Private Data Aggregation on a Budget

Morten Dahl Valerio Pastro Mathieu Poumeyrol

March 15, 2017

Abstract

We provide a practical solution to performing cross-user machine learning through aggregation on a sensitive dataset distributed among privacy-concerned users.

We focus on a scenario in which a single company wishes to obtain the distribution of aggregate features, while ensuring a high level of privacy for the users. We are interested in the case where users own devices that are not necessarily powerful or online at all times, like smartphones or web browsers. This premise makes general solutions, such as general multiparty computation (MPC), less applicable.

We design an efficient special-purpose MPC protocol that outputs aggregate features to the company, while keeping online presence and computational complexity on the users' side at a minimum. This basic protocol is secure against a majority of corrupt users, as long as they do not collude with the company. If they do, we still guarantee security, as long as the fraction of corrupt users is lower than a certain, tweakable, parameter. We propose different enhancements of this solution: one guaranteeing some degree of active security, and one that additionally ensures differential privacy.

Finally, we report on the performance of our implementation on several realistic real-world use-cases across different devices.

Contents

1 Introduction **3**
- 1.1 General Setting . . . 4
- 1.2 Solution Outline . . . 5
- 1.3 Extensions . . . 6
- 1.4 Related Work . . . 7

2 Preliminaries **9**
- 2.1 Secret Sharing . . . 9
- 2.2 Homomorphic Encryption . . . 12
- 2.3 Pseudorandom Generators . . . 14

3 Secure Aggregation Protocol **14**
- 3.1 Secure Aggregation Protocol . . . 14
- 3.2 Towards Active Security . . . 17
- 3.3 Performance Analysis . . . 18

4 Obtaining Differential Privacy **19**
- 4.1 Differential Privacy . . . 20
- 4.2 Private Aggregation Protocol . . . 21
- 4.3 Binomial Realisation . . . 22

5 Applications **23**
- 5.1 Correlation-preserving Representation . . . 24
- 5.2 Analytics . . . 25
- 5.3 Drug-use Survey . . . 26
- 5.4 Location Heatmaps . . . 28
- 5.5 Movie Ratings . . . 29

6 Implementation **30**
- 6.1 Efficient Packed Secret Sharing . . . 30
- 6.2 Paillier . . . 31

1 Introduction

Many people make decisions based on recommendations: from trivial tasks like choosing restaurants, to more important ones such as choosing the right school or doctor. With the advent of the Internet, these recommendations are shifting from being a word of mouth to being delivered to users through their computers or phones, via data aggregation services that compile several users' experiences into easy-to-understand recommendations.

This change yields immediate and more accurate answers to the users, and is a profitable opportunity for the service-providing companies. Moreover, the techniques used to solve this type of problems are also extendable to many other settings such as surveys, analytics, and probabilistic models; in general, these applications fall in the area of machine learning and (automated) statistics.

However, this shift introduces new challenges that were not apparent before. By large, data aggregation is performed on the users' data after it is stored in the clear on the company's servers. This can be a risk for both the users and the company: the users may be concerned about how their data is used by the company beyond the intended service, and hence they may be reluctant to share relevant but sensitive data; the company may worry about possible data breaches, which can compromise the users' trust in the company and damage its reputation and business, or carry legal consequences.

Data aggregation, providing immense potential but at the same time posing great privacy issues, hence looks like a double-edged sword – this is not the case though, and while there exist general techniques to mitigate the downsides, we are here interested in a more specific practical question:

Can we develop a real-world system
for statistical computation over the users' private inputs,
without revealing them?

In this paper, we answer this question by proposing an efficient technical solution for performing data aggregation in a way that minimises privacy-related issues for both the service-providing company and the users. Abstractly, we can think of a set of N users, where each user P_i has a private input x_i, and a company that has a server S and wants to obtain the (anonymized) distribution D of the x_i (so that S can compute statistical functions or run some machine learning on it), and nothing more. Pictorially, we want to compute the functionality in Table 1.

