapter is an overview over general limitations of privacy-preserving aggregation schemes, meaning weaknesses that would occur in an ideal scheme but which are still undesirable for real-world applications.

2. Preliminaries

This chapter covers the required preliminaries necessary for this thesis. In the first half of this chapter proper definitions of relevant security terms and concepts used throughout this thesis will be given. The second half introduces building blocks adhering to these definitions, that will allow us to construct more advanced cryptographic primitives.

2.1. Formal Definitions

With exact definitions varying between different publications, and not necessarily being equivalent, it is important to unambiguously outline the definitions that this thesis will follow. This section also introduces various cryptographic terms that are required to better understand the following chapters of this thesis.

2.1.1. Cryptosystems

Most public-key encryption schemes rely on the assumed difficulty of specific mathematical problems to prove their security. Common examples include the *integer factorisation problem* for the RSA cryptosystem or the *discrete logarithm problem* for the ElGamal cryptosystem. In this thesis we focus on a cryptosystem based on the decisional composite residuosity assumption (DCRA) to make use of some of its interesting properties:

Definition 1 (Decisional Composite Residuosity [44])**.** Let $n \in \mathbb{N}$ be a product of two large primes and $z \in \mathbb{N}$ an integer. We say that z is an n-th residue modulo n^2 if an invertible element (unit) $y \in \mathbb{Z}^*_{n^2}$ exists such that

$$z \equiv y^n \bmod n^2.$$

Definition 2 (Decisional Composite Residuosity Assumption[42, 44])**.** The assumption that no probabilistic polynomial time distinguisher exists for n-th residues modulo n^2, that is it is hard to decide whether z is an n-residue modulo n^2, is called the *decisional composite residuosity assumption* (DCRA).

To make use of this assumption we also require a framework in which to use it. This framework will be an asymmetric public-key encryption scheme.

Definition 3 (Public-Key Encryption [29])**.** A *public-key encryption scheme* is a set of three probabilistic polynomial algorithms (*Gen*, *Enc*, *Dec*) with the following functionality:

Gen The *Gen algorithm* takes as input the security parameter 1^κ and generates a key pair (pk, sk). The first entry is usually referred to as *public key*, the second as *secret* or *private key*. For the sake of convenience we also assume that pk and sk are both of length at least κ, and that we can determine κ from pk, sk.

Enc The *Enc algorithm* takes as input a public key pk and a message m from some message space $\mathcal{M}$, which may depend on pk. It outputs a ciphertext c.

Dec The *Dec algorithm* is a deterministic algorithm that takes as input a secret key sk and a ciphertext c, and outputs the corresponding message m or the special symbol $\perp$ ("bottom") to indicate failure.

We also require correctness for a public-key encryption scheme, that is $\mathrm{Dec}_{sk}(\mathrm{Enc}_{pk}(m)) = m$ for any key pair (pk, sk) generated by $\mathrm{Gen}(1^\kappa)$ and for any (legal) message m, except with possibly negligible probability.

Since it is not possible for a public-key encryption scheme using a deterministic encryption algorithm to even achieve Chosen-plaintext attack (CPA) security [29], only probabilistic encryption algorithms are considered in most cases. This is the reason why we usually write $c \leftarrow \mathrm{Enc}_{pk}(m)$ to denote encryption and simply $m := \mathrm{Dec}_{sk}(c)$ for decryption.

The encrypted messages of an encryption scheme generally don't allow any meaningful form of processing without previous decryption. While this restriction might be completely irrelevant for some use cases, or even desirable, it turns out to be of considerable use to be able to perform at least some mathematical operations on the encrypted values. This is exactly what homomorphic encryption[1] allows us to do:

Definition 4 (Homomorphic Encryption [41])**.** A homomorphic encryption scheme with message space $\mathcal{M}$, ciphertext space C and functions $f : \mathcal{M}^w \to \mathcal{M} \in \mathcal{F} \subseteq \bigcup_{w \geq 0} \{ f : \mathcal{M}^w \to \mathcal{M} \}$ is an encryption scheme $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ with $\mathcal{M}$ together with an evaluation algorithm *Eval* that on input a public key pk, a function $f : \mathcal{M}^w \to \mathcal{M} \in \mathcal{F}$, and a sequence $c \in C^w$, outputs a ciphertext $\mathrm{Eval}(pk, f, c) \in C$.

For a homomorphic encryption scheme to be considered correct, we require that for any function $f : \mathcal{M}^w \to \mathcal{M}$ encrypting some data ($c = \mathrm{Enc}(pk, m)$), evaluating it using f ($c' = \mathrm{Eval}(pk, f, c)$) and decrypting it ($m' = \mathrm{Dec}(sk, c')$) yields the same result as directly computing $f(m)$.

Definition 5 (Homomorphic Correctness [41])**.** An encryption scheme $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Eval})$ is *$\mathcal{F}$-homomorphic* if for any keys $(sk, pk) \leftarrow \mathrm{Gen}(\kappa)$, function $f : \mathcal{M}^w \to \mathcal{M} \in \mathcal{F}$, and messages $m \in \mathcal{M}^w$, we have $\mathrm{Dec}(sk, \mathrm{Eval}(pk, f, \mathrm{Enc}(pk, m))) = f(m)$.

Functions f are represented using an encoding, for which we use boolean circuits.

[1] Derived from the Greek words *homos*, meaning "same" and *morphe*, meaning "shape" or "form". A homomorphism is thus a structure-preserving mapping (function) between two spaces (the plaintext space and ciphertext space).

2.1.2. Secret Sharing

Secret sharing revolves around the idea of sharing a secret with multiple parties in such a way, that the original secret remains private. If done correctly no knowledge of any individual secret share allows a party to gain any information regarding the shared secret, only when enough parties work come together is it possible to reconstruct the secret. Both computationally hard and information-theoretic secure secret sharing schemes exist, but given that the information-theoretic schemes are efficient for privacy-preserving data aggregation, our definition of secret sharing schemes will include information-theoretic security by default.

The simplest example of a secret sharing scheme is one that splits the secret into n pieces, called shares, and requires all n shares to reconstruct the original secret.

We will refrain from providing a definition for this type of secret sharing scheme and instead provide a more general definition, namely (t, n)-threshold secret sharing. These type of schemes create n shares but only require any arbitrary subset of t shares to reconstruct the secret, while any subset of $t - 1$ shares yields no information about the secret.

Even going back half a century we already find the following question asked in [59]:

> *Eleven scientists are working on a secret project. They wish to lock up the documents in a cabinet so that the cabinet can be opened if and only if six or more of the scientists are present. What is the smallest number of locks needed? What is the smallest number of keys to the locks each scientist must carry?*

While finding the solution is not too difficult in and of itself (it requires 462 locks and 252 keys per scientist), it shows just how fast the trivial approach becomes unfeasible, even with small numbers. Problems such as this are one of the main motivations behind threshold secret sharing.

Definition 6 ((t, n)-Threshold Secret Sharing [50])**.** Let $\mathbb{F}$ be a finite field, $l \in \mathbb{N}$ and $x \in \mathbb{F}^l$ a secret element. Let $n > 0$ denote the total number of shares generated by the scheme and $0 < t \leq n$ the number of shares required to reconstruct x. A (t, n)-*threshold secret sharing scheme* consists of two probabilistic polynomial algorithms, one for generating shares (*Share*) and one for reconstructing the secret from a set of t shares, which have the following functionality:

- **Share(**x**)** $\rightarrow \{ [x]_1, \ldots, [x]_n \}$
  The *share algorithm* takes as input a secret x and outputs n shares $\{ [x]_1, \ldots, [x]_n \} \in (\mathbb{F}^l)^n$.

- **Reconstruct(**$S \subseteq \{ [x]_1, \ldots, [x]_n \}$**)** $\rightarrow x$**,** $|S| = t$
  The *reconstruct algorithm* takes as input a subset of t shares $S \subseteq \{[x]_1, \ldots, [x]_n\}$ and uses them to compute and output an element $y \in \mathbb{F}^l$.

For a (t, n)-threshold secret sharing scheme to be of any real use we require it to have certain properties:

Correctness We require that for all valid inputs x sharing and reconstructing t shares returns the original x. That is

$$\forall S_t \in \binom{\text{Share}(x)}{t} : \text{Reconstruct}(S_t) = x,$$

where $\binom{A}{t}$ represents the set of all subsets of A of size t.

Efficiency Knowledge of any t shares makes x efficiently computable.

Information-theoretic secure Knowledge of any $t-1$ or fewer shares reveals no information about x, even if the adversary is computational unbound.

While not particularly useful, this definition also permits $(0, n)$-threshold schemes, where the secret needs to be known to the Reconstruct algorithm in advance, and $(1, n)$-threshold schemes. We often imply (t, n) and only write *threshold secret sharing* if the parameters are given by the context or not relevant.

While using threshold secret sharing schemes makes it possible to reconstruct the secret, even when some parties holding shares fail to participate, it also decreases the number of malicious parties required to unveil a secret. If only t parties are required to correctly perform the intended protocol using (t, n)-threshold secret sharing, t malicious parties are also enough to derive the secret value. If we employ threshold secret sharing schemes we must always consider a fine balance between functionality and privacy when selecting t.

Let us now consider some properties of secret sharing schemes:

Definition 7 (Linear Secret Sharing [45])**.** Let S be a (t, n)-threshold secret sharing scheme, with $0 < t \leq n$, and $x, y \in \mathbb{F}^l$ secrets, each split into n share $\{[x]_1, \ldots, [x]_n\}$, $\{[y]_1, \ldots, [y]_n\}$. Let $J \subseteq \{1, \ldots, n\}, |J| \geq k$ be an index set. We say S is a linear secret sharing scheme (over $\mathbb{F}^l$) if $\{[x]_1 + [y]_1, \ldots, [x]_n + [y]_n\}$ are valid shares of $(x + y)$, that is

$$\textbf{Reconstruct}(\{[x]_j + [y_j]\}_{j \in J}) = (x + y)$$

for any valid subset J.

This property allows us to combine multiple secrets into one single secret, which can only be unveiled if enough share holding parties decide to work together to reconstruct it. Linear secret sharing will be one of the first ideas we take a closer look at to achieve privacy-preserving data aggregation later on in chapter 3.

Definition 8 (Verifiable Secret Sharing [22])**.** Let S be a (t, n)-threshold secret sharing scheme over $\mathbb{F}$ and *Check*, *Encrypt* two PPTA algorithms. We say S together with *Check* and *Encrypt* is a *verifiable secret sharing* scheme if it meets the following requirements:

Verifiability constraint Upon receiving a share of the secret, any party must be able to verify whether or not it is a valid share. If a share is valid, there exists a unique secret which will be output by the *Reconstruct* algorithm when it is run on any t distinct valid pieces.

Unpredictability There is no polynomial-time strategy for picking $u < t$ pieces of the secret, such that they can be used to predict the secret with more than negligible probability.

2.1.3. Indistinguishability

If a scheme is secure against even an adversary with an unlimited amount of computing resources and time, we call it information-theoretic secure. While Shannon originally introduced this term to proof that the one-time pad [56] system was secure [51], we can also apply this definition to other schemes that encrypt data or try to make data inaccessible by other means:

Definition 9 (Information-theoretic Security (Encryption) [15, 37])**.** Let E be an encryption system. Let M be a message and C a ciphertext generated by encrypting M using E. We say E is *information-theoretic secure* if M and C are *statistically* independent.

Definition 10 (Information-theoretic Security (Secret Sharing))**.** Let S be a (t, n)-threshold secret sharing scheme. Let D be a secret and $\{ C_1, \ldots, C_n \}$ the set of shares generated by secret sharing D using S. Let $I \subset \{ 1, \ldots, n \}, |I| < t$ be an index set. We say S is *information-theoretic secure* if D and $\{ C_i \mid i \in I \}$ are *statistically* independent for all D and all index sets I.

This is in contrast to computational hardness assumptions, where we only require that no effective algorithm (usually meaning that it runs in *polynomial time*) can derive any meaningful relation between the in- and output of a scheme (except with negligible probability), while an unbound adversary might still be able to statistically relate in- and output. To properly define computational indistinguishability we model these in- and output using ensembles:

Definition 11 (Probability Ensemble [24])**.** Let I be a countable index set. An ensemble indexed by I is a sequence of random variables indexed by I. Namely, any $X = \{ X_i \}_{i \in I}$, where each X_i is a random variable, is an ensemble indexed by I.

Definition 12 (Computational Indistinguishability [24])**.** Two ensembles, $X = X_{nn\in\mathbb{N}}$ and $Y = Y_{nn\in\mathbb{N}}$, are indistinguishable in polynomial time if for every probabilistic polynomial-time algorithm D, every positive polynomial $p(\cdot)$, and all sufficiently large n's,

$$|Pr[D(X_n, 1^n) = 1] - Pr[D(Y_n, 1^n) = 1]| < p(n)$$

While information-theoretic security might be a desirable goal for some use cases, using it as our only reference for what a "secure" scheme should be severely limits what we can achieve with cryptography. For the most relevant applications it suffices to consider real-world threats and omit the possibilities of attackers or events that are too unlikely to exist or occur. It thus has become common to separate between polynomial and super-polynomial attackers as well as inverse-polynomial and inverse-super-polynomial probabilities. We consider an attacker that can break a given scheme with polynomial runtime $p(n)$ a threat in the same way, as we consider a scheme that can be broken by an adversary with probability $1/p(n)$ not secure, for some (positive) polynomial p. If any attacker's runtime is asymptotically lager than $p(n)$, or the adversary's probability is asymptotically smaller than $1/p(n)$, we consider the scheme to be asymptotically secure. In these cases we consider the adversary's chances of success to be negligible:

Definition 13 (Negligible Functions [29, 24])**.** A function f is *negligible* if for every polynomial $p(\cdot)$ there exists an N such that for all integers $n > N$ it holds that $f(n) < 1/p(n)$.

We usually write negl to denote an arbitrary negligible function.

2.1.4. Zero-Knowledge

The intuitive idea behind zero-knowledge proofs is that one party (the prover) wants to convince a different party (the verifier), that some statement they make is true. This could relate to proving knowledge of some secret information, without leaking information about the secret information itself. While it is relatively easy for the prover to convince the verifier of their knowledge by simply revealing the secret, the difficulty arises by trying to do so without revealing it.

To understand what a zero-knowledge proof is, we need to first define the zero-knowledge property itself. Since we do not require the exact mathematical details we will only provide a shorter, informal definition:

Definition 14 (Zero-Knowledge Property [40])**.** Let P be a prover and V a verifier interacting to convince V that some statement S is true. During their interaction V may not learn anything other than the validity of S. For any third party it is computationally (statistically) indistinguishable whether they have witnessed a real interaction where P knows that S is true, or a simulated interaction replicating a possible interaction using only V's knowledge.

Definition 15 (Zero-Knowledge Proof [40])**.** Let P be a prover and V a verifier interacting to convince V that some statement S is true using a protocol. We say that this protocol is a zero-knowledge proof if it satisfies three properties:

Completeness If S is true an honest prover P will always convince an honest verifier V of this fact.

Soundness If S is false, then no (dishonest) prover P can convince an honest verifier that S is true, except with a small probability.

Zero-knowledge It satisfies the *zero-knowledge property* defined above.

The *soundness* property of zero-knowledge proofs permits that even a dishonest prover can convince an honest verifier with some small probability, since even a prover without the secret knowledge still has a small chance of success. To counteract this issue it is common practice to sequentially repeat a zero-knowledge proof multiple times. While an honest prover can use its knowledge to successfully convince an honest verifier as often as required, a dishonest prover repeatedly relying on the soundness error will most certainly be found out after enough iterations have passes.

Definition 16 (secret-shared non-interactive proof (SNIP) [12])**.** Let $\mathbb{F}$ be a finite field, $x \in \mathbb{F}^l$ and n be equal to the number of secret shares, secret proof shares as well as the number of verifiers. Let $\text{Valid} : \mathbb{F}^l \to \mathbb{F}$ be a predicate such that $\text{Valid}(x) = 1$ if x is a valid input. Let $[\tilde{x}]_i$ denote the ith share of the proof of $\text{Valid}(x) = 1$. A SNIP consists of two probabilistic polynomial algorithms, one for generating the proof shares (*Prove*) and one for verifying that $\text{Valid}(x) = 1$ (*Verify*), which have the following functionality:

Prove The *Prove algorithm* takes as input a predicate $\text{Valid}(\cdot)$ and a secret x. Without communicating with any other party it outputs a list of secret proof shares $\{[\tilde{x}]_1, \ldots, [\tilde{x}]_n\}$.

Verify The *Verify algorithm* is a distributed consensus algorithm, running on n nodes simultaneously, which takes a predicate $\text{Valid}(\cdot)$, and list of secret proof shares $\{[\tilde{x}]_1, \ldots, [\tilde{x}]_n\}$, which are distributed so that each node receives exactly one unique proof share as input. The nodes may communicate amongst themselves and output 1 if and only if they are convinced that $\text{Valid}(x) = 1$.

The SNIP protocols we make use of in this thesis share most of their properties with generic zero-knowledge interactive proof systems [26]. If not otherwise stated the SNIPs used in this thesis hold the following properties:

Correctness For *correctness* we require that for all valid inputs x and predicates $\text{Valid}(\cdot)$:

$$\text{Verify}_{\text{Valid}(\cdot)}(\text{Prove}_{\text{Valid}(\cdot)}(x)) = 1.$$

That is if the secret proof shares $\{[\tilde{x}]_1, \ldots, [\tilde{x}]_n\}$ were generated by an honest proof algorithm, an honest verify algorithm will always be convinced that $\text{Valid}(x) = 1$.

Soundness For *soundness* we require that the verify algorithm rejects x with overwhelming probability if $\text{Valid}(x) \neq 1$, that is the probability of a false positive is negligible, even if it is chosen by an attacker with super-polynomial runtime:

$$\forall x \in \mathbb{F}^l, \text{Valid}(x) \neq 1 : \Pr[\text{Verify}_{\text{Valid}(\cdot)}(\text{Proof}_{\text{Valid}(\cdot)}(x)) = 1] \leq \text{negl}$$

Zero Knowledge The zero-knowledge property (Definition 14) holds.

The main difference between traditional zero-knowledge proof systems and SNIPs arises from the fact that the latter allow for a single prover to convince multiple verifiers from its knowledge, whereas a traditional zero-knowledge proof systems assume a single prover interacting with a single verifier.

2.1.5. Multi-Party Computation

Definition 17 (Arithmetic Circuit [12])**.** An *arithmetic circuit* C over a finite field $\mathbb{F}$ takes as input a vector $x = \langle x^{(1)}, \ldots, x^{(L)} \rangle \in \mathbb{F}^l$ and produces a single field element as output. We represent the circuit as a directed acyclic graph, in which each vertex is either an *input*, a *gate*, or an *output* vertex.

Input vertices have in-degree zero and are labelled with a variable in $\left\{ x^{(1)}, \ldots, x^{(l)} \right\}$ or a constant in $\mathbb{F}$. Gate vertices have in-degree two and are labelled with the operation + or $\times$. The circuit has a single output vertex, which has out-degree zero.

To compute the circuit $C(x) = C(x^1, \ldots, x^{(l)})$, we walk through the circuit from inputs to outputs, assigning a value in $\mathbb{F}$ to each wire until we have a value on the output wire, which is the value of $C(x)$. In this way, the circuit implements a mapping $C : \mathbb{F}^l \to \mathbb{F}$.

In a setting where a number of participants each some piece of data and want to compute a result based on all of their data multiple approaches exist. If there is no requirement for the pieces of data do remain secret, the participants could simply reveal their data to the others and compute the result in public. Computing the result becomes more difficult, if the data of each participant should remain private. Depending on the exact requirements different approaches and solutions exist and secure multi-party computation is one of them. Secure multi-party computation protocols aim to provide two things:

Input privacy No information about any party's private input can be inferred from the execution of the protocol, apart from what can be inferred from the output of the modelled function alone.