	P_1	...	P_N	S
Input:	x_1	...	x_N	$\perp$
Output:	$\perp$	...	$\perp$	D

Table 1: The functionality we want to efficiently implement. D represents the distribution of the x_i.

Our solution removes the bulk of the complexity of computing D by translating the computation to a simple sum over a different (and sometimes larger) representation of the x_i – and while this works in a general setting, we underline that it can be tweaked for more specific scenarios: in Section 5 we give details about the usage and performance of our solution in different realistic applications such as analytics, surveys, location heatmaps, and movie ratings.

1.1 General Setting

We assume that the set of users is dynamic (as in: changing over time) and not necessarily large, and that user devices have limited computational power and are sporadically available. As for the company, we may imagine a small company or a start-up launching a new digital service relying on data aggregation processes.

This setting models many real-world scenarios, such as website users who only make a few visits, mobile users who download an app and uninstall it after a while, and sensors with limited lifespan. While this is our primary focus, our solution will deliver even for less challenging scenarios, such as bigger companies that have the resources to partner with external parties[1] that are somehow trusted by the users – this makes matters easier because it allows reliably running a secure multi-party computation (MPC) protocol between the company and its partners to deliver (only) the result to the company; however, we seek *not* to rely on such partners, since a small company or a start-up might not have the means to establish these partnerships. Specifically, we are in a scenario where we would like to use MPC, but there is only a single powerful party, the company, who is reliably online, and the rest are computationally weak and seldom online.

Corruptions Our aim is to protect both against possible hackers who retrieve the company's data after the computation, and against company employees who have access to all the data stored at the company. However, we assume that the code run by honest players follows the protocol, even if this code is provided by the company. One justification is that the company does not want to modify the source code, for example because of the high risk of honest employees (or reverse engineers) raising a flag after seeing malicious code. Also, we want both the honest users' inputs and the output to be kept private from any outsiders, i.e. any set of corrupt parties not colluding with the company.

These are the vital privacy and security guarantees that we require; jumping ahead, our protocol satisfies even stronger ones: namely, we maintain security also if the server is corrupted together with any set of other players, as long as among these players there is only a small number of special users, called *clerks*, that in some sense simulate a partner (as described above) for the company. Concretely, our protocol can be made to

[1] For example governments, research labs, or independent institutions such as the Electronic Frontier Foundation.

guarantee *active* security against corrupt clerks and *passive* security against any other party.

1.2 Solution Outline

We here give a basic version of our aggregation protocol to introduce the high level concepts, abstracting away for now practical issues such as how to implement point-to-point channels and exclude performance optimizations crucial for a deployable system. We return to these in Section 3 where a formal description of the aggregation protocol is given.

Here, we assume that a *committee* C of n clerks (e.g. users) has already been formed, with the property that if the adversary corrupts the server S, then it corrupts at most t of the clerks, where t and n are parameters of the protocol. Each user is provided with point-to-point channels to the server and to each clerk. We also require that the clerks are all online in the final round of the protocol. These are strong assumptions that will be relaxed below.

Our protocol is divided into two phases: an *input* phase and an *output* phase. In the input phase, each user processes its input and sends material to the server and the clerks. In the output phase, each clerk and the server locally manipulate the material received, and after the clerks have sent their results to the server, the latter is able to obtain the result (and nothing else). As a feature, the input-providing users are only required to participate in the input phase.

In more detail, in the input phase, each user P_i processes their input into a convenient (but sometimes larger) representation x_i which makes it easy to calculate the desired function: specifically, the distribution D of the users' original inputs is the sum[2] of the new representations, $D = \sum_{i=1}^{N} x_i$. This allows us to translate the original problem of obtaining distributions to computing sums. At this point, P_i samples a uniform one-time pad key p_i and sends it to the server; at the same time, P_i shares a one-time encryption $c_i = x_i + p_i$ to the committee via Shamir secret sharing for n players with privacy threshold t. From now on, P_i can be offline.

Once enough users have completed the above input procedure, the output phase can take place: the server sums up all the encryptions it received to obtain an aggregated encryption $c = \sum_{i=1}^{N} c_i = \sum_{i=1}^{N} x_i + p_i$, and each clerk C_j sums up the received shares, obtaining a share σ_j of the sum of all the keys (by the fact that Shamir is a linear secret sharing scheme). Each clerk C_j then sends σ_j to the server, who can then reconstruct the aggregated key $p = \sum_{i=1}^{N} p_i$ by Shamir's reconstruction procedure over the σ_j, and, subtracting p from c, obtain the desired result $x = c - p = \sum_{i=1}^{N} x_i$.

Intuitively, since the only data sent to the clerks are one-time pad encryptions, the clerks can never learn any inputs on their own. Likewise, since the server only gets the users' keys, it can never learn any of the inputs on its own. Finally, any coalition that includes the server and at most t clerks will never will be unable to reconstruct any individual input.

[2]Note that our protocol extends straightforwardly to weighted sums.

1.3 Extensions

Our actual aggregation protocol improves on the basic protocol given above in several aspects outlined below.

Input Size and Mitigation We mentioned that the users need to format their inputs in a convenient representation so that the actual computation can be realised as a sum. Depending on the application, this new representation can expand the inputs to a certain extent (exponentially in the worst case), but we employ different techniques to mitigate this: first, by using *packed* secret sharing, and secondly, by using techniques for reducing the expansion factor of public key encryption schemes by encrypting multiple values at once. See Section 2 and 5.1 for more details.

Shorter One-Time Pad Keys By using a strong pseudorandom generator, the one-time pad keys p_i may be replaced by a short seed s_i to reduce the communication cost. In other words, P_i will sample $p_i \leftarrow \mathsf{PRG}(s_i)$ and only send s_i to the server, who can then perform the same computation to likewise obtain p_i.

Communication Model While the basic protocol relies on point-to-point channels, we may use a *bulletin board* together with a public key encryption scheme to store each party's encrypted messages, making them available to the recipients whenever the latter are online.

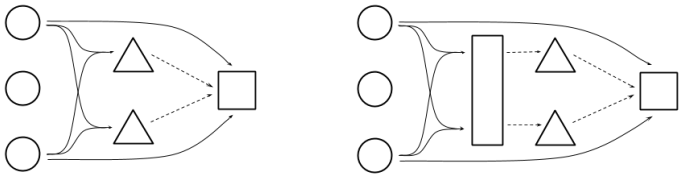

Figure 1: Different communication models between users (circles), clerks (triangles), the server (square), and optionally the bulletin board (rectangle). Left: *basic flow*. Right: *flow with bulletin board*.

One motivation to use this model in practice is that it well captures our setting, in which users are sporadically online. But another somewhat surprising feature is that in many settings it also allows us to reduce the amount of computation performed by the clerks and avoids the need for a direct link between users and clerks.

Outsourcing Clerk Computation In the bulletin board model, if we use an additively homomorphic public key encryption scheme, we can reduce both the download and computation complexity for each clerk from linear in the number of users to constant. In the above protocol, each clerk receives a number of encryptions proportional to the number of users, for the clerk to add up, decrypt, and send to the server. With an additively homomorphic scheme, the bulletin board can perform the addition instead,

and send each encrypted sum to the corresponding clerk, so that the latter only has to decrypt and send the result back. This comes at the cost of having the users perform potentially more expensive encryption, and of having the bulletin board compute in the ciphertext space what the clerks would have computed in the plaintext space, which may be much more expensive; however, we argue that in some scenarios the gain for clerks may justifies the computational overhead of the bulletin board: clerks are weak devices and are not reliably online, while the bulletin board can be sponsored by the company willing to invest more resources.