Correctness All honest parties receive the correct output of the computation, as if it were computed on the inputs of all parties by a trusted third party.

In the context of multiparty computation multiple different parties need to work together to compute the result. While all participating parties should share the common goal to arrive at the correct result, any given party might not behave honestly during execution. The literature usually classifies the different types of adversaries into three distinct classes with unique behaviour and goals.

Definition 18 (Semi-Honest (Passive) Adversary [3])**.** Semi-honest adversaries can only manipulate parties to misbehave in passive ways. A party manipulated in such a way follows the intended execution of the protocol, but tries to learn more information than intended, such as by analysing the messages sent between other parties.

Semi-honest adversarial behaviour tries to model information leakage in a system and is not really a suitable model for most real-world applications [3].

Definition 19 (Malicious (Active) Adversary [3])**.** Malicious adversaries can manipulate parties to misbehave both, passively and actively. Parties acting actively malicious are not bound by the rules of the protocol. They may actively try to manipulate other parties and gain information by behaving in a way not intended by the protocol.

Protocols secure against malicious adversaries provide strong security guarantees, as the participating honest parties can be sure that their data is protected, even in the face of actively malicious parties. While it has been shown that, given the correct cryptographic assumptions, any probabilistic polynomial-time multiparty functionality can be securely computed for any number of malicious adversaries [25, 43], these guarantees come at a substantial price. Such constructions are usually not efficient enough to be considered for practical real-world usage, which is why Aumann and Lindell have introduced a third type of adversary that can be placed between semi-honest and malicious adversaries.

Definition 20 (Covert Adversary Model [3])**.** Let $0 < \epsilon \leq 1$ be the deterrence factor. Then every attempt at cheating is detected by the honest parties with probability at least ϵ.

While this definition does not guarantee that any given party does not cheat, it does guarantee that any cheating party will be discovered with probability of at least ϵ. A malicious party can still learn private information without being noticed with probability $1 - \epsilon$. In certain settings a sufficiently large ϵ will motivate potential malicious parties to refrain from cheating, especially in settings where cheating is directly connected with unavoidable repercussions.

2.1.6. Output Privacy

Even given an aggregation scheme that guarantees that the participating parties learn no more than what is revealed at the end of the computation, a user's privacy might still be at risk. Multiparty protocols only provide input secrecy, meaning that participating parties can jointly compute a function without revealing their secret input. However, any (third) party that has access to the aggregation result can use this knowledge to possibly infer information about a participating user. This is especially the case in situations where the aggregation scheme yields exact results. Published results of these schemes might be susceptible to simple linkage and reconstruction attacks and in case of machine learning applications also to model inversion attacks . Differential privacy aims to protect users from these kinds of attacks by ensuring that the output does not reveal too much about an individual user's input, which can be achieved by including fine-tuned noise in the output. Differential privacy is already widely used by different entities, such as Google [60], Apple [53] and the US Census Bureau [13].

Definition 21 (Differential Privacy [19, 18, 31])**.** A randomized algorithm $M : D^n \rightarrow \mathcal{Y}$ is (ϵ, δ)-differentially private if for any two neighbouring databases $X, X' \in D^n$, and all sets $T \subseteq \mathcal{Y}$, $Pr[M(X) \in T] \leq e^{\epsilon} \cdot Pr[M(X') \in T] + \delta$.

Definition 22 (Gaussian Mechanism [17, 31])**.** Given any function $f : D^n \rightarrow \mathbb{R}^k$ with ℓ_2-sensitivity $\Delta_2(f)$ and privacy parameters (ϵ, δ), the Gaussian mechanism is defined as:

$$M_{Gauss}(x) = f(X) + (Y_1, \ldots, Y_k),$$

where Y_i are independent and identically distributed random variables drawn from a Gaussian distribution $\mathcal{N}(\mu = 0, \sigma^2)$ with probability density function $P(x|\mu = 0, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{(2\sigma^2)}}$ and $\sigma^2 > \frac{2\ln(\frac{1.25}{\delta}) \cdot (\Delta_2(f))^2}{\epsilon^2}$.

2.1.7. Data-Aggregation Scheme

Before concluding this section we want to finally give an exact definition as to what we actually mean with *Data-Aggregation Scheme*. As stated previously the goal of this thesis is the privacy-preserving aggregation and evaluation of personally identifiable information, which requires us to pay careful attention to the following definition, so that all schemes presented afterwards can share a common baseline.

Definition 23 (Data-Aggregation Scheme)**.** Let $f : \mathcal{D}^l \rightarrow \mathcal{E}$ be a function that maps from an l-dimensional input-space $\mathcal{D}^l$ to an output-space $\mathcal{E}$. The easiest *data-aggregation scheme* is a set of two probabilistic polynomial protocols (*Submit*, *Aggregate*) running independently from each other. Any node in an aggregation scheme is either wishing to submit data to be included in the aggregation (client nodes), or is (to some degree) responsible for the aggregation (server nodes). A data-aggregation scheme may require some nodes to act as client and server simultaneously. Each protocol has the following properties:

Submit The *Submit protocol* runs on all client nodes that wish to submit their data for aggregation. Each client node takes an element x of input-space $\mathcal{D}$ and a function f as input. It then propagates from x derived information through the network to one or more server nodes. Which information is propagated and how it is derived from x may depend on f. While participating in this protocol client nodes may communicate with other client nodes.

Aggregate The *Aggregate protocol* runs on all server nodes. Each server node is initialised with the identical, not necessarily secret function f and takes all the propagated values it has received from participating clients as input. The server nodes may then work together to compute $f(x_1, \ldots, x_n)$, where n is the number of client nodes and x_i is the secret value submitted by client node i

We usually simply refer to server nodes as servers, and to client nodes as clients. To allow for some flexibility in the later chapters, mainly regarding different possible infrastructure layouts of the network, the definition is kept somewhat informal.

For a data-aggregation scheme to be considered useful in the context of thesis, we usually want it possess some key properties, which include privacy, anonymity, differential-privacy, correctness and robustness, which are defined in the following section.

2.1.8. Privacy and Robustness

To formally state our desired notion of privacy we can employ a simulation based definition:

Definition 24 (f-Privacy [12])**.** Let k be the number of servers and n be the number of clients in an aggregation scheme. We say that an aggregation scheme provides *f-privacy* for a function f, if for:

- every subset of at most $k - 1$ servers, and
- every subset of at most n clients,

there exists an efficient simulator, that for every choice of client inputs $(x_1, \ldots, x_n)$, takes as input:

- the public parameters to the protocol run (all participants' public keys, the description of the aggregation function f, the cryptographic parameters, etc.),
- the indices of the adversarial clients and servers,
- oracle access to the adversarial participants, and
- the value $f(x_1, \ldots, x_n)$,

and outputs a simulation of the adversarial participant's view of the protocol run whose distribution is computationally indistinguishable from the distribution of the adversary's view of the read protocol run.

This definition remains somewhat informal, especially what exactly the oracle access entails. If will, however, be sufficient for our purposes.

For our definition of anonymity we want to prevent an adversary from learning which client submitted which data point. An aggregation function that simply takes all the data points and returns them lexicographically sorted would meet these requirements, since after being sorted it is no longer possible for any adversary to map a data point to a client. To define anonymity we make use of this very example:

Let `SORT` be the function that takes n inputs and outputs them in lexicographically increasing order.

Definition 25 (Anonymity [12])**.** We say that a data-aggregation scheme provides *anonymity* if it provides f-privacy, in the sense of definition 24, for f = `SORT`.

A scheme providing this form of anonymity allows the potential disclosure of the entire list of client inputs $(x_1, \ldots, x_n)$ without an adversary gaining any information on which client submitted which x_i.

Note the difference between *anonymity* and *differential privacy*. While both remove information from the aggregate output on purpose, they follow different ideas in doing so. If we say that differential privacy hides **which** exact data points were submitted, anonymity hides **who** submitted these data points. In this case anonymity does not prevent an adversary from making precise deductions over a **group** of users.

With the definitions for f-privacy and anonymity in place we can now use the algebraic concept of symmetric functions to further relate both definitions. Remember the definition of a symmetric function:

Definition 26 (Symmetric Function [33])**.** Let $\mathbb{F}$ be a finite field and $x_1, \ldots, x_n \in \mathbb{F}$ be a set of n variables in $\mathbb{F}$. A function f in the variables $x_1, \ldots, x_n$ is a symmetric function if it is an absolute invariant of the symmetric group $\mathfrak{S}_n$; that is, for all permutations γ of $\{1, \ldots, n\}$,

$$f(x_{\gamma(1)}, \ldots, x_{\gamma(n)}) = f(x_1, \ldots, x_n).$$

Lemma 1 (f-privacy and symmetry [12])**.** *Let $\mathcal{D}$ be a data-aggregation scheme that provides f-privacy, in the sense of definition 24, for a symmetric function f. Then $\mathcal{D}$ provides anonymity.*

Lemma 2 (Non-symmetric anonymous data collection schemes [12])**.** *Let f be a non-symmetric function. Then there is no anonymous data collection scheme that correctly computes f.*

Proof sketches for Lemma 1 and Lemma 2 can be found in [12].

Corrigan-Gibbs and Boneh note that a data collection scheme that provides both *f-privacy* and uses a symmetric function f also satisfies anonymity (Lemma 1) [12].

While providing f-privacy and anonymity are both key interests for a privacy-preserving aggregation scheme, the scheme also needs to provide correctness and robustness before it can even be considered useful in practice. Whereas correctness considers whether a scheme computes the correct result, robustness tries to ensure that even adversarial clients can only

influence the final aggregate by submitting *valid* inputs. This is especially important if we consider the transition from ideal functionalities to protocols. An ideal functionality might, for example, only consider integers between 0_{10} and 10_{10} as valid inputs, which we may represent using a 4-bit integer. Using this 4-bit integer however, it is also possible to represent values not in the intended range, such as 13_{10}, which we do not consider to be a valid inputs. This leads us to the following definition:

Definition 27 (Robustness (against adversarial clients) [12])**.** Let $\mathcal{E}$ be the set of all valid data values a client may submit. Fix a security parameter $\lambda > 0$. We say that an n-client data-aggregation scheme provides *robustness* if, when all servers execute the protocol faithfully, for every number m of malicious clients (with $0 \leq m \leq n$), and for every choice of honest client's inputs $(x_1, \ldots, x_{n-m}) \in \mathcal{E}^{n-m}$, the servers, with all but negligible probability in λ, output a value in the set:

$$\{ f(x_1, \ldots, x_n) \mid (x_{n-m+1}, \ldots, x_n) \in \mathcal{E}^m \}.$$

A similar definition can be used to define robustness against adversarial servers. Unless specified otherwise, we will always use the term robustness and mean robustness against adversarial clients.

Definition 28 (Secure Channel [29])**.** When talking about a secure channel we mean a channel that is both encrypted and authenticated. We use the common definitions for encrypted and authenticated channels.

Even though a secure channel is both encrypted and authenticated, it does not protect against all forms of information leakage. For instance, a party observing the channel can see if - and to whom - a message was sent as well as make deductions about the length of the underlying unencrypted message. For a party with active control over the network it might also be possible to drop packages based on the above mentioned information.

2.1.9. Aggregation Function Definitions

Aggregation schemes can support one or more aggregation functions, which describe the function that is evaluated by the scheme. When speaking about aggregation functions we usually only refer to their realised functionality and not their implementation, which can vary widely between different schemes.

In this section we will take a look at all the functions used in this thesis. We will provide a definition for each function, investigate their implementation independent properties and how they relate to each other.

SUM Let $\mathcal{D}$ be a group under addition. We say a function $f : \mathcal{D}^d \to \mathcal{D}$ computes SUM if its output is the sum of all inputs. If applicable the calculation may "under-" or "overflow" the modulus (order).

SUM$_{I \subset \mathbb{Z}}$ Same as SUM, but client inputs are restrained to the interval $I \subset \mathbb{Z}$. Let p be the cardinality of $\mathcal{D}$ and n be the number of participating clients. If $\lfloor p/n \rfloor$ is an upper bound for the cardinality of I SUM$_{I \subset \mathbb{Z}}$ will not overflow. This limits the amount possible input values significantly, but allows something similar to integer arithmetic.

COUNT Equivalent to $\texttt{SUM}_{\{0,1\}}$ and can be interpreted as counting how many clients submitted 1.

SORT Let $\mathcal{E}$ be a set on which a total order is defined. We say a function $f : \mathcal{E}^d \to \mathcal{E}^d$ computes SORT if each element is no smaller (larger) than the previous element and the output is a permutation of the input.

The authors of Prio have provided affine-aggregatable encodings (AFEs) [12]. When encoding their input using an appropriate AFE, a data aggregation scheme can compute more advanced aggregates than simple sums. As such we will not spend too much time on exploring different functions and refer to their work instead.

2.2. Building Blocks

Building blocks can be seen as "instantiations" of some of the formal definitions we have seen in the previous section. For some of the preceding definitions we only give a single construction, while we provide two constructions for other definitions. This is due to these constructions having different properties, which makes them suitable for different aspects within a data aggregation scheme.

2.2.1. Cryptosystems

The Paillier cryptosystem, first presented by Pascal Paillier in 1999, is a probabilistic, asymmetric public-key encryption scheme (Definition 3) based on the assumption that computation of n-th residues modulo n^2 is computationally difficult. It is IND-CPA secure under the decisional composite residuosity assumption (Definition 2), and additive homomorphic (Definition 4), meaning that given only the public key and the encryptions of the messages m_1 and m_2, it is possible to calculate the encryption of $m_1 + m_2$.

Construction 1 (Paillier Cryptosystem [44])**.** The Paillier cryptosystem consists of three probabilistic polynomial time algorithms *Gen* (Algorithm 1), *Enc* (Algorithm 2) and *Dec* (Algorithm 3):

Algorithm 1: Gen (Paillier)

Input: Security parameter κ (bit length of keys)
Output: Public key (n, g) and private key (λ, μ)

$p \xleftarrow{\$} \mathbb{P}_{\kappa/2}$; /* Select a random prime of bit length κ / 2 */
$q \xleftarrow{\$} \mathbb{P}_{\kappa/2}$;
assert($\gcd(pq, (p-1)(q-1)) == 1$); /* Always true if p and q are of equal length [29] */
$n := pq$;
$\lambda := \text{lcm}(p-1, q-1)$;
do
  $g \xleftarrow{\$} \mathbb{Z}^*_{n^2}$;
while $\gcd(L(g^\lambda \mod n^2), n) \neq 1$ /* Ensure that n divides order of g [44] */;
$\mu := (g^\lambda \mod n^2)^{-1} \mod n$;
return $(pk = (n, g), sk = (\lambda, \mu))$

Algorithm 2: Enc (Paillier)

Input: Public key (n, g), plaintext message $m \in \mathbb{Z}_n$
Output: Ciphertext c

$r \xleftarrow{\$} \mathbb{Z}^*_n$;
$c = g^m r^n \mod n^2$;
return c

Algorithm 3: Decrypt (Paillier)

Input: Private Key (λ, μ), public key (n, g), ciphertext c
Output: Plaintext message m

$u := c^\lambda \mod n^2$;
$v := g^\lambda \mod n^2$;
$m := L(u)/L(v) \mod n$;
return m

Algorithm 4: Helper Function $L_n(x)$ (Paillier)

Input: Integer x, public key (n, g)
Output: Integer, result of the L function
return *(x - 1)/n*; /* a/b refers to the quotient of a divided by b and not to modular multiplication of a times b^{-1} */

Proofs for *correctness* and *security* of the Paillier cryptosystem can be found in Paillier's original publication [44].

Homomorphic Properties We can easily verify the homomorphic properties of the Paillier cryptosystem:

Homomorphic addition of plaintexts Multiplying two ciphertexts c_1 and c_2 leads to the addition of the underlying plaintexts m_1 and m_2:

$$\begin{aligned} c_1 \cdot c_2 &= g^{m_1} r_1^n \cdot g^{m_2} r_2^n \\ &= g^{m_1+m_2}(r_1 r_2)^n \\ &= g^{m_1+m_2}(r'^n) \mod n^2 \\ Dec(g^{m_1+m_2}(r'^n) \mod n^2) &= m_1 + m_2 \mod n. \end{aligned}$$

Homomorphic multiplication with constants Raising a ciphertext c_1 with corresponding plaintext m_1 to a constant k will decrypt to the product of m_1 and k:

$$\begin{aligned} c_1^k &= (g^{m_1} r^n)^k \\ &= g^{m_1 k}(r^k)^n \\ &= g^{m_1 k} r'^n \mod n^2 \\ \mathrm{Dec}(g^{m_1 k} r'^n \mod n^2) &= k m_1 \mod n. \end{aligned}$$

2.2.2. Secret Sharing

This section provides two constructions for secret sharing schemes as defined in Definition 6. The first construction provides (n, n)-threshold secret sharing, while the second one allows for (t, n)-threshold secret sharing.

2.2.2.1. Additive Secret Sharing

One of the easiest ways to implement secret sharing is additive secret sharing, which can be implemented in a very straight forward fashion. To retrieve the secret using additive secret sharing every single share is required, which makes additive secret sharing a (n, n)-threshold scheme.

Construction 2 (Additive secret sharing)**.** Let $\mathbb{F}$ be a finite field, $x \in \mathbb{F}$ and n the desired number of shares. Let $[x]_i$ denote the ith share of x.

Algorithm 5: Share (Additive secret sharing)

Input: Secret data $x \in \mathbb{F}$
Output: Ordered list $\mathbb{F}^n$ of shares

```
S ← {};                                          /* Set of secret shares */
for i = 1 to n − 1 do
    [x]_i ←$ F;                                  /* Select a share uniformly at random */
    S ← S ∪ {[x]_i};
[x]_n = Σ_{i=1}^{n−1} [x]_i;                     /* Shares 1 to n − 1 uniquely determine share n */
S ← S ∪ {[x]_i};
return S
```

Algorithm 6: Reconstruct (Additive secret sharing)

Input: List of shares $\{[x]_1, \ldots, [x]_n\} \in \mathbb{F}^n$
Output: Secret value $x \in \mathbb{F}$
return $x = \sum_{i=1}^{n} [x]_i$

Algorithm 7: Additive secret sharing

Function *Share*$(x \in \mathbb{F}) \rightarrow \mathbb{F}^n$**:**
  $S \leftarrow \{\}$; /* Set of secret shares */
  for $i = 1$ **to** $n - 1$ **do**
    $[x]_i \xleftarrow{\$} \mathbb{F}$; /* Select a share uniformly at random */
    $S \leftarrow S \cup \{[x]_i\}$;
  $[x]_n = \sum_{i=1}^{n-1} [x]_i$; /* Shares 1 to $n-1$ uniquely determine share n */
  $S \leftarrow S \cup \{[x]_i\}$;
  return S

Function *Reconstruct*$(\{[x]_1, \ldots, [x]_n\} \in \mathbb{F}^n) \rightarrow \mathbb{F}$**:**
  return $x = \sum_{i=1}^{n} [x]_i$

Properties Since it is not possible to infer any information about the secret without having acquired access to all shares, additive secret sharing is information-theoretical secure. Furthermore it is a linear scheme (Definition 7), which allows the combination of single shares using affine operations. If $[x]_i$ and $[y]_i$ are the ith shares of x and y respectively, $[x + y]_i$ is the ith share of $x + y$ and $[\alpha x + \beta]_i$ is the ith share of $\alpha x + \beta$, for $\alpha, \beta \in \mathbb{F}$ [8].