Active Security against all Clerks The above idea of having the clerk computations outsourced to the bulletin board can be further extended so that the protocol achieves active security against corrupt clerks: since the only work each clerk has to perform is then just a decryption, we could have the clerk send a proof of correct decryption alongside the resulted value, and have the bulletin board check it before proceeding. One simple way to do this, for example if the encryption scheme used is Paillier, is to have each clerk submit the encryption randomness together with the plaintext, so that the bulletin board can check correctness by simply re-encrypting the plaintext under the randomness provided and check whether it equals the ciphertext originally sent for decryption.

Active Security against some Clerks As an alternative to the previous approach, the protocol can also be enriched to achieve active security against a certain amount of corrupt clerks regardless of the communication model used and with only a small change to the server. This can be achieved by replacing Shamir's reconstruction algorithm with its robust counter part, making it $((n-t)/2, 0)$-robust. This means that if at most $(n-t)/2$ clerks are actively corrupt, our original protocol will still be secure.

Since this method only changes the reconstruction procedure on the server, and adds no overhead to communication or computation for anyone else, it can be more attractive than the previous approach (or to use general zero-knowledge proofs) when the computational overhead of an encryption scheme such as Paillier is prohibitive.

Differential Privacy While the basic protocol provides privacy of each user's input given the output, there is still a chance that the exact aggregation leaks "too much" about the users' inputs. This concern is usually addressed through differentially privacy, and our protocol accommodates mechanisms that can be run efficient between the clerks as part of the aggregation. See Section 4 for details.

1.4 Related Work

Despite a long line of work, including generic systems and specialised protocols sacrificing expressiveness for efficiency and flexibility, to the best of our knowledge none of these are optimal in the setting we consider, assuming either reliable or invested clerks. On

the other hand, some of these systems offer additional security measures, such as range proofs for inputs and active security against noise sampling for differential privacy.

Multi-party computation (MPC) General MPC solutions [Yao82, GMW87, BGW88, CCD88], even the most recent and practical ones [DPSZ12, DKL+13, ZRE15, KOS16], are not particularly well-suited for our setting because they usually require several players to have a (powerful) computer whose public identity is known by everyone and which is online essentially at all times during the computation. These systems, however, may be used to compute arbitrary functions and in some cases offer stronger security guarantees with respect to corruption.

Server-aided MPC Our setting carries similarities to server-aided MPC [FKN94, DI05, DIK+08, BCD+09, KMR11], where the players are split into input providers and a few servers who receive the inputs and carry out the computation between them. In our setting, though, there is only one server, and the input providers are weak and seldom online. We argue that this scenario is closer to the real world for many companies, where users can join and leave the computation at any point, and are only willing to spend a small amount of resources, while the company, with substantial benefits from offering the service, has a strong incentive to invest in computational power and online presence.

(Somewhat) Homomorphic Encryption Homomorphic encryption schemes [BV11, BGV12, FV12] are good candidates for our task: users encrypt their inputs under the same public encryption key, send these encryptions to the company who, not knowing the decryption key, homomorphically aggregates all values into a single encryption.[3] This is then sent to a *decryptor* that has the private decryption key and can deliver the output. This puts large requirements on the decryptor, both in protecting the key and ensuring availability. Using a threshold variant [DJ01] to split the decryption key between a group of decryptors is one remedy, but can have the downside of a complex key generation protocol, with expensive interaction or orchestration between the decryptors [HMRT12, LTV12, HLP11], even when limited to our simpler aggregation function [RN10]. Our protocol follows a similar approach, but avoids expensive key setup and minimises interaction and requirements for online presence. Others use encryption schemes without efficient decryption [EDG14], making them unsuitable for aggregation over large sets, or use schemes with huge expansion factors [CRFG12].

Secret sharing The work of [DKM+06] also uses secret sharing and provides (actively secure) protocols for sampling discrete noise from both the Binomial and Poisson distributions. However, their work does not address how most of the computation can be offloaded to a single server and in general puts a heavy load on (a subset of) the users which is unsuitable for our setting. The work of [DCZ10] is similar in this respect. Other work [ÁC11, SCR+11] require an expensive synchronisation phase between users

[3] Notice that the degree of homomorphicity can be lowered to additive, such as Paillier [Pai99] or LWE-based [Reg05, LP11], if the aggregation is a linear function.

in which they set up one-time pads that cancel out in the final aggregation; this further adds strong availability requirements for decryption, or results in a much larger communication overhead compared to our protocol [CSS12]. Others [HN12, CAF13] assume reliable clerks through which all communication passes.

Local Differential Privacy An alternative to the cryptographic solutions is for the users to add a small amount of "noise" to their inputs before sending them unencrypted to the server [EPK14]. This approach guarantees the same level of privacy without requiring third parties to hold the decryption key such as the clerks. Moreover, it is extremely efficient both computationally and in terms of data transmission, and allows the company to mix and re-use the noisy inputs for different computations without having to re-run any input protocols. However, because the noise accumulates in the output, large amounts of data is needed before the signal overpowers the noise, making this more appropriate for "Internet scale" data sets and less attractive for e.g. smaller companies and start-ups. In this light our extended protocol may be seen as falling into the line of work that seeks to reduce the overall noise added by shifting from a "local" to a "global" model, yet providing the same level of privacy.

2 Preliminaries

Throughout the paper we write x_i to mean the ith coordinate of a vector $x \in X_1 \times \cdots \times X_n$. For any set $I \subseteq \{1, \ldots, n\}$ we write x_I to mean the vector whose ith coordinate is $(x_I)_i = x_i$ for $i \in I$ and $(x_I)_i = \perp$ for $i \notin I$, for a special symbol $\perp$ not in any X_j. We write X^n for $X_1 \times \cdots \times X_n$ when $X_i = X$ for all $1 \leq i \leq n$.

2.1 Secret Sharing

For simplicity, we assume that messages, or secrets, (usually denoted by m) belong to a vector space $\mathbb{F}^k$ for some k, and shares (usually denoted by σ_j) to the base field $\mathbb{F}$. Abusing notation, we also occasionally use σ_j to denote an element from set $\mathbb{F} \cup \{\perp\}$ where $\perp$ is a special rejection symbol not in $\mathbb{F}$.

Definition 2.1 (Secret Sharing)**.** An *n-player, r-reconstruction, t-private secret sharing scheme* over a message space $\mathbb{F}^k$ and share space $\mathbb{F}$ is a tuple $(\mathsf{Share}, \mathsf{Recon})$ of PPT algorithms with the following syntax:

- $\mathsf{Share}(m) \to \sigma$: On input message $m \in \mathbb{F}^k$, output share sequence $\sigma \in \mathbb{F}^n$. When needed we will write $\mathsf{Share}(m; R)$ to explicitly say that the random coins used in Share are described by randomness R.
- $\mathsf{Recon}(\sigma) \to m$: On input σ with $\sigma_j \in \mathbb{F} \cup \{\perp\}$, output a message $m \in \mathbb{F}^k$.

And the following properties:

- *t-privacy*: Any t out of n shares of a secret give no information on the secret itself: for any $m, m' \in \mathbb{F}^k$, any $I \subseteq \{1, \ldots, n\}$ of size $|I| = t$, the distributions $\mathsf{Share}(m)_I$ and $\mathsf{Share}(m')_I$ are identical.

- *(perfect) r-reconstruction*: any set of at least r "well-formed" shares unambiguously defines a secret. In other words, for any $I \subseteq \{1, \ldots, n\}$ with $|I| \geq r$, and for any $\sigma \in \mathbb{K}^n$, if there exists an $m \in \mathbb{F}^k$ such that $\mathsf{Share}(m)_I = \sigma_I$ then we have:

$$\Pr[\mathsf{Recon}(\sigma_I) = m] = 1.$$

Definition 2.2 (Linear Scheme)**.** A secret sharing scheme $(\mathsf{Share}, \mathsf{Recon})$ is *linear* if each share is computed as a fixed linear combination of the secret and some uniform randomness [CDM00]. In other words, $\mathsf{Share}(m; R)$ can be described as an n-by-r matrix M operating against $(m, R)^T$ with R interpreted as a uniform element from $\mathbb{F}^{r-k}$. Notice that this implies that Recon can be expressed as a linear function as well. Also notice that any linear combination of vectors of shares is indistinguishable from a vector of "well-formed" shares (due to the uniformity of the randomness used in Share).