2.2.2.2. Shamir's Secret Sharing

Continuing in this direction we now want to describe a (t, n)-threshold secret sharing scheme that is also applicable for any arbitrary $1 \leq t \leq n$, granting us more flexibility than the simpler additive scheme. One of the most well known solutions to this problem was published by Shamir in 1979 [50] and is based on polynomial interpolation. His construction makes use of the fact that, given t points in the 2-dimensional plane $(x_1, y_1), \ldots, (x_t, y_t)$ with pairwise distinct x_i's, there is exactly one polynomial $q(x)$ of degree $t - 1$ for which $q(x_i) = y_i$ for all $1 \leq i \leq t$:

Construction 3 (Shamir's Secret Sharing [50])**.** Let $\mathbb{F}$ be a finite field, $a_0 \in \mathbb{F}$ a secret value, t the number of shares that are required to reconstruct a_0, and $n < |\mathbb{F}|$ the number of shares that are generated.

The *Share algorithm* randomly samples coefficients $a_1, \ldots, a_{t-1} \in \mathbb{F}$ using a uniform distribution over $\mathbb{F}$, which it uses us to construct a polynomial

$$q(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{t-1} \cdot x^{t-1}.$$

To generate the n shares $[a_0]_i$ it now evaluates q at points $1, \ldots, n \in \mathbb{F}$ and outputs them together with their index:

$$[a_0]_i = (i, q(i))$$

for $1 \leq i \leq n$. Note that by construction each share is also a point on the polynomial.

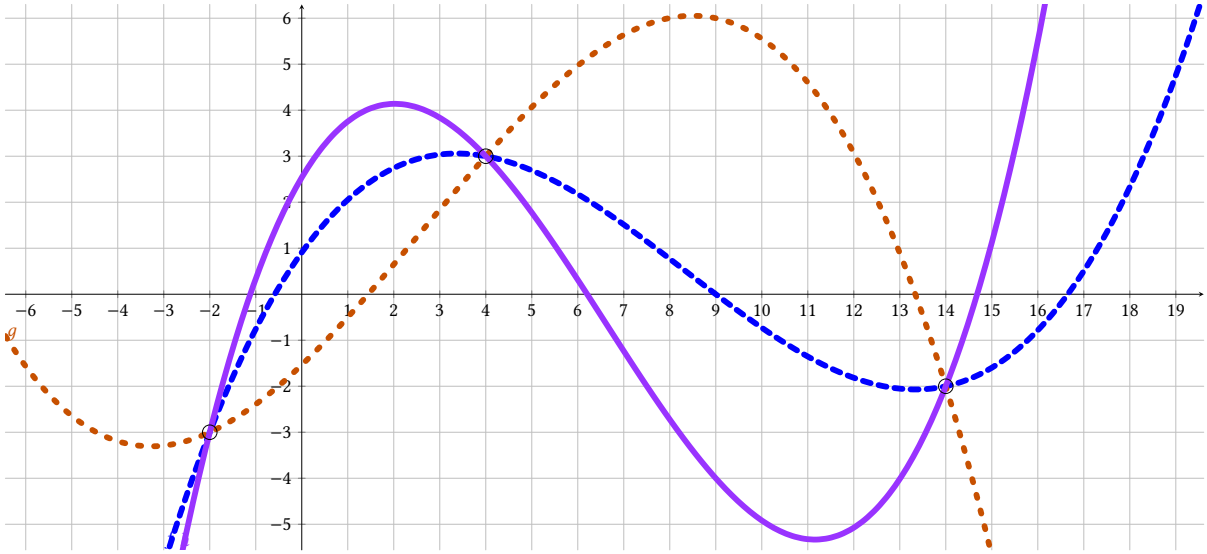

Figure 2.1.: Three points do not uniquely define a polynomial of degree 3.

To unveil the secret value, the *Reconstruct algorithm*[2] needs to find the constant term a_0 of the polynomial $q(x)$. This can be achieved by first using polynomial interpolation to calculate the polynomial $q(x)$ and then evaluating $q(0)$. An optimized formula using Lagrange polynomials does not require computing unused constants and is stated here [35]:

$$a_0 = q(0) = \sum_{j=0}^{t-1} y_i \prod_{\substack{m=0 \\ m \neq j}}^{t-1} \frac{x_m}{x_m - x_j},$$

where $(x_i, y_i) = [a_0]_i = (i, q(i))$ are t points on the polynomial.

Properties Shamir's secret sharing shares has much of its properties in common with the simpler additive secret sharing scheme, these include

Information-theoretic security Any combination of up to $t - 1$ shares does not reveal any information about the secret value [50].

Linearity (Definition 7) Shares of secrets x and y can be used to calculate shares of secret $x + y$ [45].

Minimality The size of each share does not exceed the size of the original data [50].

Both of the above constructions are only concerned with sharing a secret $x \in \mathbb{F}$. To share a higher dimensional secret $x' \in \mathbb{F}$ we can simply share the secret component wise.

2.2.3. Zero-knowledge proofs

SNIPs While most standard zero-knowledge proofs assume a scenario where a single prover tries to convince a single verifier that they posses some secret knowledge, SNIPs consider scenarios where a single prover tries to convince multiple verifiers that the prover not only

[2]While the trivial approach presented here has complexity $O(t^2)$, more efficient algorithms for polynomial interpolation with complexity $O(t \log^2 t)$ are discussed in [1] and [32].

has knowledge of this secret, but that the verifiers also hold secret shares of this secret. While classical non-interactive zero-knowledges (NIZKs) or succinct non-interactive argument of knowledges (SNARKs) can also achieve this, the usage of SNIPs can provide clear advantages in this setting, such as a reduced amount of expensive cryptographic operations.

Construction 4 (*Prio* SNIPs [12])**.** Let Valid be a predicate that is implemented using an arithmetic circuit using exactly M multiplication gates. Let $\mathbb{F}$ be a finite field such that $2M \ll |\mathbb{F}|$ and let $x \in \mathbb{F}$ be secret for which the prover wants to convince the verifiers that $\text{Valid}(x) = 1$.

To construct a SNIP as introduced by Prio the prover and verifiers adhere to the following steps:

Step 1: Client evaluation In this step the prover evaluates the Valid circuit locally on its secret input x, which provides it with the knowledge of the value of each wire in the circuit during the computation of $\text{Valid}(x)$. The prover uses this knowledge to construct two polynomials f, g, which encode the values of the input wires on each of the multiplication gates. Using polynomial interpolation it creates a third polynomial, h,which encodes the output wires of all the multiplication gates.

To construct the polynomials the prover proceeds as follows. Sorting the multiplication gates topologically from inputs to outputs, the t-th multiplication gate has inputs u_t and v_t. For all $1 \leq t \leq M$ we define the polynomials f and g to be the lowest-degree polynomials such that $f(t) = u_t$ and $g(t) = v_t$. We can then define the polynomial h as $h = f \cdot g$. Since f and g are constructed using the inputs of the multiplication gates, it is easy to see that they will have degree at most $M-1$, as such h will have degree at most $2M-2$. Since $h(t) = f(t) \cdot g(t) = u_t \cdot v_t$ for all $t \in \{ 1, \ldots, M \}$, $h(t)$ is equal to the output wire $(u_t \cdot v_t)$ of the t-th multiplication gate in the $\text{Valid}(x)$ circuit.

The prover now splits the coefficients of h using additive secret sharing and sends the i-th share of the coefficients $[h]_i$ to verifier i.

Step 2: Consistency checking at the servers At the beginning of step 2 each verifier i holds a share $[x]_i$ of the prover's secret value x, as well as a share $[h]_i$ of the polynomial h. Using the same arithmetic circuit for the Valid predicate, the verifier can now use this information to locally compute shares $[f]_i$, $[g]_i$ of the polynomials f and g.

Assuming a verifier had access to a share for each wire value in the arithmetic circuit, it is easy to see that it could construct shares $[f]_i$ and $[g]_i$ using polynomial interpolation as the prover does in step 1. We now show that verifier i can indeed construct shares for each of the wire values using only $[x]_i$ and $[h]_i$ and following the gates in topological order.

Input gates Input gates are either laballed with a constant in $\mathbb{F}^l$, which are part of the circuit and thus known to the verifier, or labelled with a variable, of which the verifier has received a share, namely $[x]_i$.

Sum gates Using any two wire shares the verifier already has knowledge of, it can derive the wire share of the missing third wire using only affine operations.

Multiplication gates The verifier has shares of both of the input wires for each multiplication gate. Using $[h]_i$ it can calculate a share for the output wire value of the t-th multiplication gate as $[h]_i(t)$.

Using these shares the verifier now uses polynomial interpolation to construct $[f]_i$ and $[g]_i$. If all parties have acted honestly up to this point, the verifiers now hold shares of polynomials f, g and h such that $f \cdot g = h$.

Let us now consider a dishonest prover, which could have sent the verifiers shares of a different polynomial $\hat{h}$ such that, for some $t \in \{ 1, \ldots, M \}$, $\hat{h}(t)$ is *not* the value on the output wire in the t-th multiplication gate of the Valid(x) computation. If this is the case the verifiers do not hold shares of the polynomials f and g, but instead hold shares of different polynomials $\hat{f}$ and $\hat{g}$. It is then with certainty that $\hat{h} \neq \hat{f} \cdot \hat{g}$. This can be verified by considering the least t_0 for which $\hat{h}(t_0) \neq h(t_0)$. Such a t_0 must exist, since otherwise h and $\hat{h}$ would be equal. By construction we have that $\hat{f}(t) = f(t)$ and $\hat{g}(t) = g(t)$ for all $t \leq t_0$. Using

$$\hat{h}(t_0) \neq h(t_0) = f(t_0) \cdot g(t_0) = \hat{f}(t_0) \cdot \hat{g}(t_0),$$

we see that $\hat{h}(t_0) \neq \hat{f}(t_0) \cdot \hat{g}(t_0)$ and as such $\hat{h} \neq \hat{f} \cdot \hat{g}$. This approach was inspired by the use of polynomial identities to check the consistency of secret shared values in [9].

Stap 3a: Polynomial identity test Starting of step 3a each verifier i holds shares $[\hat{f}]_i$, $[\hat{g}]_i$ and $[\hat{h}]_i$ of polynomials $\hat{f}, \hat{g}$ and $\hat{h}$. We have also seen that $\hat{f} \cdot \hat{g} = \hat{h}$ if and only of the verifiers collectively hold wire shares that, when summed up, equal the internal wire values of the Valid(x) circuit computation done by the prover. The verifiers now need to check whether this equality holds.

The construction presented by Prio uses the Schwartz-Zippel randomized polynomial identity test [48, 66] to check for this property. The underlying idea of this test is that if two polynomials are equal, subtracting one from the other should yield the zero polynomial. In other words, if $\hat{f} \cdot \hat{g} \neq \hat{h}$ then the polynomial $\hat{f} \cdot \hat{g} - \hat{h}$ is a non-zero polynomial (of degree at most $2M - 2$). We know that such a polynomial can have at most $2M - 2$ zeroes in $\mathbb{F}$. That means that, for a (independent and identically) random value $r \in \mathbb{F}$, the verifiers will detect that $\hat{f} \cdot \hat{g} \neq \hat{h}$ with probability at least $1 - \frac{2M-2}{|\mathbb{F}|}$.

Before executing this test, the verifiers need access to such a random value $r \in \mathbb{F}$. In the assumption model made by Prio it is sufficient if a single verifier samples r and distributes it to the other verifiers[3] . Since polynomial evaluation only requires affine operations on the shares, each verifier can locally evaluate the polynomial shares $\hat{f}, \hat{g}$ and $\hat{h}$ on the point r to get shares $[\hat{f}(r)]_i$, $[\hat{g}(r)]_i$ and $[\hat{h}(r)_i]$.

Let us now assume for a moment that each verifier can also locally multiply shares $[\hat{f}(r)]_i$ and $[\hat{g}(r)]_i$ locally, which is *not* an affine operation, to generate a share $[\hat{f}(r) \cdot \hat{g}(r)]_i$. Using the so generated share each verifier can use an affine operation to produce share $\sigma_i = [\hat{f}(r) \cdot \hat{g}(r) - \hat{h}(r)]_i$. To ensure that $\hat{f}(r) \cdot \hat{g}(r) = \hat{h}(r)$ the verifiers publish their shares σ_i and compute $\sum_i \sigma_i = 0 \in \mathbb{F}$. The verifiers reject the proof if $\sum_i \sigma_i \neq 0$.

[3]The aggregation scheme the authors of this SNIP construction describe only makes correctness and robustness guarantees when all verifiers are honest. If this is the case we can indeed assume that r is randomly sampled from $\mathbb{F}$. If the sampling of r is not random the sampling verifier can negatively impact the availability of the aggregation scheme and, if the sampling is always done by the same verifier, perform a selective denial-of-service attack.

Step 3b: Multiplication of shares We now take a closer look at the assumption we needed to make at the end of the previous step, regarding the multiplication of shares $[\hat{f}(r)]_i$ and $[\hat{g}(r)]_i$ to get $[\hat{f}(r) \cdot \hat{g}(r)]_i$, without leaking anything about the values of $\hat{f}(r)$ and $\hat{g}(r)$ in the process. This can be achieved using Beaver triples (subsection 2.2.4). To improve performance these triples do not need to be calculated using expensive MPC techniques, but can instead be created by the client for the servers. Corrigan-Gibbs and Boneh have also shown, that even if the client sends shares of an invalid triple to the servers, that the servers will catch this client with high probability [12].

Step 4: Output verification Assuming all verifiers have behaved honestly up to this point, each verifier i now holds shares $[w_1]_i, [w_2]_i, \ldots$ of the values of all the wires in the Valid(x) circuit computation. This includes shares of the input wires $[x]_i$ and output wire $[o]_i$ of the circuit. To uncover the result of the Valid(x) verification computation the verifiers now publish their output wire shares $[o]_i$ and sum up the shares to receive the result. If $\sum_i [o]_i = 1$ it follows that also Valid$(x) = 1$, except with a small failure probability due to the polynomial identity test.

Choosing a random value for polynomial identity testing Step 3a in the previous construction required us to decide on a random variable r, which was randomly sampled by one of the verifiers. This decision is justified if we assume that at least one of the verifiers is honest, since the so derived shares σ_i have to add up to 0.

2.2.4. Beaver Triples

While we have already seen secret sharing schemes that allow affine operations on shares, we have yet to see a construction that also allows us to multiply shares of x, y to receive shares of $x \cdot y$. In 1991 Beaver introduced a technique [5] that allows us to do just that. Using random one-time-use share $([a]_i, [b]_i, [c]_i) \in \mathbb{F}^3$ with the constraint that $a \cdot b = c \in \mathbb{F}$, distributed by a trusted dealer, it becomes possible to efficiently multiply shares of x, y without leaking any information, if enough shareholding parties work together.

Construction 5 (Beaver Triples [5, 12])**.** Let $\mathbb{F}$ be a finite field, $n \in \mathbb{N}$ the number of of shares, $(a, b, c) \in \mathbb{F}^3$ a *multiplication triple* with the constraint that $a \cdot b = c$, randomly sampled by a trusted dealer and not known to the parties multiplying the shares, and $([a]_i, [b]_i, [c]_i) \in \mathbb{F}^3$ shares of these values generated using a linear secret sharing scheme. Let $[x]_i, [y]_i \in \mathbb{F}$ be the shares of values $x, y \in \mathbb{F}$ that we wish to multiply.

Each party starts by computing the values

$$[d]_i = [x]_i - [a]_i \quad ; \quad [e]_i = [y]_i - [b]_i$$

which it then broadcasts to all other parties. This allows every party to reconstruct d and e using the received shares, which can be used to compute:

$$\sigma_i = den^{-1} + d[b]_i + e[a]_i + [c]_i,$$

where n^{-1} refers to the inversion of the element n in the field $\mathbb{F}$. σ_i is now a share of the product $x \cdot y$.

Correctness Showing correctness only requires some simple arithmetic transformation:

$$\begin{aligned}\sum_i \sigma_i &= \sum_i (den^{-1} + d[b]_i + e[a]_i + [c]_i) \\ &= de + db + ea + c \\ &= (x-a)(y-b) + (x-a)b + (y-b)a + c \\ &= (x-a)y - (x-a)b + (x-a)b + (y-b)a + c \\ &= (x-a)y + (y-b)a + c \\ &= xy - ay + ya - ba + c \\ &= xy - ba + c \\ &= xy\end{aligned}$$

We start by inserting the definition of σ_i, reconstructing shares $[a]_i$, $[b]_i$ and $[c]_i$, doing some simple arithmetic transformations and finally making use of the constraint of the multiplication triple that $c = ab$. This proof shows that $\sum_i \sigma_i = xy$, from which the claim that $\sigma_i = [xy]_i$ directly follows.

Security In his paper [5] Beaver has shown, that secure multiplication in MPCs becomes possible using this construction.

2.2.4.1. Differential Privacy

3. Existing Approaches

Having seen the building blocks introduced in section 2.2, we can now examine some existing approaches to the problem at hand. While this list of approaches does not claim to be exhaustive, it does cover the most common ideas that have been presented to date to solve the issue of privacy-preserving data aggregation. We can broadly classify the existing approaches into three main architectures, which describe the underlying communication model, in which the servers and clients are involved. These are Client-Server Architectures (CSAs), (Client Cluster)-Server Architectures (CcSAs) and Client-(Distributed Server) Architectures (CDsAs), each of which providing advantages over the others, while also having a unique set of challenges, when considering their use for privacy-preserving data aggregation, especially regarding privacy, correctness and robustness. The above architectures will be explained in the following sections. To build trusted aggregation schemes for each architecture, we can combine different sets of tools: Statistics and probability, trusted hardware (such as Intel SGX [39]) and trusted schemes. In this thesis we will place our focus on trusted schemes.

For two of the three architectures we describe one or more schemes that try to solve the problem in a somewhat unique way. The (Client Cluster)-Server Architecture is included here since it will be referenced in a later chapter. We will not delve too deep into most of the presented approaches unless necessary for future sections in this thesis. However, where possible we will reference relevant publications that can provide a more detailed insight into the approaches. Each of the approaches described within this section will end with an overview of its strengths and weaknesses as well as a summary of its security implications.

3.1. Client-Server Architecture

The Client-Server Architecture (Figure 3.1) is the simplest of the three architectures and follows a star like pattern. It has a single entity at its centre that consists of one or more servers, but the clients only communicate with one of them. Any additional servers may only communicate with the other servers. Any client acts independently from other clients and does not communicate with other clients. While the exact properties are dependant on the scheme and not the architecture, we can see that this architecture could provide relatively easy access to initial scaling, as new clients only need to know how to communicate with the server. A major disadvantage is the cumulation of trust on the single central entity.

3.1.1. Homomorphic Encryption

As defined in the previous chapter, homomorphic encryption (Definition 4) is a form of encryption, that still allows certain computations to be performed on the encrypted data, without the need of decryption. In a an environment where all parties trust each other,

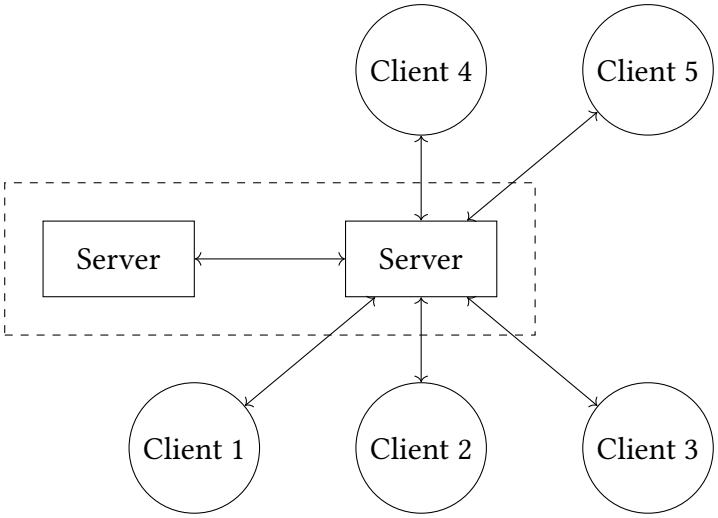

Figure 3.1.: Client-Server Architecture: Clients communicate directly with a single central entity.

a privacy-preserving aggregation scheme for the `SUM` function could simply be realized by choosing an `SUM`-homomorphic cryptosystem such as the Paillier cryptosystem. Since every party trusts every other party it is sufficient if every member encrypts their data and sends it to a central server. Using the additive homomorphic property of the scheme, the server can now compute the encrypted value of the result and decrypt the result. It is easy to see, that this simple approach does not protect against noisy servers, since we have currently no way of preventing the server from decrypting the values before it aggregates them. As such it is only useful for scenarios where an honest party does not want to learn clients' private date involuntarily.

To alleviate this issue we can add a second server to this construction and share their tasks. While one server is responsible for collecting the encrypted data from the clients, only the other server possesses the decryption key. Unless both servers work together, this prevents either server from learning a client's private data.

3.1.1.1. Properties

This leaves us with the following properties for this relatively easy aggregation scheme.

Communication Each client needs to send a single encrypted message to a single server. Once this server has received all messages it aggregates them and sends them as a single ciphertext to the server with the decryption key.

Robustness This data aggregation scheme does not provide robustness against malicious clients, since the server cannot verify the input based on its encryption alone. If desired a zero-knowledge proof (Definition 15) could be added to verify that the submitted value adheres to some set of restrictions.

Privacy Privacy is solely dependant on the assumption, that the two central servers do not collude. The server collection the encrypted data cannot derive any information from the ciphertext, and the server decrypting the result does not have any information regarding its original composition.

3.2. Client-(Distributed Server) Architecture

Client-(Distributed Server) Architectures (Figure 3.2) don't have a required number of servers strictly defined in the protocol, instead the number of servers can be chosen when initialising the scheme. The Client-(Distributed Server) Architecture is similar to the Client-Server Architecture in that clients don't need to communicate with other clients before they can submit data, but the two architectures differ when considering how clients submit their data. In the Client-(Distributed Server) Architecture clients need to connect and submit a piece of information to every central server to take part in the aggregation scheme.

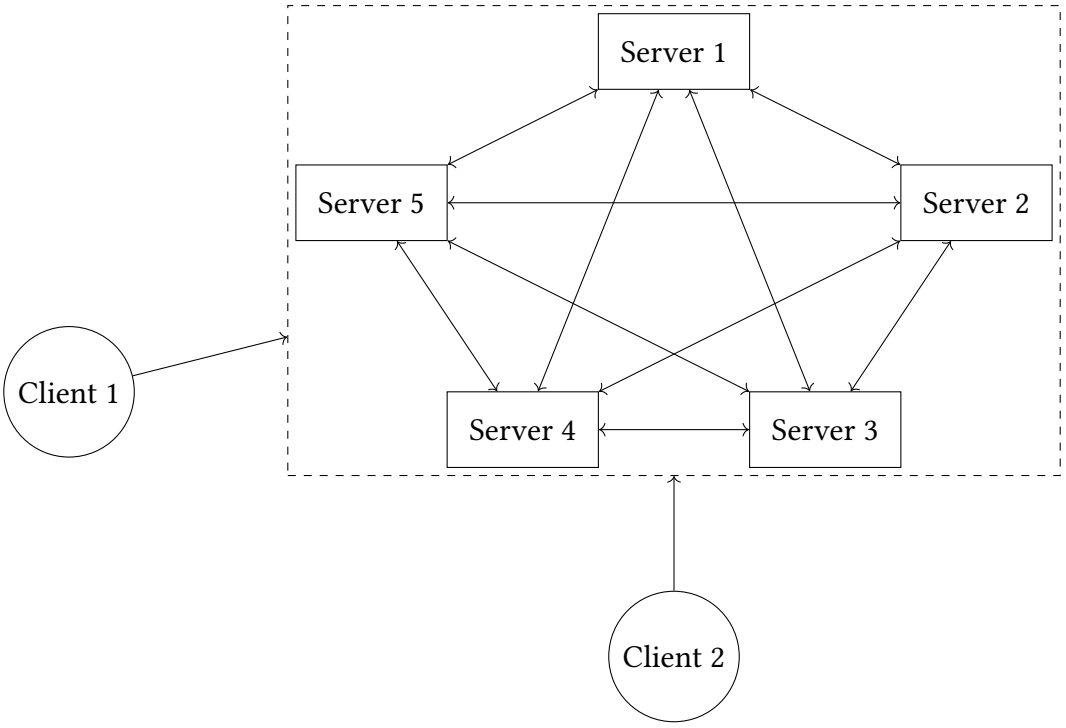

Figure 3.2.: Client-(Distributed Server) Architecture: Clients communicate with multiple distributed servers.

3.2.1. Additive Secret Sharing

A simple scheme following the Client-(Distributed Server) Architecture can be implemented by employing additive secret sharing [12]. Let us consider a scenario, where the clients are trying to vote on a proposal. Each client holds a single bit x_i of information, which indicates whether the client is against or in favour of the proposal, represented by the values 0 or 1 respectively. To determine whether the proposal has been accepted, the servers are now tasked with determining how many clients voted in favour, which means counting how many clients

submitted 1 as their value, or, in different words, the sum over all the x_i. Since we want to employ additive secret sharing as outlined in Construction 2, we select a prime p as system parameter, which we can then use to construct a finite field $\mathbb{F}_p$ of order p.

To initialise the scheme with n servers, each server j holds an accumulator $A_j \in \mathbb{F}_p$, with $A_j = 0$ as initial value. The scheme now follows these three steps:

Upload Since the secret x_i can only be 0 or 1, we can interpret x_i as an element of $\mathbb{F}_p$. The client then used additive secret sharing to split the secret into n shares $[x_i]_j$, one for each server. Using a secure communication channel the client then sends one share to every server.

Aggregate After receiving a secret share from client i, server j simply adds the share to its accumulator: $A_j \coloneqq A_j + [x_i]_j \in \mathbb{F}_p$.

Publish Due to the linearity of additive secret sharing we know that the accumulator of each server holds a share of the final result after it has received a share from every client. Finally every server publishes the value of its accumulator which we can use to reconstruct the desired sum:

$$\sum_{j=1}^{n} A_j = \sum_{j=1}^{n} \sum_{i=1} [x_i]_j = \sum_{i=1} \sum_{j=1}^{n} [x_i]_j = \sum_i x_i \in \mathbb{F}_p.$$

It has to be noted, however, that we have to choose the modulus, the order of the finite field, p with some care. If p is smaller than the number of clients the accumulation of all values could be larger as p as well. In this case the accumulation would "overflow" and lead to an incorrect result.

3.2.1.1. Properties

The biggest strength of this scheme lies in its striking simplicity, which, on the other hand, also contributes to most of its shortcomings. We can thus summarise the following properties:

Communication From a single client's point of view the communication follows a simple pattern, as in only a single message per server per client is required and no interactive communication protocol is used. Considering all servers and clients the total number of messages scales linearly in the number of both. Each client uses an unidirectional channel to send a single message to each server, whereas the servers require some form of bidirectional communication to broadcast their accumulators. The scheme does not provide any mechanisms to ensure that all clients have submitted their shares to all servers, which can pose an issue, since a server may only publish its accumulator once it is certain that it has received shares from all the clients. We also need to consider any overhead that is required to setup the required communication channels.

Robustness This scheme does not provide robustness against malicious clients. As per our definition robustness requires that a client can only influence the result by submitting a *valid* input, which in the scenario described above is either 0 or 1. Since the clients only ever submit shares, which are elements of $\mathbb{F}_p$ themselves, a single server has no way of confirming whether the received share is a share of 0 or 1 respectively or the share of any arbitrary element of $\mathbb{F}_p$.

A similar story holds for malicious or malfunctioning servers, which can always manipulate the value of their accumulator without the possibility of any other party noticing. Therefore, it is not robust against malicious servers either.

Privacy With additive secret sharing as the big and only foundation, the defining properties of additive secret sharing are also present in this scheme. As such it can easily be seen that as least one honest server suffices for the client's data to stay information-theoretical hidden, since knowledge of all shares is required to derive any meaningful information.

Combining the observation that `COUNT` is a symmetric function with our previous remark that the additive secret sharing scheme fulfils `COUNT`-privacy, Lemma 1 gives us that this scheme also provides anonymity.

3.2.2. Prio

A more advanced approach to the aggregation problem is called Prio [12], which was first described in 2017 and extends the additive secret sharing based scheme introduced in subsection 3.2.1. To guarantee that even faulty or adversarial clients can only submit valid data points, Prio introduces SNIPs, a form of zero-knowledge proofs, that allow the servers to verify the adherence of the client's data to some set of restraints without gaining any additional information about the data, apart from the data being valid. The approach presented by the authors only requires a single server to be honest to prevent any client's data to be compromised. Prio supports a vast set of different aggregation functions, from basic `SUM` to `LEAST-SQUARES` regression, which is a mathematical method to find the best-fitting curve for a given set of points by minimising the sum of the squares of the distance between the given points and the curve itself. For most of these functions Prio learns nothing about the client's data, except what can directly be inferred from the aggregation result. However, some more advanced aggregation functions may leak some additional information about the clients' data. It further differentiates itself from the additive secret sharing scheme by not only supporting elements of $\mathbb{F}$ as data points, but also vectors of higher dimension in $\mathbb{F}^l$.

3.2.2.1. Robustness

One of the major weaknesses of the additive secret sharing scheme (subsection 3.2.1) is the undesired property that a single malicious client can manipulate the output to a degree that is not intended by the protocol. The voting example in subsection 3.2.1 assumes that every participating party submits either a 0 or 1. However, since each server only receives a share of the secret value, it is not trivially possible for the servers to verify the input without also invalidating the client's privacy. A client can thus submit any arbitrary value that is an element of the finite field $\mathbb{F}$ over which the scheme is defined. This allows the client to either increase

or decrease the likelihood of the proposal being accepted, as long as the modulus does not "over-" or "underflow".

Prio solves this by providing the servers with a validation predicate Valid($\cdot$), which the servers can use to decide whether to accept or reject a client's submission $x \in \mathbb{F}^l$. A submission should only be accepted if Valid(x) = 1.

To implement the Valid($\cdot$) predicate Prio uses SNIPs (Construction 4). Without leaking information about x to the servers and for any arbitrary predicate Valid, SNIPs allow the servers to quickly verify whether Valid(x) = 1.

3.2.2.2. The Scheme

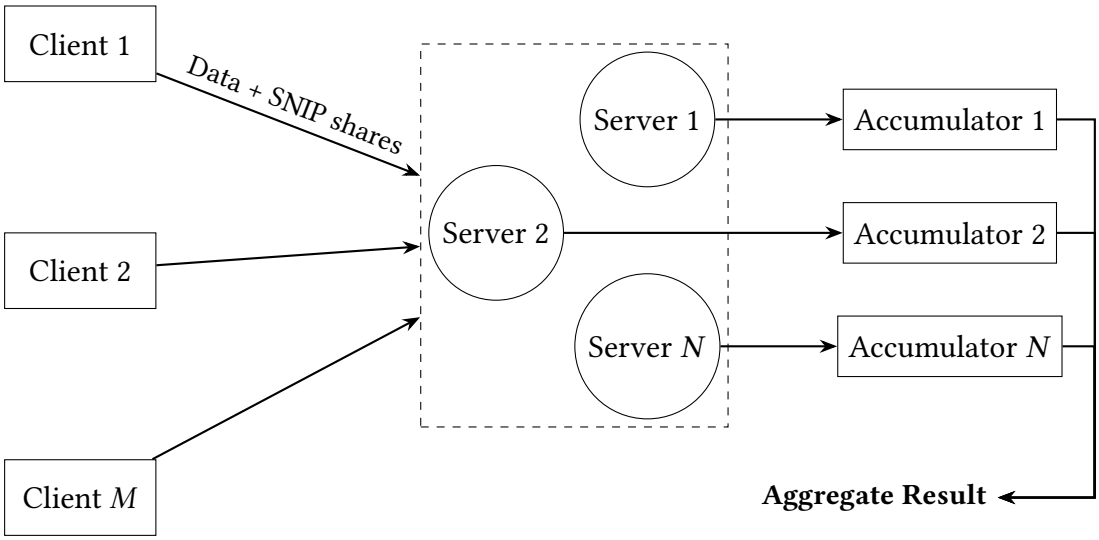

Figure 3.3.: The Prio scheme

The underlying idea is very similar to the additive secret sharing scheme, which the following is heavily based on. Instead of a single bit of information, a client in Prio can hold a multidimensional secret vector $x \in \mathbb{F}^l$. Even though Prio supports a plethora of different aggregation functions, we will restrict ourselves to computing sums for now. An explanation of how the other functions are achieved can be found in the corresponding paper [12]. As in the additive secret sharing scheme each of the n servers has its own aggregator $A_j \in \mathbb{F}^l$, which is initialised with $A_j = 0$. An additional requirement is the exact Valid used in the protocol, which has to be agreed upon by the servers and be somehow accessible to all clients. Figure 3.3 outlines the protocol. Prio now follows these steps:

Upload Client i used additive secret sharing to split its secret x_i into n shares $[x_i]_j$, one for each server. Using SNIPs (Definition 16) to prove that Valid(x_i) = 1, the client also calculates n distinct proof shares $[\tilde{x}_i]_j$, again, one for each server. The client then sends a pairing of a share and its corresponding proof share via a secure channel to each server.

Validate After all servers have received a share and corresponding proof share from client i, they can start to evaluate the Valid predicate themselves. To do so the servers must communicate with each other, but are not allowed to share any of the received shares. In case Valid(x) evaluates to 1 the protocol may continue, in all other cases the shares of x_i across all servers are discarded.

Aggregate After having successfully validated a secret share from client i, server j simply adds the share to its accumulator: $A_j := A_j + [x_i]_j \in \mathbb{F}^l$.

Publish Due to the linearity of additive secret sharing we know that the accumulator of each server holds a share of the final result after it has received a share from every client. Finally every server publishes the value of its accumulator which we can use to reconstruct the desired sum:

$$\sum_{j=1}^{n} A_j = \sum_{j=1}^{n} \sum_{i=1} [x_i]_j = \sum_{i=1} \sum_{j=1}^{n} [x_i]_j = \sum_i x_i \in \mathbb{F}^l.$$

3.2.2.3. Pseudocode

To better visualise the exact workings of Prio, algorithm 8 describes the scheme using high level pseudocode.

3.2.2.4. Properties

In this basic form Prio shares some of its properties with the additive secret sharing scheme, while improving significantly in some aspects. It does however come with some additional costs, such as creating the proof shares and evaluating the Valid predicate.

Communication As it is the case with the simple additive secret sharing scheme, all a client has to do in Prio is to send a single message to each of the n servers, which consists of a secret share and a proof share. This can be done using an unidirectional communication channel. The communication overhead among the servers is dependent on the implementation of SNIPs. For the implementation presented in this paper however the servers require bidirectional communication channels to broadcast the intermediate results of the Beaver's Triple MPC protocol. This requires n^2 sent messages.

Robustness Prio enables robustness against malicious clients by using the Valid predicate, as has been outlined in subsubsection 3.2.2.1. Prio does not, however, provide robustness against malicious or faulty servers. While robustness against such servers would be a positive property, the original authors intentionally decided against it, stating a loss of privacy as the main reason. If Prio were to provide robustness against k faulty servers it would needs to provide a mechanism for $n - k$ servers to still compute the correct aggregate, where n is the number of Prio servers. This also means that $n - k$ adversarial servers working together could break the privacy of any single client by reconstructing its input before truthfully continuing with the Prio, with the other servers being entirely oblivious to this breach. Therefore this version of Prio could only protect privacy when at most $n - k - 1$ servers are working together, a considerable loss compared to the actual Prio.

Privacy As with the previously introduced additive secret sharing based scheme, Prio guarantees privacy if at least one server remains honest. Sharing of the private data itself remains information-theoretical secure. Since SNIPs possess the zero-knowledge property, we can additionally guarantee, that the servers learn nothing about a client's secret data x.

Algorithm 8: Prio

Input: Set of m clients,
with client i having a secret input x_i,
Number of servers n,
Aggregation function f,
Valid predicate $\text{Valid}(\cdot)$

Output: Aggregation result $f(X)$
where $X = \{ x_1, \ldots, x_m \}$ is the set of all secret client inputs

`/* Aggregator Initialisation */`
foreach *Server* j **do**
  $S_j \coloneqq 0$;

`/* Client Submissions */`
foreach *Client* i **do**
  $\{[x]_{i,1}, [x]_{i,2}, \ldots, [x]_{i,n}\} \leftarrow \text{AdditiveSecretShare}_n(x)$;
  $\{[\pi]_{i,1}, [\pi]_{i,2}, \ldots, [\pi]_{i,n}\} \leftarrow \text{SNIP}_{n,\text{Valid}(\cdot)}(x)$;
  foreach *Server* j **do**
    Submit $[x]_{i,j}$ and $[\pi]_{i,j}$ to server j;

`/* Receive and Verify */`
foreach *Submission* $[s]_i = \left\{ ([x]_{i,1}, [\pi]_{i,1}), \ldots, ([x]_{i,n}, [\pi]_{i,n}) \right\}$ **do**
  $valid \coloneqq \text{VerifySNIP}_{\text{Valid}(\cdot)}([s]_i)$;
  if $valid \neq 1$ **then**
    Reject submission $[s]_i$;
  else
    foreach *Server* j **do**
      $S_j \coloneqq S_j + [x]_{i,j}$;

`/* Aggregate Result */`
foreach *Server* j **do**
  $Publish(S_j)$;
  $S \coloneqq Receive(S_1, \ldots, S_{j-1}, S_{j+1}, \ldots, S_n)$;
  return $\sum_{s \in S} s + S_j$;

3.3. (Client Cluster)-Server Architecture

The (Client Cluster)-Server Architecture (Figure 3.4) is similar to the Client-Server Architecture in that the clients form a star pattern around a single central entity, which can be a single server. What differentiates the two architectures however is the fact, that clients in the (Client Cluster)-Server Architecture are grouped into clusters. Clusters are not required to be static and can be updated as needed by a given scheme. Inside their respective clusters clients are aware of each other and can freely communicate. Being inside a cluster allows a client to exchange messages with other clients before submitting data to the central entity. Depending on the setting as much as on the scheme, this can potentially be used to build trust inside the cluster and hide yourself among the other clients. We will examine different ways to create those clusters when we investigate scenarios where this architecture might become useful (section 5.1).

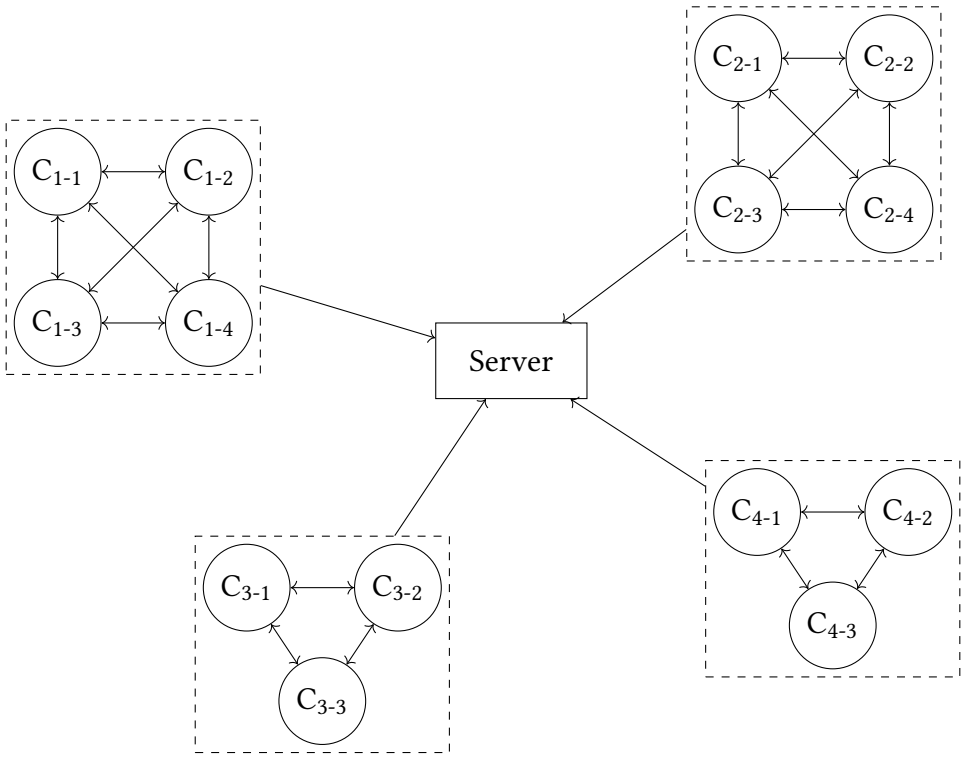

Figure 3.4.: (Client Cluster)-Server Architecture: Clients communicate within clusters.