Observation 2.3 (Share Simulatability)*.* A t-private secret sharing scheme $(\mathsf{Share}, \mathsf{Recon})$ has *simulatable shares* if, given any set of t well-formed shares and any secret m, it is possible to efficiently complete that set to a full well-formed sharing of m. Formally, there exists a PPT algorithm $\mathsf{SimShare}$ with the following properties, where m is any secret and $\sigma = \mathsf{Share}(m)$: for any secret m' and any set $A \subseteq \{1, \ldots, n\}$ with $|A| \leq t$, there exists a randomness R' such that for

$$\sigma' = \mathsf{SimShare}(m', \sigma_A)$$

we have $\sigma' = \mathsf{Share}(m'; R')$ and $\sigma_A = \sigma'_A$.

Implementing $\mathsf{SimShare}$ for linear schemes is straightforward: given m' and σ_A, sample a uniform solution to the system of linear equations in the R'_i given by

$$\begin{pmatrix} \sigma_{j_1} \\ \vdots \\ \sigma_{j_{|A|}} \end{pmatrix} = (M)_A \begin{pmatrix} m'_1 \\ \vdots \\ m'_k \\ R'_1 \\ \vdots \\ R'_{r-k} \end{pmatrix}$$

and output $\mathsf{Share}(m'; R')$. Notice that fixing any m' yields at least one solution for the system: for the sake of contradiction, assume that m' gives no solution to the system; this means that the distribution of the secret given $|A| \leq t$ shares is not uniform, which would violate t-privacy (specifically, the distributions $\mathsf{Share}(m)_A$ and $\mathsf{Share}(m')_A$ are distinguishable).

Definition 2.4 (Robust Secret Sharing)**.** A secret sharing scheme $(\mathsf{Share}, \mathsf{Recon})$ is (ρ, δ)-*robust* if the following holds: an adversary that adaptively modifies up to ρ shares makes Recon fail with probability at most δ. More formally, we define the experiment $\mathsf{Exp}(s, \mathsf{Adv})$ for any $m \in \mathbb{F}^k$ and interactive adversary Adv.

E.1. Sample $(\sigma_1, \ldots, \sigma_n) \leftarrow \mathsf{Share}(m)$.

E.2. Set $I := \varnothing$. Repeat the following while $|I| \leq \rho$.

- Adv chooses $i \in \{1, \ldots, n\} \setminus I$.
- Update $I := I \cup \{i\}$ and give s_i to Adv.

E.3. Adv outputs modified shares σ'_i for $i \in I$; let $\sigma'_i := \sigma_i$ for $i \notin I$.

E.4. Compute $m' = \mathsf{Recon}(\sigma'_1, \ldots, \sigma'_n)$.

E.5. If $m' \neq m$ output 1, else 0.

For any (unbounded) adversary Adv and any $m \in \mathbb{F}^k$ it should hold:

$$\Pr[\mathsf{Exp}(m, \mathsf{Adv}) = 1] \leq \delta.$$

We now give a series of example of schemes that can be used in our construction.

Construction 2.5 (Additive Secret Sharing)**.** Additive secret sharing among n players with $k = 1$ and privacy threshold $t = n - 1$ is defined as follows. To share a secret $m \in \mathbb{F}$ among players, set $\mathsf{Additive.Share}(m) = \left(r_1, \ldots, r_{n-1}, m - \sum_{i=1}^{n-1} r_i\right)$ for uniform r_i. To reconstruct, simply compute the sum of all shares: $\mathsf{Additive.Recon}(\sigma_1, \ldots, \sigma_n) = \sum_{i=1}^{n} \sigma_i$.

Additive secret sharing is an example of a linear scheme that enjoys n-reconstruction and no robustness. However, this scheme is still attractive in reliable environments due to the simplicity and speed of its operations.