4. Client Based Prio

Now that we are familiar with the existing approaches seen in the previous chapter, we can try and devise a new scheme that improves upon the existing ones.

While Prio (subsection 3.2.2) is a versatile scheme, all security guarantees given rely on the assumption that at least one server is honest, meaning it does neither collude with other servers nor clients. While this might be a valid assumption, we can also make a case for a different approach. Trusting a server is an abstract concept and wanting some more control over who you trust can be a valid objection to the original scheme. The scheme presented in this section will examine how we can reduce the amount of required trust in the central servers and instead rely on potential trust in (some of) the other participating parties. This allows us to still provide security guarantees even in cases where the servers do not fulfil the trust assumptions. The concept is based on the idea that a client's data can still be private if it has previously communicated with one or more honest clients, even though the servers - or server - themselves have all been compromised. This approach might be beneficial in a context where building a trust relationship with clients is reasonably easy, a scenario we will examine in chapter 5.

4.1. Construction Idea

Making use of this additional trust assumption requires us to make two modifications. The first of which is a shift of architecture. While Prio follows a Client-(Distributed Server) Architecture, this approach is based on a combination of the (Client Cluster)-Server Architecture and Client-(Distributed Server) Architecture. The role of the central servers will closely resemble their role in the original Prio scheme, but instead of individual clients submitting their data to the central servers, clusters of clients will calculate a pre-aggregate and submit this intermediate result to the central servers. This construction limits us to associative functions, even though the original Prio scheme allowed non-associative functions as well, albeit with varying levels of performance. To calculate the pre-aggregate the clients follow a scheme similar to the original Prio.

The assumptions made by Prio about the servers become unfeasible when made about the clients. To provide robustness, Prio relies on all servers being honest. Since servers are usually not anonymous it is quite easy to evict a server from future participation in the scheme, which prevents them from gaining access to future aggregation results. While Prio does not provide any tools to detect maliciously acting servers, it is still possible that such a server could be noticed, for example, by humans working with the aggregation results. Having a chance of being detected when acting maliciously provides an intrinsic motivation for the servers not to actively behave malicious. Clients on the other hand usually lack such an intrinsic motivation, especially in cases where they can remain fully anonymous and where any repercussions for

the misbehaving party from the scheme can easily be mitigated by the misbehaving party by introducing a new client into the system. This idea is based on the covert adversary model (Definition 20).

To counteract this issue we different techniques have already been examined, such as verifiable secret sharing or Zero-Knowledge proofs in distributed systems [6]. For this protocol we will make use of Shamir's threshold secret sharing scheme [50], as a drop in replacement for the additive secret sharing in the original Prio scheme. This will be our second modification. A threshold secret sharing scheme allows us to fine-tune the relation between how many clients need to behave honestly for the pre-aggregation to be robust versus how many clients need to collude to be able to unveil secret private data.

4.2. Modifications

In this section we will describe the modifications made to the original Prio protocol in detail, and go over the drawbacks and benefits each one provides.

4.2.1. SNIPs with Shamir's Secret Sharing

Before we can replace the additive secret sharing used in Prio's SNIPs with Shamir's secret sharing, we have to make sure that Shamir's secret sharing still provides the prerequisites needed for the security guarantees of SNIPs. To do so we will revisit the construction of SNIPs (Construction 4) as seen in Prio, review every step, re-examine all the arguments where properties of the underlying secret sharing scheme were used, and verify that any other assumptions we may have made still hold for our improved version of the scheme.

Step 1: Client evaluation Step 1 is executed on the client submitting the data and is primarily concerned with creating shares of the polynomial $h = f \cdot g$. This is done by evaluating the Valid$(\cdot)$ predicate on the client's secret value x, followed by polynomial interpolation on the inputs of the multiplication gates to construct polynomials f and g. The client then splits the coefficients of the polynomial h into shares and distributed them to the servers. To prevent a dishonest client from creating shares in a way that is not intended by the protocol, such as not choosing a polynomial of degree $\leq t$, we will introduce a verification mechanism later in this section.

The process of creating shares does not yet use any properties of the additive secret sharing scheme.

Step 2: Consistency checking at the servers This step is executed by each server independently and ends with each server having shares not only of the polynomial $\hat{h}$, but also shares of the polynomials $\hat{f}$ and $\hat{g}$. If every party acted honestly up to this point the shares of these polynomials are identical to shares of the polynomials f, g and h. Remember that $\hat{h}$ is the polynomial whose shares a client actually submit, which might differ from shares of h if the client behaves dishonestly. The shares of polynomials $\hat{f}$ and $\hat{g}$ are the shares derived from the polynomial shares submitted by the client, $\hat{h}$. To do so we argued that the knowledge of a share of x and the corresponding share of $\hat{h}$ is enough to reconstruct shares of all wire values

in the arithmetic circuit that represents Valid(·). This argument only requires affine operations on shares, which is still possible with Shamir's secret sharing, as it is a linear scheme. Using the shares of the wire values it is possible to use polynomial interpolation, as in step 1, to construct shares of the polynomials $\hat{f}$ and $\hat{g}$.

Step 3a: Polynomial identity test Step 3a is the first step that requires the servers to communicate with each other, which is needed to confirm whether $\hat{f} \cdot \hat{g} = \hat{h}$ actually holds. Remember that this equation holding is equivalent to the servers holding wire shares that, when reconstructed, equal the wire values of the Valid(x) circuit computation and is only possible if $\hat{h} = h = f \cdot g = \hat{f} \cdot \hat{g}$. Considering that we operate on the coefficients of the polynomial and that, again, Shamir's secret sharing is linear as well, this property still holds. Following this the servers execute the Schwartz-Zipple randomized polynomial identity test [48, 66] to check whether this relation holds. This check requires a random value $r \in \mathbb{F}$, which could be randomly sampled by one of the servers in the original construction. Since this is not so trivial in our case we will for now assume that we do indeed have such a random value r and take a closer look at this in the following section (subsection 4.2.2). The rest of step 3a does not deviate from the original construction, except that we need to substitute the additive secret sharing's reconstruction algorithm with the one from Shamir's secret sharing. This allows us to verify whether the shares of the polynomial $\hat{f}(r) \cdot \hat{g}(r) - \hat{h}(r)$ do indeed reconstruct to the zero polynomial $0 \in \mathbb{F}$.

Step 3b: Multiplication of shares Step 3b introduces Beaver's Trick, which is used to work around the issue of multiplicating shares. As we we have already seen the requirements of Beaver's Trick (subsection 2.2.4) we can easily verify that that both additive secret sharing and Shamir's secret sharing can be used with Beaver's Trick.

Step 4: Output verification In the final step it is argued that, since each server already has the shares for each wire in the circuit, they simply have to publish the share of the output wire. Once enough servers have published this share the output of the Valid(x) predicate can be reconstructed.

4.2.2. Sampling a Random Value

The original Prio solves the problem of generating a random value $r \in \mathbb{F}$ by having a single server sample it by itself. While the idea behind this approach and its implications have already been discussed previously (paragraph 2.2.3), the same reasoning can no longer be applied once a client is doing the sampling. The Schwartz-Zippel randomized polynomial identity test [48, 66] requires all the participating parties to unanimously agree on the value of r, which becomes difficult in a cluster where clients may drop and stop responding at any given time without prior announcement, especially so when parties actively trying to impair the aggregation are present.

Before we describe a possible protocol for requesting a random value from one of the servers, we need to discuss the requirement of such a protocol:

Share-before-Randomness Since the random value r will be used for polynomial identity testing, all participating clients need to publish their shares before r can be requested by any one client. This is required to make sure that no polynomial can be chosen depending on r, which would allow a malicious client to subvert the polynomial identity test and make it possible for this client to submit data, that does not adhere to the Valid predicate.

4.2.2.1. Centralised Sampling

Multiple approaches exist which enable clients in any given cluster to receive the same random value from the central servers. In the most trivial of setups all clients across all clusters request their random value from the same server, which allows us to embed the identity of this server in the protocol itself. While this approach is independent from the method in which the clients are clustered, it also comes with some undesirable implications regarding privacy and accessibility .

In cases where the clustering (see section 5.1) is done by the central servers, it is might also be possible for the servers to assign a designated sampling server to each cluster to be used by clients in that cluster. If the clusters are created without aid of the central servers, the clients can use a cluster identifier, uniquely identifying to which cluster they belong, such as a public key already shared between the clients, or a membership function , as input for a function that maps to one of the central servers.

If we assume that each participation client within a cluster has access to an additional, not necessarily secret, regularly changing value, that is always known to all clients in the cluster, (such as a reasonably accurate clock) this would empower us to use a mapping that relies on the cluster identifier and this additional value. As a result the server from which the clients request their randomness is no longer static but becomes dynamically changing. This, depending on the scenario (section 5.1), might make it considerably more difficult for a malicious server to cooperate with a malicious client.

4.2.2.2. Distributed sampling

If we want to remove the centralised servers from this part of the protocol entirely, a different approach is also possible, albeit requiring more compute and communication among the clients. For this approach each participating client samples its own random value r and publishes a commitment using a reliable broadcast. Following this the clients can publish their random value, aggregate it with the random values of all the other clients and use this aggregate as the random value. This guarantees that all participating parties share the same value r and that r is indeed random, if at least one client sampled it randomly from $\mathbb{F}$. Implementing this alternative comes with its costs however. Each participating client needs to perform a reliable broadcast, we have to consider what happens with clients that don't respond or respond too late and pay more attention to clients that simply drop out, which may force us to start the sampling process anew. This process may take multiple rounds and is, depending on our assumptions, not necessarily more secure than requesting a random variable r from the servers, which is the reason why this idea will not be part of the final scheme unless stated otherwise.

4.2.3. Prerequisites

As with every (Client Cluster)-Server Architecture we require clusters of clients to execute this scheme. Reasonable ways to create clusters vary widely with the context the scheme is deployed in, as such we will be discussing this step in section 5.1.

4.2.4. Malicious Secret Shares

Client based Prio faces two issues when working on shares that the original Prio does not need to consider. This is due to the fact that clients in the original Prio are never in contact with shares from other clients, while other clients handling shares is a core concept of client based Prio. Both issues arise from dishonest clients submitting pre-aggregate shares to the servers. A central server receiving a share from a client has no way of knowing whether the share was generated honestly or maliciously, given that an honest share is statistically indistinguishable from a malicious one. This problem can be solved by using a verifiable secret sharing scheme, such as Feldman's scheme, which allows the server to verify that the shares can be used to reconstruct a well-defined secret. While this does force clients to submit valid shares (or be detected otherwise), it does not prevent them from correctly sharing an incorrect value.

Sharing an incorrect value is not only an issue when submitting shares to the servers, but also during the evaluation of the Valid predicate. By using incorrect shares on purpose during the distributed computation of the Valid predicate, a sufficiently large group of malicious clients can force the protocol to discard the submissions of honest clients. While we cannot necessarily determine which clients submitted incorrect secret shares, we can use existing techniques, such as Reed-Solomon codes, to at least detect if incorrect shares have been submitted at all, that is to verify whether all shares describe the same polynomial. Since the construction and evaluation of Reed-Solomon codes are expensive operations, we can combine this with an initial and more intuitive approach that is fast to execute. Even though the intuitive approach does not scale well with an increasing number of incorrectly submitted shares, it does allow us to determine whether incorrect shares are present at all.

4.2.4.1. Intuitive Verification

The intuitive approach takes its inspiration directly from Shamir's (t, n)-threshold secret sharing. To uniquely define a polynomial of degree $t - 1$, t pairs of (x, y)-coordinates are always sufficient and required, and by construction all of the n Shamir secret shares describe a point on this polynomial. It is easy to see that any subset of t shares necessarily describe this polynomial, as such all polynomials described by any subset of t shares are identical.

To check whether a subset of $t + 1$ shares contains (at least) one incorrect one, we can use the first t shares to construct a polynomial and simply test whether share $t + 1$ is also a point on this polynomial. Conceptually this is easy, since shares are simply coordinates in the (x, y) plane. Using the first t shares to reconstruct a polynomial and iteratively testing whether a different share is on the same polynomial allows us to verify whether any subset of at least $t + 1$ shares contains one or more incorrect shares.

Depending on the parameters t and n and the largest number of incorrect shares e we expect we can further check for incorrect shares using this intuitive approach, such as grouping the

shares into $e+1$ pairwise distinct sets of size at least t. In this case at least one set must contain no incorrect share.

Given that polynomial interpolation and testing whether a given point is on that polynomial can be achieved using only addition and multiplication over $\mathbb{F}$, this can initial step can be achieved using multi-party computation with arithmetic circuits.

4.2.4.2. Reed-Solomon Codes

A more elaborate technique for detecting errors are *Reed-Solomon* codes [47]. A Reed-Solomon code takes as input t finite field elements, called symbols, and add adds $n-t$ check symbols to the data. This allows a Reed-Solomon code to detect (but not reconstruct) any combination of up to $n-t$ erroneous symbols, or to locate *and* correct up to $\lfloor \frac{n-t}{2} \rfloor$ symbols. The relatedness between Reed-Solomon codes and Shamir's secret sharing has already been described by McEliece in 1981 [38]. Interpreting shares generated by Shamir's secret sharing as a Reed-Solomon code allows us to either detect or to detect and locate maliciously submitted shares, but require a much more expensive multi-party computation.

4.2.5. Output Privacy

Client based Prio provides exact results. This means that the output is precise, deterministic and free from any error or noise. As a consequence it does not provide any form output privacy, which would prevent an attacker from deducing information about the secret input from only the aggregation result. To increase output privacy we can additionally introduce noise to the output, which is usually done using a differential privacy mechanism. For now we assume high-dimensional inputs, where each dimension can be represented using a whole number. These assumptions allow us to use a simple Gaussian mechanism, in cases where lower-dimensional inputs are used or floating point precision is required more complex mechanisms might be a better choice. Keller et al. provide a collection of different mechanisms [31] that can be used instead of the Gaussian Mechanism when required.

Guaranteeing output privacy in this scheme can be achieved by adding a differential privacy mechanism at one of three stages, where each stage provides a different balance between quality of the aggregation result and privacy guarantees.

4.2.5.1. Client

The first stage where noise can be added is by the client itself. While adding noise before submitting the data appears to be independent of the scheme that will process the data, this is only partially true. Whatever value is provided by the client still needs to be accepted by the Valid predicate. If noise is included before the validity of an input is checked, the validity check must account for that noise. This necessitates a reduced strictness of the Valid predicate, which allows a dishonest party to submit more "extreme" values.

Adding noise before even submitting data makes it impossible for anyone to reconstruct the original secret value with certainty, but also leads to a very high noise in the aggregate and therefore a low result quality.

4.2.5.2. Client Cluster

In their paper Keller et al. also provide instructions on how each party in a multiparty computation protocol can locally sample a noise share, with the restriction that these noise shares reconstruct to one single instance of Gaussian noise [31]. These noise shares can then be linearly combined with the local aggregator before each client secret shares their aggregator with the central servers. Adding noise using this proposed construction requires that no client that is part of the cluster drops out during aggregation, since this would lead to less than the intended amount of noise in the pre-aggregate. The generation of noise shares is relatively cheap compared to the other steps required for the pre-aggregation, as such it is feasible to compute the noise shares before continuing with the pre-aggregation steps. This allows us cheap and early restarts of the aggregation protocol if a client drops out even before pre-aggregation.

Assuming that the assumption holds, that at least t out of the n clients behave honestly, this is a secure way of adding noise. We also benefit from a reduced amount of uncertainty in the final aggregation output, since only one instance of noise per cluster is added, instead of one instance of noise per client.

If restarting the protocol is not a desired option we can also use the proposed MPC noise share generation protocol to add more then one instance of noise per cluster. Designing the protocol to add $\lceil \frac{n}{t} \rceil$ instances of noise increases the uncertainty again, but also guarantees that at least one instance of total noise is added, if at least t clients behave honestly.

As the most interesting stage, we will add noise at this point.

4.2.5.3. Central Servers

The last stage where noise can be added is before the servers publish the aggregation result. It is obvious that adding noise at this stage leads to the aggregation result with the highest quality, since only a single instance of noise has been added. While a single server adding noise and the distributed noise generation MPC protocol adding noise are undistinguishable, the MPC protocol is strictly preferable, since it prevents any single server from seeing the aggregation without added noise. The drawback however is the required trust in the servers. Whereas adding noise at any of the previous stages required no trust in the servers at all, adding noise at this stage would allow the servers to construct an aggregation result without any noise included if they all worked together.

4.3. The Scheme

Now that we have seen all the required preliminaries, we can return to describing the improved scheme. As we mentioned earlier the clients have to be grouped into clusters, before the scheme can start aggregating data, this is done according to the chosen selection mechanism.

Pre-aggregation phase In the pre-aggregation phase we mostly follow the Prio scheme as described in subsubsection 3.2.2.2, except that we also implement these differences:

Clients are Servers Instead of dedicated servers, each client now also acts as a server for the cluster it is assigned to. This includes doing all the computation and communication that the dedicated servers used to do.

Neutral clients Unless a client is guaranteed to have a secret value every time data gets aggregated, we have to consider scenarios where this is not the case. For aggregation functions that have a neutral element those clients can instead choose to submit this element. If a neutral element does not exist it is also possible for a client to submit an invalid proof for the Valid predicate, so that it does not evaluate to 1. In case we require additional information about the number of clients not wanting to submit a data point it is also a valid option to extend the Valid predicate with an additional output value, such as 2, that a client can force as a result if it so desires. Since an output of 2 still leads to the other clients (servers) discarding the input, this change does not make it easier to submit an invalid input. Depending on the effort needed to create the clusters it might also be a feasible approach to forego this problem by simply creating new clusters for every aggregation.

Adding noise To provide output privacy we introduce a MPC protocol that adds noise to the input values, so that the added noise summed over all clients represents a Laplace mechanism. This is based on [31].

SNIPs with threshold secret sharing Instead of using SNIP with additive secret sharing as presented in the original paper, the clients (servers) instead use Shamir's secret sharing.

Resharing and Forwarding The clients now hold shares of their cluster's pre-aggregate, which still use Shamir's (t, n)-threshold scheme. This means that simply forwarding the shares to the central servers is not guaranteed to work: While any client might have at least t shares, doing calculations on shares requires shares with identical indices. Considering this, each client reshares its share of the pre-aggregate. Since the central servers are unlikely to drop out mid aggregation we can also switch to an (n, n)-threshold scheme. To allow servers to check for consistent shares we use a verifiable secret sharing scheme, such as Feldman's scheme [22]. To do so each client uses Shamir's secret sharing reconstruct algorithm and uses the new secret sharing scheme to generate one share for each of the central servers and computes the verifiable secret sharing scheme's *encrypt* function to receive Y. The client then sends Y and one share to each of the central servers.

Aggregation phase The central servers now posses shares of the pre-aggregates from all clusters. Using the corresponding Y for every received share, the servers can locally confirm the validity of all their shares . These shares can then be used as the input values of an MPC protocol that performs polynomial interpolation and outputs the final aggregation result. The next subsection introduces such an MPC protocol based on Lagrange interpolation.

Lagrange Interpolation Examining the example in Figure 4.1 , we notice that the central servers cannot trivially reconstruct the aggregation result. When a client uses Shamir's secret sharing to submit its secret data it generates a set of (x_i, y_i) coordinates where the values of all x_i are pre-defined. Given the linearity of Shamir's secret sharing it is easy to see that the clients can trivially combine received shares to compute a share of the pre-aggregate, as we have already seen in .

Once the clients secret share their pre-aggregate share with the central servers however, these new share no longer represent coordinate on the polynomial required to reconstruct the aggregation result, but instead a shares of the y_i values at position x_i. This prevents the servers from trivially interpolating their shares to compute the aggregate, we could imagine this situation as a "type mismatch".

Creating a Lagrange basis allows us to efficiently compute the aggregate nevertheless.

Since Shamir's secret sharing is a linear scheme adding all the shares from a single cluster yields the shares of this cluster's pre-aggregate. Using the shares of the pre-aggregates we can now compute the final aggregation result. To do so we have to re-share the Shamir shares, which in turn enables us to linearly combine those shares to compute the aggregation result.

4.3.1. Pseudocode

The following pseudocode summarises how client based Prio works. To improve readability the pseudocode has been split into multiple blocks, representing a "phase" of the protocol. Algorithm 9 initialises the protocol, algorithm 10 is taken from to show how a distributed Laplace Mechanism can be implemented, and could be replaced by a mechanism more suitable for a given scenario. Algorithm 11 describes the Pre-Aggregation of the client based Prio protocol, responsible for sharing the secrets within a cluster, followed by algorithm algorithm 12, which outlines how the central servers can calculate the final aggregate.

Algorithm 9: Client Based Prio: Initialisation

Input: Set of m clients,
grouped into l clusters $\{ C_1, \ldots, C_l \}$ according to clustering constraints in section 5.1,
where the i-th client in cluster j has secret input x_i^j,
Number of servers n,
Aggregation function f,
Valid predicate Valid$(\cdot)$,
Threshold parameter t for Shamir's secret sharing

Output: Aggregation result $f(X)$
where $X = \left\{ x_1^1, \ldots, x_{m_l}^l \right\}$ is the set of all secret client inputs,
$|C_i|$ the number of clients in cluster i

```
/* Central Server Aggregator Initialisation                    */
foreach Central Server j do
    S_j := 0;
/* Client Aggregator Initialisation                            */
foreach Cluster k do
    foreach Client i do
        /* Initialise the aggregator of the i-th client in cluster k   */
        A_i^k := 0;
```

Algorithm 10: Client Based Prio: DP Mechanism

foreach *Cluster* k **do**
  foreach *Client* i **do**
    /* Generate and add a noise share using an appropriate mechanism from [31] */
    $A_j^k := A_j^k + generateNoise(\cdot)$

Copy from paper, here or in preliminaries?

Algorithm 11: Client Based Prio: Pre-Aggregation

/* Sharing and Pre-Aggregation */
foreach *Cluster* k **do**
  /* Shamir's Secret Sharing */
  foreach *Client* $i \in C_k$ **do**
    $\{[x]_{i,1}^k, [x]_{i,2}^k, \ldots, [x]_{i,|C_k|}^k\} \leftarrow \text{ShamirSecretShare}_{(t,n)}(x)$;
    $\{[\pi]_{i,1}^k, [\pi]_{i,2}^k, \ldots, [\pi]_{i,|C_k|}^k\} \leftarrow \text{SNIP}_{(t,n),\text{Valid}(\cdot)}(x)$;
    foreach *Client* $i' \neq i \in C_k$ **do**
      Submit $[x]_{i,i'}^k$ and $[\pi]_{i,i'}$ to client i';

  /* Pre-Aggregate Receive and Verify */
  foreach *Submission* $[s]_i^k = \{([x]_{i,1}^k, [\pi]_{i,1}^k), \ldots, ([x]_{i,|C_k|}^k, [\pi]_{i,|C_k|}^k)\}$ **do**
    $valid := \text{VerifySNIP}_{(t,n),\text{Valid}(\cdot)}([s]_i^k)$;
    if $valid \neq 1$ **then**
      Reject submission $[s]_i^k$;
    else
      foreach *Client* i' **do**
        $A_{i'}^k := A_{i'}^k + [x]_{i,i'}^k$;

Algorithm 12: Client Based Prio: Central Collection and Aggregation

foreach *Cluster* k **do**
  foreach *Client* $i \in C_k$ **do**
    $Y, \{[A]_{i,1}^k, [x]_{i,2}^k, \ldots, [A]_{i,n}^k\} \leftarrow \text{FeldmanSecretShare}_{(n,n)}(A_i^k)$;
    foreach *Central Server* j **do**
      Submit Y and $[A]_{i,j}^k$ to central server j;

foreach *Central Server* j **do**
  foreach *Submitted Unique Secret* Y *and Share* $[A]_{i,j}^k$ **do**
    $valid := FeldmanCheck(Y, [A]_{i,j}^k)$ **if** $valid \neq 1$ **then**
      Reject submission $[A]_i^k$;

4.3.2. Properties

Communication The communication overhead consists of (number of cluster) Prio runs and one message from each of the clients to every server.

Robustness Client based Prio provides robustness against malicious clients using the Valid predicate. If all clients but the client submitting the data are honest it is easy to see that the cluster will only accept the data with negligible probability. Based on our construction using a (t, n)-threshold secret sharing scheme the cluster can correctly compute the Valid predicate for a given input, it at least t clients in any given cluster execute the protocol honestly.

Privacy If at least t out of the n participating parties in a cluster are honest, client based Prio guarantees that no party can learn the input data of a single client, or derive information from it due to the information-theoretic property of Shamir's secret sharing. Therefore client based Prio guarantees privacy even in situations where all central servers have been corrupted.

4.4. Example

Before we conclude with a detailed explanation on how to reconstruct the aggregation result of the Client Based Prio aggregation scheme, we will first describe an example run of the protocol using (n, n) Shamir secret sharing (Construction 3). In this example we simply wish to calculate the sum of the client's input data over the finite field $\mathbb{F}_7$. Our example consists of a single cluster with three clients, one of which, Client C_1, does not submit data but partakes in the calculation of the pre-aggregate, and two servers. This example uses the bottom half of Figure 4.1 as reference.

Notation Each client has knowledge of a secret polynomial $q_j(x)$, where the index j references the name of the client. We use lowercase letters to refer to the coefficients of the Shamir polynomial of a single client, where a_i belongs to the term x^i:

$$q_j(x) = \sum_{0}^{m} a_i x^i.$$

This also means that $a_0 x^0 = a_0$ represents the secret input of client A.

Input Each client is initialised with its secret input that it wishes to submit to the aggregation protocol. We assume the following inputs:

$$a_0 = 2$$
$$b_0 = 3$$

It is easy to see, that the result of this aggregation should equal $5 \in \mathbb{F}_7$.

Creating Client Shares Before the clients can create shares of their secret, they need to create a secret polynomial according to the restraints of Shamir's secret sharing, meaning each client needs to randomly sample to coefficients, which could result in these polynomials:

$$q_A(x) = 2 + 5x + 6x^2$$
$$q_B(x) = 3 + 3x + 0x^2$$

Each client now evaluates its polynomial at $x \in \{ 1, 2, 3 \}$, keeps one of the shares and sends the other two to the remaining clients in the cluster. At the end of this step client A has all the shares of the three polynomials where $x = 1(q_A(1), q_B(1))$, clients B and C have the shares where $x = 2$ and $x = 3$ respectively. At this point a share still represents a point on one of the polynomials:

$$q_A(1) = 6 \qquad q_A(2) = 1 \qquad q_A(3) = 1$$
$$q_B(1) = 6 \qquad q_B(2) = 2 \qquad q_B(3) = 5$$

Sharing the Shares Each client now combines the shares it has received. Taking all the clients into account this is akin to computing the sum of the all polynomials of the clients together. This means that each client now holds a single share ((x, y) point on the polynomial) of the polynomial $r(x)$, of which the trailing term $r(0) = r_0$ is the still hidden aggregation result:

$$r(1) \equiv_7 5$$
$$r(2) \equiv_7 3$$
$$r(3) \equiv_7 6$$

If we were to reconstruct the shares now this would be identical to executing the original Prio protocol.

Since our example uses two servers, the clients now need to share their point on the polynomial by creating a polynomial of degree 1, with the secret input being the coefficient of r known to the server ($r(1) = 6$ for client A, and so on). This may result in the following polynomials, where u_i refers to the polynomial created by client i and the leading coefficient has been randomly sampled:

$$u_A(x) = 5 + 1x$$
$$u_B(x) = 3 + 6x$$
$$u_C(x) = 6 + 2x$$

Using these polynomials each client can now calculate the values of u_i(x) at positions $x \in \{ 1, 2 \}$ and share $u_i(1)$ with server 1 and $u_i(2)$ with server 2 respectively:

$$u_A(1) = 6 \qquad u_A(2) = 0$$
$$u_B(1) = 2 \qquad u_B(2) = 1$$
$$u_C(1) = 1 \qquad u_C(2) = 3$$

It is at this point we realise that the servers do not simply have shares of a single instance of Shamir's secret sharing, but shares of shares of the original values, which is why the reconstruction using Lagrange polynomials is necessary. We will investigate how reconstruction using Lagrange polynomials works in the following section.

4.5. Lagrange Interpolation

To conclude this chapter about Client Based Prio we will demonstrate how the servers can reconstruct the aggregation result and proof the correctness of this method. In addition to the notation used in the example above (section 4.5), we will use L^C when referring to Lagrange basis polynomials calculated by the clients and L^S when referring to Lagrange basis polynomials calculated by the central servers.

Examining the combined information available to the clients after the pre-aggregation phase, the final aggregation result could be computed using the following equation:

$$r_0 = \sum_{j=0}^{m} L_j^C u_j(0)$$

where L_j^C form the Lagrange basis and $u_j(0)$ is the secret of client j. We now claim that the desired aggregation result $L^S(0)$ also equals

$$L^S(0) = \sum_{i=0}^{n} S(i) L_i^S(0),$$

with

$$S(i) = \sum_{j=0}^{m} L_j^C(0) u_j(i),$$

where $S(i)$ is only known to server i, due to the required knowledge of shares $u_j(i)$.

Using the definition of Lagrange polynomials and simple arithmetic transformation we can now show that r_0 and $L^S(0)$ are indeed equal:

$$\begin{aligned}
L^S(0) &= \sum_{i=0}^{n} S(i) L_i^S(0) \\
&= \sum_{i=0}^{n} \left(\sum_{j=0}^{m} L_j^C(0) u_j(i) \right) L_i^S(0) \\
&= \sum_{j=0}^{m} \left(\sum_{i=0}^{n} L_i^S(0) u_j(i) \right) L_j^C(0) \\
&= \sum_{j=0}^{m} u_j(0) L_j^C(0).
\end{aligned}$$

□

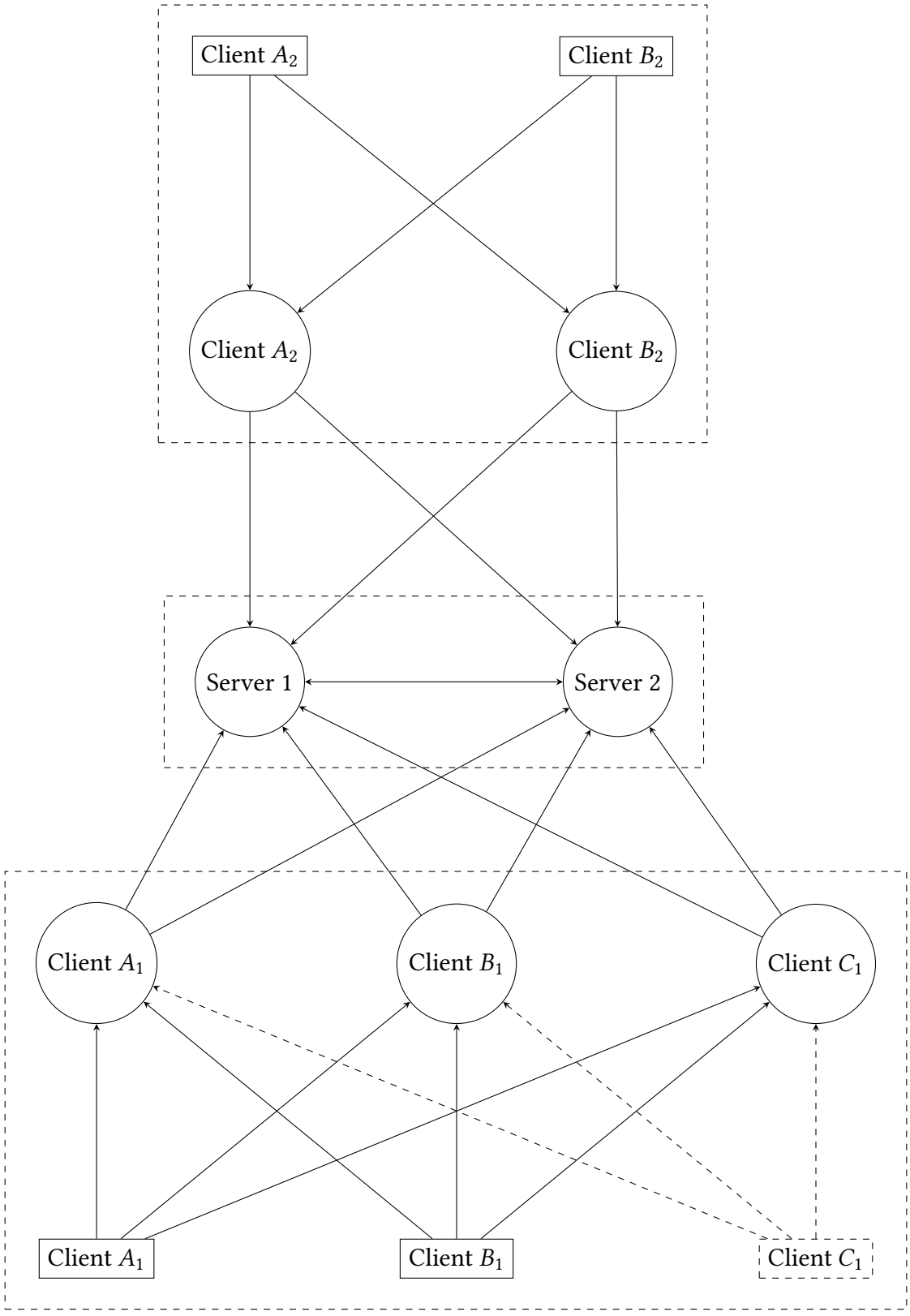

Figure 4.1.: Client Based Prio: Example

A Client Based Prio setup consisting of two central servers and two clusters.

5. Evaluation

What remains is a comparison of the strengths and weaknesses of the presented schemes in different real world scenarios, as well as an assessment of an ideal scheme to examine the underlying problems shared by any approach that aims to aggregate date in a privacy friendly manner.

5.1. Scenarios

We start this section by introducing four different scenarios where privacy-preserving data aggregation might be used in practice. Each scenario focuses on different aspects, such as trusts assumptions and useful aggregation functions, which makes for an interesting comparison in the following section. The diverse nature of potential applications often requires a tailored approach that takes into account the unique challenges faced in this instance. Major challenges include client spoofing, which might be trivial if the client is simply a piece of software running on a mobile device, or requiring significant effort if the client code is bound to an expensive piece of hardware, such as a car and clustering, that is finding a privacy-preserving process that can be used to group individual clients into clusters.

5.1.1. Car Manufacturer

The first scenario we will describe is that of a car manufacturer that wishes to aggregate data from their cars. This might be especially relevant for self-driving cars that already include a large number of on-board sensors which are already collecting data. Since such a car is usually an expensive item that will be used for a considerable amount of time, a car manufacturer might be more inclined to adequately prepare and equip the cars to partake in privacy-preserving aggregation, as to motivate buyers to not opt-out from data aggregation. This preparation could include requesting a digital certificate from a trusted entity, such as a certificate authority (CA), for each unique car produced. This certificate is tied to an individual car and cannot be shared among different cars. If the car manufacturer additionally requires the certificate authority to use certificate transparency (CT), individual clients are empowered to monitor all issued certificates and efficiently detect a dishonestly acting certificate authority [34].

Having a certificate linked to an expensive physical item in this manner makes it difficult for an attacker to spoof cars by creating “virtual” cars.

A different benefit provided by physical items is their persistence. While new cars get produced and old ones destroyed, this process is usually slow and well documented. According to a study published in 2021, the mean lifetime of a car in Western European countries is slightly above 18 years [27]. This means that the rates of new cars appearing or old cars disappearing are quite small and not something where too much attention during data aggregation is required.

A car manufacturer can also include heuristics as to whether a car will stay online for the entire duration of the aggregation scheme. While we will not investigate how an adequate heuristic may work, assumptions such as expecting a car going above a certain speed to not simply turn off and permanently disconnect appear to be reasonable. This advantage however comes with the downside that a car's network connection is exclusively via mobile networks, and as such is highly dependant on its current location, which may prevent a continuous connection. As such this scenario lends itself well to aggregation schemes that tolerate short disconnects of clients, but may in turn require an increased amount of communication rounds.

5.1.1.1. Use Cases

We can identify three main types of cars that are currently used across the world. These are *Internal Combustion Engine (ICE)* cars, *electric* cars and *self-driving* cars. While some of the data we are interested in is shared between different types of cars, we can also identify use cases unique to any one car type. If a manufacturer can guarantee and not only promise that the private data of a customer's car stays secure, and that even the manufacturer cannot access this data, customers might be more inclined to help the manufacturer in calculating the aggregation.

Telemetry The first point of interest common to not only car manufactures is the collection of telemetry data. For an ICE car this might be measurements on engine performance, fuel efficiency and general mechanical wear. Having detailed information on how real-world driving impacts those measurements allows the manufacturer to optimise their cars for real-world usage based on data from more than a select few cars. Telemetry for an electric vehicle might include metrics regarding battery health such as battery degradation over time, charging patterns and energy consumption. This information provides the manufacturer with the information to improve their battery management systems (BMS) and extend battery life across all models. A manufacturer can also use this knowledge to optimise scheduled maintenance by better determining which parts, on average, require closer inspections. Self-driving cars come with a plethora of sensors preinstalled, most of which can be used for some kind of telemetry. What really differentiates self-driving cars though is the AI based decision-making. Aggregation functions can be used to find and closer evaluate situations where humans have to manually override the car's decision, which can lead to improved decision-making in future iterations.

Traffic Patterns Privacy-preserving aggregation can also be deployed to investigate traffic patterns to identify common routes and congestion points without identifying individual cars and drivers. Sharing this insight with urban planners creates possibilities to improve the infrastructure, which can bring benefits not only to drivers, but also to cyclists and pedestrians. Infrastructure improvements include positioning of (new) charging stations for electric vehicles and modifications to the road network to prevent congestion, especially in areas with a large number of cyclists or pedestrians. Advances sensors available in some cars can be used to find inaccuracies or errors in digital maps by detecting a spike of inconsistencies between the maps and the actual driving patterns in a given region.

Sharing traffic patterns with the manufacturer is an especially sensitive topic, since it includes location information possibly paired with accurate timestamps. Proving a method that provably separates this data from an individual is a huge step forward for user privacy.

5.1.1.2. Clustering

For most car manufactures public estimates on the number of produced and sold cars are readily available ([21]). While the numbers may not be exact or up to date, they still provide an accurate estimate. If we account for sufficiently large estimate for error we can guarantee that any subset of cars that is larger than this estimate contains as least one "real" car.

Considering the above point clustering the cars becomes a manageable task. Since spoofing cars is difficult even for the manufacturer, we can rely on the manufacturer to create the clusters for us, as long as the cluster size is sufficiently large.

5.1.1.3. Schemes

The increased control over the hardware and software, and by extension trust, a car manufacturer has regarding a car sold by them could allow them to more heavily rely on client based input verification only. It is conceivable for multiple car manufacturers to work with each other to aggregate data in an honest manner, or for an honest, independent third party to exist that helps during aggregation. Using publicly available information the number of cars on the roads can be estimated reliably.

Homomorphic Encryption While the lack of external input validation is an obvious drawback, depending on the amount of trust and the level of precision needed, a homomorphic encryption based approach might be feasible. The homomorphic encryption based scheme requires only one message from the client to the server, which can be beneficial in some settings, but given the ever increasing network coverage this is unlikely to be an important consideration.

Additive Secret Sharing Additive secret sharing based schemes deviate from homomorphic encryption based schemes in one key factor. The car is now required to send one message to each server. This increases network usage but still requires only one round and should generally be possible without issues. This drawback effectively buys a 1 in n threshold, instead of the 1 in 2 threshold for guaranteed privacy in the homomorphic encryption based scheme.

Prio Prio allows external input validation, bought with an increased computational overhead on the client and the servers and an increased network traffic between the servers. Whether this validation is needed depends on the increased resource investment in implementation and care compared to the expected improvement in the aggregation result. It is otherwise identical to additive secret sharing.

Client based Prio With the available information, we can rely on the car manufacturer to create the clusters, and given the mostly static pool of cars there is also no direct need to create new cluster regularly. If *error* is an upper bound for an estimate regarding how many cars exist on paper, but not in the real world, a cluster size of $2 \cdot error$ guarantees that at least half the cluster consists of real cars. A drawback is the increased network traffic between cars, which is required for the pre-aggregation step.

5.1.2. Browser Installations

The next scenario we consider is a stark contrast to the previous one. Whereas cars as clients were mostly persistent and could be seen as (semi-)honest for the most part, using (running) browser installations as clients paints an entirely different picture. This scenario describes clients that are highly dynamic, untrusted, but computationally capable at the same time.

Internet browsers are used by most humans with access to the internet on an almost daily basis for a plethora of different tasks, such as reading the news, connecting with friends via social media, shopping for new home appliances or simply using a search engine to find relevant information. This information can be used to create highly detailed user profiles [46, 7, 55, 62] which are only becoming more and more valuable to certain companies, such as advertisement agencies.

The practice of creating these profiles is in direct conflict with a user's interest of privacy. Using a data aggregation scheme that allows the aggregating of user data without compromising their privacy allows for users to preserve their privacy while still providing useful statistics.

While the above use case is seen as questionable by many, a browser may also collect statistics regarding performance metrics or error handling, which can be used to fix security and software flaws in the browser.

On the other hand aggregation schemes deployed in this scenario can make use of the high computational capacity provided by a stationary PC or, more generally speaking, of a device with a sufficiently powerful CPU that is not reliant on battery power. This allows us to make use of more cryptographically advanced techniques, that might otherwise negatively impact users.

5.1.2.1. Use Cases

Data collected by a browser can include a large amount of data that is both sensitive and highly valuable. As such many unaffiliated parties are interested in aggregating it. While the interest of any aggregating party might be unique, we can identify some common use cases:

Website Analytics Website owners are usually interested in user behaviour data. This is data that describes how users interact with the website, how much time they spent (time-on-page), which links they click and how they navigate. Evaluating this provides the website owners with the information needed to improve content and general user experience

Ad Metrics Advertising agencies rely on metrics such as impressions (the number of displayed banners on a website), clicks and conversion rates to estimate campaign effectiveness. Advertisement agencies are often paid by the amount of traffic they generate for the advertiser. An advertiser is only interested in paying for traffic generated by real humans, so advertising agencies are motivated to provably prevent any type of malicious manipulation, such as clock fraud.

User Preferences Browsers, as any other software, rely on feedback from real users to improve their user experience. This includes statistics about the most commonly used plug-ins or extensions, information about preferred settings, the workflow and errors encountered by the browser's user base. Access to these allows future versions of the software to be optimized for actual real-world usage. While this is a similar use case to *website analytics*, it differs in who is responsible for the data aggregation. A browser itself can be downloaded from a trusted source and its code doesn't change until it is updated. Websites on the other hand are even more dynamic, simply reloading a website could lead to a completely different result.

5.1.2.2. Clustering

Since we have no meaningful way to verify or establish trust between clients, creating clusters in which clients can trust is close to impossible.

5.1.2.3. Schemes

Installing a browser on a personal computing device is a very simple process. While techniques exist that would allow limiting users to one install per device, these techniques are neither commonly used for browsers nor considered in this scenario. This makes it difficult to justify any sort of trust towards a single browser instance. Justifying trust in a cluster of clients remains as difficult considering a malicious entity can create a nearly unlimited number of browser instances.

Homomorphic Encryption Without the possibility of verifying a client's input, schemes build on only homomorphic encryption need to trust the clients to only submit valid data.

Additive Secret Sharing Additive secret sharing based schemes suffer from the same problem as homomorphic encryption based schemes. The lack of an external form of input validation paired with missing trust in the clients makes this an unfitting scheme.

Prio The added input validation, which makes Prio not require trusting the clients, makes Prio a good candidate for this type of scenario. In 2018 Mozilla, the company behind Firefox, experimented with deploying Prio for privacy-preserving telemetry aggregation [28].

Client based Prio Requiring trust in at least a certain number of clients in a given cluster is not a justified assumption in this scenario. If this assumption turns out to be unjustified, client based Prio is not a secure scheme to use.

5.1.3. Mobile Apps (Public Transportation)

For our third scenario we consider the case where clients are applications installed from an app store on a mobile device. To somewhat restrict this scenario we focus on applications provided by a transportation company to its customers. Mobile applications as clients share a few properties with both of the previous scenarios, while also requiring us to find unique solutions to a new set of problems. As with the browser installation scenario above, mobile applications as clients describes a highly dynamic, untrusted, but computationally less capable scenario.

Applications offered by a public transportation company provide a set of different functionalities, which often include routing between different stations and buying tickets to board the buses and trains.

While this type of information about a single person could potentially allow the transportation company to create a detailed profile of this person, simply by knowing where a person goes at which day in the week. On the other hand the transportation company could also use this data to better plan out future routes and investigate where additional connections are required to satisfy demand. If this type of aggregation could be done live it might also be possible for the transportation company to satisfy demand on demand.

It is difficult to make exact predictions on when the users terminate the application, either by manually killing it, shutting down their phone or letting the operating system suspend the application to save on system resources. Mobile operating systems however, allow applications to run in the background, even if the phone's screen is turned off and the phone inside a pocket. We can combine this observation with the fact that mobile devices also come with a wide variety of sensors to some of which we have full access. If investigate a set of sensors that is available on mostly all devices we can make active use of these during the scheme.

5.1.3.1. Use Cases

Data collected by these applications turns out to be highly valuable for many different use cases. As stated above we will limit ourselves to applications provided by transportations companies. but any company might add more use cases based on their own preferences.

Passenger Count Counting the number of passengers that are currently in a given vehicle allows a transportation company to better estimate future demand for this section of the network. Efficient and anonymous schemes fulfilling this need already exist, but it still remains a valid use case that any scheme presented here should support in some way of form.

Common Destination The transportation company might also be interested in common destinations, split across different target groups. Knowing where people exit the network allows a higher focus on a fewer stations, instead of a spread focus on more station when it comes to improving the infrastructure. A different target group are the people currently inside a train on the network. Finding out the most common exit stations for these people allows estimates about future demand in these same areas, information which can be used to extend the infrastructure, if so required, by opening a new connection between two stations or. If our scheme supports output privacy it also becomes possible to generate full histograms to answer these questions.

Ticket Type Germany introduced the "Deutschlandticket" (*Germany Ticket*) in 2023. Compared to other tickets the Deutschlandticket is valid not only with the transportation company from which it was bought, but also in most other short-distance traffic connections across Germany . To receive proportionately subsidies from the German state, the transportation company needs to estimate how many of its customers are using the Deutschlandticket, compared to how many customers are using a different type of ticket. This is currently achieved by an increased number of conductors, but an appropriately designed aggregation scheme could also be suitable to achieve this task.

Intent Without Action The last use case we want to consider in this scenario are intents without action. Intent without actions refers to a customer who searches for or looks at a connection inside the application, but ends up not purchasing a ticket. It is important to remember that not purchasing a ticket is not equivalent to not making the trip since for example monthly passes exist that authorise a person to use the network without paying for a single fare ticket. Solving this scenario requires either information about all the tickets currently in a person's possession and the information whether the trip was made, or employing a heuristic that tries to evaluate this based on similar but different situation.

5.1.3.2. Clustering

Clustering in this scenarios is a difficult task. Having a central server assign a cluster without any client interaction provides the client with no trust guarantees regarding the other members of the cluster. On the other hand it is also not easily possible for the client to discover other clients around it using wireless technologies, such as Bluetooth, since all such technologies require some sort of identification, before a client can request to be grouped with other clients it has discovered locally. However, simply knowing which clients interact enables a server to draw additional conclusions. A study from 2013 using a large sample of Facebook users for example was able to determine with high accuracy which of those were in a relationship, simply based on their connections in this graph [4].

5.1.3.3. Schemes

While installing apps on mobile devices is even easier than installing a browser on a PC, it is likely that a public transportation company implements some sort of verification that limits how many instances a malicious actor can feasibly create. If customers need to link their bank account to purchase tickets, this information can also be used to limit the number of app instances that can be linked to a single bank account.

Homomorphic Encryption Without the possibility of verifying a client's input, homomorphic encryption based schemes remain unfitting.

Additive Secret Sharing The lack of trust makes additive secret sharing based schemes fall short in the same way as homomorphic encryption based schemes do.

Prio Using the same reasoning as in the *Browser Install* scenario, we can see that the assumptions required by Prio match the one provided in this scenario. As such using Prio can be considered in this scenario.

Client based Prio While placing trust in browser instances was not justifiable, placing trust in app instances might be justifiable under certain conditions. Most mobile devices come with Bluetooth, a short-range wireless technology, which can be used to identify devices nearby. Trusting at least some devices your device has been close to can be a well founded assumption. Since app instances are limited it is difficult for a malicious party to create multiple instances, and "cloning" the same instance in multiple locations, such as in a railroad car, makes it easy to notice and exclude if a certain device appears to be too close too often.

5.1.4. Mobile Messaging

For our final scenario we focus on already established trust relationships between subsets of clients. If we can find and interact with such an infrastructure it directly provides the trust assumptions required for many of the previously described data aggregation schemes. As it turns out many popular instant messaging applications, such as WhatsApp [58], Telegram [54], Signal [52] and Threema [23], already provide this functionality. Messages sent between two clients (may) use some form of *end-to-end* encryption , which means the messages are only readable by the two participating parties. While end-to-end encryption does not guarantee that a participating client is who they claim to be, these services allow clients to verify each others identity by comparing, depending on the exact implementation, some kind of shared secret over a trusted medium, such as by meeting in real-life.

Unfortunately neither of these services allows us to extend their infrastructure. Gaining in popularity however is the Matrix protocol [36] , an open protocol for secure decentralised communication. Matrix was build with a focus on security and privacy, offers encryption and also allows clients to verify each other . What makes Matrix unique is that is allows extensions to its infrastructure, which we can then use as basis for our privacy-preserving aggregation scheme.

5.1.4.1. Use Cases

Given the still somewhat niche nature of Matrix and the additional requirement to install an extension, the current use cases for this scenario remain somewhat limited.

General Usefulness Given existing infrastructure for privacy-preserving data aggregation, application providers can opt to make use of it instead of building new infrastructure from scratch. This can be especially useful for less popular applications or applications that only seldom aggregate user data but still with to provide user-privacy, since hooking into existing infrastructure can be achieved with considerable less work than alternative options.

Truly Anonymous Polls When we limit ourselves to Matrix itself we still get some interesting use-cases. Privacy-preserving aggregation can for example be used to create truly anonymous polls within group chats, even in groups where each possible pair of users have compared their shared codes.

5.1.4.2. Clustering

As in the previous scenario, clustering remains a difficult task. Some existing trust-infrastructures already include some concept of "groups", which could be repurposed as clusters. This decision needs to be made on a case by case basis, depending on the exact requirements and underlying trust guarantees.

5.1.4.3. Schemes

Relying on pre-existing infrastructure sets this scenario apart from all the others, which allows us to make heavy use out of the already established trust relationship between clients. Depending on the exact setting and use case of this scenario, no central server exists (e.g. polls in a peer to peer group chat), making trust in a central server superfluous. While data verification is usually a desirable feature, there might also be instances where input data verification is not strictly required and ignored in favour of a simpler data aggregation scheme.

Homomorphic Encryption Homomorphic encryption requires two central servers. If none of the central servers is part of the existing infrastructure we do not have a way to establish the required level of trust. If on the other hand at least one of the central servers is run by a member of the existing trust structure an argument could be made for homomorphic encryption to be somewhat secure in this setting. How usable this approach is in real-life depends on different factors, since running an always connected data collection server on a mobile device might be infeasible. If the mobile devices only run the decryption server and the collection is handled separately, this approach might be worth considering. Whether the lack of input validations is an issue depends on the intended usage.

Additive Secret Sharing Additive secret sharing has similar considerations as homomorphic encryption regarding servers that are either part or not part of the existing infrastructure. It has the by now well known advantage of better privacy guarantees, namely requiring that at least 1 in n servers is honest instead of 1 in 2, but now also requires all collection servers to be always online and available.

Prio Prio supplements additive secret sharing with input validation, but shares the same issues otherwise.

Client Based Prio Client based Prio makes use of the strong existing trust between individual clients and therefore makes it possible to use completely external central servers without negatively impacting client privacy. The increased communication overhead however makes this a better candidate for settings where clients have a continuous power connection.

5.2. Scheme Comparison

We can now summarise key information about all schemes and scenarios presented in this thesis in two tables. Table 5.1 compares the four distinct scenarios with each other, focusing on the properties a given scenario can provide. *Provided Trust in Servers/Clients* refers to the amount of trust that a server or client can reasonably hold in this scenario. As we have seen earlier this amount of trust can vastly differ based on different factors such as cost (of the client) and knowledge that has been previously obtained. *Ease of Clustering* describes how much effort is required to create the client-clusters. While simply creating clusters is not too difficult, we specifically refer to clustering methods where a client can be reasonably sure that the clustering does not negatively impact its privacy.

Table 5.2 compares the privacy-preserving data aggregation schemes with each other, with our main focus on the assumptions required by a given scheme for it to guarantee privacy, which we call *Required Trust in Servers/Clients*. This matches *Provided Trust in Servers/Clients* in the previous table. We also consider the *Communication Overhead* that the scheme produces, meaning the number of messages that need to be exchanged to calculate the aggregate.

Based on the results in each cell, we assign a value of either *Low*, *Medium* or *High* or *Easy*, *Medium* or *Difficult* respectively. This value is merely used to compare results between different schemes/scenarios at a glance and to match which scenario might work well with a given scheme.

By cross-referencing rows between the two tables, we now get a good idea on how well a given aggregation scheme might work for the scenarios outlined in this thesis. We notice that both the car manufacturing and mobile E2E messaging scenario provide “high” trust in clients, but only “low” trust in servers, the only difference being the difficulty of creating appropriate clusters. Due to Client Based Prio requiring appropriate clusters it is a better fit for the car manufacturing scenario than it is for the mobile E2E messaging scenario.

Looking at the original Prio we notice a different pattern. It does not allow clients to have any influence in the final aggregation result, apart from submitting their own (valid) data but requires a slightly higher trust level in the servers compared to Client Based Prio, which makes it a good fit for the browser installation scenario, where trusting clients is not a valid assumption. This is also why Mozilla has started experimenting with variants of Prio to aggregate data [28].

Scenario	Provided Trust in Servers	Provided Trust in Clients	Ease of Clustering
Car Manufacturer (5.1.1)	**Low** • Clients cannot verify that the servers behave honestly	**High** • Clients are expensive • Certificates are uniquely linked to a single client → Makes it difficult to create fake clients • Client population changes slowly	**Easy** • High client trust allows for central cluster assignment
Browser Installations (5.1.2)	**Low** • Clients cannot verify that the servers behave honestly	**Low** • Easy to create new clients • Relatively easy to copy clients → Difficult to verify client integrity • Client population very volatile	**Difficult** • Lack of trust in all other parties • Makes it difficult to trust a clustering • Volatile client population requires constant clustering
Mobile Apps (5.1.3)	**Low** • Clients cannot verify that the servers behave honestly	**Medium** • Servers have an inherent interest in only letting verified[a] clients partake • Clients need to trust the server's inherent interest before they can trust other clients	**Difficult** • Inherent interest of server is not applicable for clustering • Clustering without leaking data about clients is difficult, if servers cannot be trusted
Mobile E2E Messaging (5.1.4)	**Low / n/a**[b] • Clients cannot verify that the servers behave honestly • Some use-cases do not require servers	**High** • Clients have verified the identity of other clients themselves	**Difficult** • Clustering without leaking data about clients is difficult, if servers cannot be trusted

Table 5.1.: Scenario comparison of presented scenarios with reasonably derivable trust assumptions and clustering information.

[a]Verified could for example mean that a client that has connected its bank account
[b]Depending on the exact use case no central servers are required.

Scheme	Required Trust in Servers	Required Trust in Clients	Communication Overhead
Homomorphic Encryption	**High** • *Correctness* if both servers are honest $(2/2)$ • *Privacy* if at least one server is honest $(1/2)$	**Low** • Does not *verify* validity of client input • Clients (and their data) are completely independent from each other	**Low** • Clients need to acquire the server's public key • Each client sends one message to the server • *Collecting* server sends one message to *decrypting* server
Additive Secret Sharing	**Medium** • *Correctness* if all servers are honest (n/n) • *Privacy* if at least one server is honest $(1/n)$	**Low** • Does not *verify* validity of client input • Clients (and their data) are completely independent from each other	**Low** • Each client sends one message to every server • Each server publishes[a] their aggregator value
Prio	**Medium** • *Correctness* if all servers are honest (n/n) • *Privacy* if at least one server is honest $(1/n)$	**None** • Uses Valid predicate to *verify* validity of client input • Clients (and their data) are completely independent from each other	**Medium** • Each client sends one message to every server • For each data submission the servers need to send $2n$ messages to verify the Valid predicate • Each server publishes their aggregator value
Client Based Prio	**Low** • *Correctness* of second central aggregation if all servers are honest (n/n) • *Privacy* even if all central servers are dishonest[b] $(0/n)$	**Medium** • Uses Valid predicate to *verify* validity of client input • How a client behaves can directly impact the aggregation • *Correctness* of pre-aggregation if any cluster has at least*threshold* semi-honest clients • *Privacy* for each cluster that has at least *threshold* honest clients	**High** • Each client sends one message to every client in the same cluster • For each data submission the clients need to send $2m$ messages to verify the Valid predicate • Each client sends one message to every central server • Each server publishes a message

Table 5.2.: Scheme comparison of presented aggregation schemes and their requirements, where n is the number of servers and m the number of clients.

[a] Depending on which parties should have access to the aggregation result the number of messages required might change.
[b] As long as assumption in "Required Trust in Clients" holds.

5.3. General Limitations

Data aggregation schemes producing an exact output are always vulnerable to certain types of attacks. While a privacy focused client might simply abstain from any form of data aggregation, this will also not yield an aggregation result at all. This section gives a short overview over two "weaknesses" that all data-aggregation schemes have in common, and ideas on how to counteract those, such as by using differential privacy, as discussed in the previous chapters.

5.3.1. Selective Denial-of-Service Attack

For this attack to be successful, an adversary needs sufficient control over the network to monitor and drop arbitrary packages. This control enables the adversary to decide which client(s) are able to submit their data, and which client(s) are prevented from submitting data. In the most extreme case all but one honest client can be prevented from communicating with the server(s) [49].

How severe such an attacks really is depends on the considered scenario (section 5.1). If the adversary does not also control a large enough number of clients, such as it might be the case in the *Car Manufacturer* scenario, the privacy of a single client can still be guaranteed. This is due to the inherent required trust that the aggregation does not take place unless enough clients have submitted their data. In cases where control of a large number of clients by a single party can be achieved without much difficulty, as it is the case in the *Browser Installation* scenario for example, an adversary with this level of network control can force an unprotected aggregation scheme to include the data of one honest client with n malicious clients with known input: $f(x_{honest}, x_{malicious_1}, \ldots, x_{malicious_n})$. Depending on the aggregation function f, a malicious party can derive $x_{hontest}$ in parts or even its entirety.

Any mechanism that can be used to defend against Sybil attacks [16] can also be used to make selective denial-of-service attacks more difficult. These defence mechanisms may include identity or personhood validation, as seen in the *Mobile Apps (Public Transportation Scenario)* social networks [2, 57, 65, 64, 63], or a high (artificial) barrier of entry such as purchasing an expensive piece of equipment like a car in the *Car Manufacturer* scenario.

5.3.2. Intersection attack

A different type of attack is commonly referred to as *intersection attack* [10, 14, 30, 61, 12] and relies on the assumption, that the clients' values remain mostly constant over time. If this is the case an adversary can first observe the output of an aggregation that has not been manipulated, $f(x_1, \ldots, x_n)$, where x_i is the input of honest client i. If the adversary is then able to prevent a single client x_j from taking part in a subsequent round of the aggregation, the adversary can learn the output of the aggregation where x_j did not submit its data, $f(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n)$. This allows the adversary to learn the difference between the two outputs, making it possible to infer information about the x_i-th input data. Using a differential privacy mechanism to add noise can defend against this type of attack [17].

6. Conclusion

We have seen multiple different approaches to privacy-preserving data aggregation that all possess unique properties, which makes it nothing but impossible to definitively argue for an objectively best scheme. What an appropriate data aggregation scheme looks like is highly dependant on the context in which the scheme is going to be deployed in, especially so on the desired trust assumptions of each of the participating parties. While we did not manage to find an aggregation scheme that performs consistently better in key metrics, like maintaining client privacy, securing against misbehaving parties or protocol overhead, we introduced a new approach that works better in scenarios, where building a trust relationship with other clients is a feasible undertaking. This allows us to guarantee client privacy, even when all aggregation servers behave maliciously.

Future Work Investigation privacy-preserving aggregation schemes remains an ongoing and complex task, and open questions remain. One of the big factors we have considered throughout this thesis is the requirement of trust in other parties. This directly leads us to question whether we can somehow better specify what kind of trust we actually require for different privacy-preserving aggregation schemes, whether different kinds of trust actually exist, or whether it is possible to maintain our level of privacy guarantees, while further reducing the required trust in other parties.

Bibliography

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. "The Design and Analysis of Computer Algorithms". In: 1974. URL: https://api.semanticscholar.org/CorpusID:29599075.

[2] Lorenzo Alvisi et al. "SoK: The Evolution of Sybil Defense via Social Networks". In: *2013 IEEE Symposium on Security and Privacy*. 2013, pp. 382–396. DOI: 10.1109/SP.2013.33.

[3] Yonatan Aumann and Yehuda Lindell. "Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries". In: *Theory of Cryptography*. Ed. by Salil P. Vadhan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 137–156. ISBN: 978-3-540-70936-7.

[4] Lars Backstrom and Jon Kleinberg. "Romantic partnerships and the dispersion of social ties: a network analysis of relationship status on facebook". In: *Proceedings of the 17th ACM conference on Computer supported cooperative work; social computing*. CSCW'14. ACM, Feb. 2014, pp. 831–841. DOI: 10.1145/2531602.2531642. URL: http://dx.doi.org/10.1145/2531602.2531642.

[5] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Berlin, Heidelberg, Germany, Aug. 1992, pp. 420–432. DOI: 10.1007/3-540-46766-1_34.

[6] Donald Beaver. "Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority". In: *Journal of Cryptology* 4.2 (Jan. 1991), pp. 75–122. DOI: 10.1007/BF00196771.

[7] Ghazaleh Beigi et al. *Protecting User Privacy: An Approach for Untraceable Web Browsing History and Unambiguous User Profiles*. 2018. arXiv: 1811.09340 [cs.CR]. URL: https://arxiv.org/abs/1811.09340.

[8] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: *20th Annual ACM Symposium on Theory of Computing*. Chicago, IL, USA: ACM Press, May 1988, pp. 1–10. DOI: 10.1145/62212.62213.

[9] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. "Near-Linear Unconditionally-Secure Multiparty Computation with a Dishonest Minority". In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Berlin, Heidelberg, Germany, Aug. 2012, pp. 663–680. DOI: 10.1007/978-3-642-32009-5_39.

[10] Oliver Berthold and Heinrich Langos. "Dummy Traffic against Long Term Intersection Attacks". In: *PET 2002: 2nd International Workshop on Privacy Enhancing Technologies*. Ed. by Roger Dingledine and Paul F. Syverson. Vol. 2482. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Berlin, Heidelberg, Germany, Apr. 2002, pp. 110–128. DOI: 10.1007/3-540-36467-6_9.

[11] California Legislature. *California Consumer Privacy Act of 2018*. Cal. Civ. Code § 1798.100 et seq. 2018. URL: https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375.

[12] Henry Corrigan-Gibbs and Dan Boneh. "Prio: Private, Robust, and Scalable Computation of Aggregate Statistics". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 259–282. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs.

[13] Aref N. Dajani et al. "The modernization of statistical disclosure limitation at the U.S. Census Bureau". In: *U.S. Census Bureau* (2020). URL: https://www2.census.gov/cac/sac/meetings/2017-09/statistical-disclosure-limitation.pdf.

[14] George Danezis and Andrei Serjantov. "Statistical Disclosure or Intersection Attacks on Anonymity Systems". In: *Information Hiding*. Ed. by Jessica Fridrich. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 293–308. ISBN: 978-3-540-30114-1.

[15] Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[16] John R. Douceur. "The Sybil Attack". In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260. ISBN: 978-3-540-45748-0.

[17] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014), pp. 211–407. ISSN: 1551-305X. DOI: 10.1561/0400000042. URL: http://dx.doi.org/10.1561/0400000042.

[18] Cynthia Dwork et al. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *TCC 2006: 3rd Theory of Cryptography Conference*. Ed. by Shai Halevi and Tal Rabin. Vol. 3876. Lecture Notes in Computer Science. New York, NY, USA: Springer, Berlin, Heidelberg, Germany, Mar. 2006, pp. 265–284. DOI: 10.1007/11681878_14.

[19] Cynthia Dwork et al. "Our Data, Ourselves: Privacy Via Distributed Noise Generation". In: *Advances in Cryptology – EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. Lecture Notes in Computer Science. St. Petersburg, Russia: Springer, Berlin, Heidelberg, Germany, May 2006, pp. 486–503. DOI: 10.1007/11761679_29.

[20] European Parliament and Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)*. May 2016.

[21] F&I Tools. *Top 15 Automakers in the World | Car Sales Rank Worldwide*. Accessed: 2025-01-17. 2025. URL: https://www.factorywarrantylist.com/car-sales-by-manufacturer.html.

[22] Paul Feldman. "A practical scheme for non-interactive verifiable secret sharing". In: *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. 1987, pp. 427–438. DOI: 10.1109/SFCS.1987.4.

[23] Threema GmbH. "Cryptography Whitepaper". In: (2025). URL: https://threema.ch/press-files/2_documentation/cryptography_whitepaper.pdf.

[24] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[25] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *19th Annual ACM Symposium on Theory of Computing*. Ed. by Alfred Aho. New York City, NY, USA: ACM Press, May 1987, pp. 218–229. DOI: 10.1145/28395.28420.

[26] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208.

[27] Maximilian Held et al. "Lifespans of passenger cars in Europe: empirical modelling of fleet turnover dynamics". In: *European Transport Research Review* 13.1 (Jan. 2021), p. 9. ISSN: 1866-8887. DOI: 10.1186/s12544-020-00464-0. URL: https://doi.org/10.1186/s12544-020-00464-0.

[28] Robert Helmer, Anthony Miyaguchi, and Eric Rescorla. *Testing Privacy-Preserving Telemetry with Prio*. Oct. 2018. URL: https://hacks.mozilla.org/2018/10/testing-privacy-preserving-telemetry-with-prio/.

[29] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. 2nd. Chapman & Hall/CRC, 2014. ISBN: 1466570261.

[30] Dogan Kedogan, Dakshi Agrawal, and Stefan Penz. "Limits of Anonymity in Open Environments". In: *Information Hiding*. Ed. by Fabien A. P. Petitcolas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 53–69. ISBN: 978-3-540-36415-3.

[31] Hannah Keller et al. "Secure Noise Sampling for DP in MPC with Finite Precision". In: (2023). DOI: 10.1145/3664476.3664490. URL: https://eprint.iacr.org/2023/1594.

[32] Donald E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. USA: Addison-Wesley Longman Publishing Co., Inc., 1997. ISBN: 0201896842.

[33] Joseph P. S. Kung, Gian-Carlo Rota, and Catherine H. Yan. *Combinatorics: The Rota Way*. Cambridge Mathematical Library. Cambridge University Press, 2009.

[34] Ben Laurie, Eran Messeri, and Rob Stradling. *Certificate Transparency Version 2.0*. RFC 9162. Dec. 2021. DOI: 10.17487/RFC9162. URL: https://www.rfc-editor.org/info/rfc9162.

[35] Qiongxiu Li and Mads Graesbøll Christensen. "A Privacy-Preserving Asynchronous Averaging Algorithm based on Shamir's Secret Sharing". In: *2019 27th European Signal Processing Conference (EUSIPCO)*. 2019, pp. 1–5. DOI: 10.23919/EUSIPCO.2019.8903166.

[36] Matrix.org. *Matrix: An Open Network for Secure, Decentralized Communication*. 2025. URL: https://matrix.org/.

[37] Ueli M. Maurer. "Information-Theoretic Cryptography (Invited Lecture)". In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Berlin, Heidelberg, Germany, Aug. 1999, pp. 47–64. DOI: 10.1007/3-540-48405-1_4.

[38] R. J. McEliece and D. V. Sarwate. "On sharing secrets and Reed-Solomon codes". In: *Commun. ACM* 24.9 (Sept. 1981), pp. 583–584. ISSN: 0001-0782. DOI: 10.1145/358746.358762. URL: https://doi.org/10.1145/358746.358762.

[39] Frank McKeen et al. “Innovative instructions and software model for isolated execution”. In: *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP ’13. Tel-Aviv, Israel: Association for Computing Machinery, 2013. ISBN: 9781450321181. DOI: 10.1145/2487726.2488368. URL: https://doi.org/10.1145/2487726.2488368.

[40] Jeremias Mechler et al. “Kryptographische Protokolle”. KIT Lecture. July 2024.

[41] Daniele Micciancio. “Fully Composable Homomorphic Encryption”. In: (Oct. 2024).

[42] David Naccache and Jacques Stern. “A New Public Key Cryptosystem Based on Higher Residues”. In: *ACM CCS 98: 5th Conference on Computer and Communications Security*. Ed. by Li Gong and Michael K. Reiter. San Francisco, CA, USA: ACM Press, Nov. 1998, pp. 59–66. DOI: 10.1145/288090.288106.

[43] Goldreich Oded. *Foundations of Cryptography: Volume 2, Basic Applications*. 1st. USA: Cambridge University Press, 2009. ISBN: 052111991X.

[44] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *Advances in Cryptology – EUROCRYPT’99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Prague, Czech Republic: Springer, Berlin, Heidelberg, Germany, May 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[45] Kenan Kingsley Phiri and Hyunsung Kim. “Linear (t, n) Secret Sharing Scheme with Reduced Number of Polynomials”. In: *Security and Communication Networks* 2019.1 (2019), p. 5134534. DOI: https://doi.org/10.1155/2019/5134534. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2019/5134534. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2019/5134534.

[46] Erasmo Purificato, Ludovico Boratto, and Ernesto William De Luca. *User Modeling and User Profiling: A Comprehensive Survey*. 2024. arXiv: 2402.09660 [cs.AI]. URL: https://arxiv.org/abs/2402.09660.

[47] I. S. Reed and G. Solomon. “Polynomial Codes Over Certain Finite Fields”. In: *Journal of the Society for Industrial and Applied Mathematics* 8.2 (1960), pp. 300–304. DOI: 10.1137/0108018. eprint: https://doi.org/10.1137/0108018. URL: https://doi.org/10.1137/0108018.

[48] Jacob T. Schwartz. “Probabilistic algorithms for verification of polynomial identities”. In: *Symbolic and Algebraic Computation*. Ed. by Edward W. Ng. Berlin, Heidelberg: Springer Berlin Heidelberg, 1979, pp. 200–215. ISBN: 978-3-540-35128-3.

[49] Andrei Serjantov, Roger Dingledine, and Paul Syverson. “From a Trickle to a Flood: Active Attacks on Several Mix Types”. In: *Information Hiding*. Ed. by Fabien A. P. Petitcolas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 36–52. ISBN: 978-3-540-36415-3.

[50] Adi Shamir. “How to Share a Secret”. In: *Communications of the Association for Computing Machinery* 22.11 (Nov. 1979), pp. 612–613. DOI: 10.1145/359168.359176.

[51] Claude E. Shannon. “Communication theory of secrecy systems”. In: *Bell Systems Technical Journal* 28.4 (1949), pp. 656–715.

[52] Signal. *Is it private? Can I trust it?* Accessed: 2025-01-29. 2025. URL: https://support.signal.org/hc/en-us/articles/360007320391-Is-it-private-Can-I-trust-it.

[53] Jun Tang et al. *Privacy Loss in Apple’s Implementation of Differential Privacy on MacOS 10.12*. 2017. arXiv: 1709.02753 [cs.CR]. URL: https://arxiv.org/abs/1709.02753.

[54] Telegram. *Telegram End-to-End Encryption API*. Accessed: 2025-01-29. 2025. URL: https://core.telegram.org/api/end-to-end.

[55] Radek Tomsu, Samuel Marchal, and N. Asokan. *Profiling Users by Modeling Web Transactions*. 2017. arXiv: 1703.09745 [cs.CR]. URL: https://arxiv.org/abs/1703.09745.

[56] G. S. Vernam. "Cipher Printing Telegraph Systems For Secret Wire and Radio Telegraphic Communications". In: *Transactions of the American Institute of Electrical Engineers* XLV (1926), pp. 295–301. DOI: 10.1109/T-AIEE.1926.5061224.

[57] Bimal Viswanath et al. "An analysis of social network-based Sybil defenses". In: *SIGCOMM Comput. Commun. Rev.* 40.4 (Aug. 2010), pp. 363–374. ISSN: 0146-4833. DOI: 10.1145/1851275.1851226. URL: https://doi.org/10.1145/1851275.1851226.

[58] WhatsApp. *WhatsApp FAQ*. Accessed: 2025-01-29. 2025. URL: https://faq.whatsapp.com/820124435853543/?helpref=uf_share.

[59] Lee J. White. "Introduction to Combinatorial Mathematics (C. L. Liu)". In: *SIAM Review* 11.4 (1969), pp. 634–635. DOI: 10.1137/1011109. eprint: https://doi.org/10.1137/1011109. URL: https://doi.org/10.1137/1011109.

[60] Royce J Wilson et al. *Differentially Private SQL with Bounded User Contribution*. 2019. arXiv: 1909.01917 [cs.CR]. URL: https://arxiv.org/abs/1909.01917.

[61] David Isaac Wolinsky, Ewa Syta, and Bryan Ford. "Hang with your buddies to resist intersection attacks". In: *ACM CCS 2013: 20th Conference on Computer and Communications Security*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. Berlin, Germany: ACM Press, Nov. 2013, pp. 1153–1166. DOI: 10.1145/2508859.2516740.

[62] Yinghui (Catherine) Yang. "Web user behavioral profiling for user identification". In: *Decision Support Systems* 49.3 (2010), pp. 261–271. ISSN: 0167-9236. DOI: https://doi.org/10.1016/j.dss.2010.03.001. URL: https://www.sciencedirect.com/science/article/pii/S0167923610000497.

[63] Haifeng Yu et al. "SybilGuard: defending against sybil attacks via social networks". In: *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '06. Pisa, Italy: Association for Computing Machinery, 2006, pp. 267–278. ISBN: 1595933085. DOI: 10.1145/1159913.1159945. URL: https://doi.org/10.1145/1159913.1159945.

[64] Haifeng Yu et al. "SybilGuard: defending against sybil attacks via social networks". In: *SIGCOMM Comput. Commun. Rev.* 36.4 (Aug. 2006), pp. 267–278. ISSN: 0146-4833. DOI: 10.1145/1151659.1159945. URL: https://doi.org/10.1145/1151659.1159945.

[65] Haifeng Yu et al. "SybilLimit: A Near-Optimal Social Network Defense against Sybil Attacks". In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. 2008, pp. 3–17. DOI: 10.1109/SP.2008.13.

[66] Richard Zippel. "Probabilistic algorithms for sparse polynomials". In: *Symbolic and Algebraic Computation*. Ed. by Edward W. Ng. Berlin, Heidelberg: Springer Berlin Heidelberg, 1979, pp. 216–226. ISBN: 978-3-540-35128-3.

A. Index of Common Notations

The notations used within this thesis closely resemble the notations introduced in [29] with minor differences.

A.1. General Notation

- $:=$ is used for deterministic assignments
- If S is a set, then $x \xleftarrow{\$} S$ denotes choosing an element x uniformly at random from S
- If A is a randomised algorithm, then $y \leftarrow A(x)$ denotes running A on input x with a uniform random tape and assigning the output to y
- $\Pr[X]$ refers to the probability of event X happening

A.2. Cryptographic Notation

- κ is the security parameter
- *PPT* is an acronym for "probabilistic polynomial time"
- (pk, sk) represents the pair of public and private-key used in asymmetric cryptography
- negl refers to an arbitrary negligible function, see Definition 13

A.3. Number Theory

- $\mathbb{F}_p$ denotes a (usually finite) field. If the order p is obvious given the context or not important we write $\mathbb{F}$ instead.
- $\mathfrak{S}_n$ denotes the symmetric group over n symbols
- $\lfloor x \rfloor$ is the largest integer less than or equal to x:

$$\lfloor x \rfloor = \max \{ m \in \mathbb{Z} \mid m \leq x \}$$

- $\lceil x \rceil$ is the smallest integer greater than or equal to x:

$$\lceil x \rceil = \min \{ n \in \mathbb{Z} \mid n \geq x \}$$

- $\gcd(x, y)$ returns the *greatest common divisor* of non-zero integers x and y. That is the largest integer that divides both x and y.
- $\mathrm{lcm}(x, y)$ returns the *least common multiple* of two integers x and y. That is the smallest positive integer that is devisable by both x and y.

A.4. Combinatorics

- For a set A and an integer t, $\binom{A}{t}$ represents the set of all subsets of A of size t.

A.5. Variable Names

- n, m are usually used when referring to how many "objects" of a given type exists in the current context, such as n servers and m clients are required for this scheme, or n shares are created.
- t is usually used to indicate the threshold value for (t, n)-threshold secret sharing.
- l is usually used to denote dimensions.