Shamir secret sharing scheme [Sha79] is an example of a linear secret sharing scheme which is robust for certain parameter choices.

Construction 2.6 (Plain Shamir)**.** Here, we give a brief explanation of the plain version of Shamir secret sharing, among n players, with $k = 1$, t-privacy, and $(t + 1)$-reconstruction. We assume $n < |\mathbb{F}|$.

- $\mathsf{Shamir.Share}_t(m)$:
  1. for $i = 1, \ldots, t$, sample uniform coefficients $f_i \in \mathbb{F}$; let $f_0 := m$, and define the polynomial $f \in \mathbb{F}[X]$ as $f(X) := \sum_{i=0}^{t} f_i X^i$
  2. for $i = 1, \ldots n$, set $\sigma_i \leftarrow f(i)$.
  3. output $(\sigma_1, \ldots, \sigma_n)$
- $\mathsf{Shamir.Recon}_t(\sigma_1, \ldots, \sigma_n)$:

1. interpolate $(\sigma_1, \ldots, \sigma_n)$ on a degree-t polynomial $f \in \mathbb{F}[X]$
2. output $m \leftarrow f(0)$

Plain Shamir enjoys linearity and no robustness.

Construction 2.7 (Robust Shamir)**.** Shamir secret sharing can be thought of as a Reed-Solomon encoding of the polynomial f, which is a message of $r = t+1$ entries, into a codeword of n entries. Reed-Solomon can decode up to $\delta/2$ errors, where $\delta = n-r+1 = n-t$ is the distance of the code. This means that by replacing the reconstruction algorithm of Shamir with an efficient Reed-Solomon decoding procedure, such as Berlekamp-Welch, Shamir becomes $((n-t)/2, 0)$-robust [MS81].

Construction 2.8 (Packed/Ramp Shamir)**.** In plain Shamir, shares and messages are both elements in the same field; however, this can be overridden by increasing the reconstruction threshold and changing the sharing procedure as follows, where we assume $|\mathbb{F}| > n + k$. Let $m = (m_1, \ldots, m_k) \in \mathbb{F}^k$ be a message to share.

- $\mathsf{PackedShamir.Share}_t(m)$:
  1. sample[4] a uniform polynomial $f \in \mathbb{F}[X]$ of degree $t+k-1$ such that $f(-i) = m_i$ for $i = 1, \ldots, k$
  2. for $i = 1, \ldots n$, set $\sigma_i \leftarrow f(i)$.
  3. output $(\sigma_1, \ldots, \sigma_n)$

This variant of Shamir enjoys t-privacy, $(t + k)$-reconstruction, linearity, and $((n - t - k)/2, 0)$-robustness if equipped with the Reed-Solomon reconstruction procedure described in Construction 2.7.

In this construction, k measures the number of secrets that can be packed into one sharing; while increasing the reconstruction threshold compared to plain Shamir ($t + k$ instead of $t + 1$), this scheme can have very short shares ($1/k$ times the message length). This will help us reduce the communication and computational complexity of our protocol.

2.2 Homomorphic Encryption

For comparison purposes later, we need to expand the traditional notion of additively homomorphic encryption to a finer-grained one that takes into account an explicit bound on the number of operations that can be performed on encryptions.

Definition 2.9. Given a plaintext space $\mathcal{P}$ that can be represented as a subset of $\mathbb{Z}^\beta$ for some integer β, and ciphertext space $\mathcal{E}$, an *α-additive homomorphic encryption scheme* is a tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$ of PPT algorithms with the following syntax:

- $\mathsf{KeyGen}(1^\kappa) \to (ek, dk)$: Given a security parameter, generate a correlated encryption and decryption key.

[4] See Section 6.1 for explicit sampling algorithms.

- $\mathsf{Enc}_{ek}(m) \to e$: Encrypt plaintext m into encryption e.
- $\mathsf{Dec}_{dk}(e) \to m$: Decrypt encryption e.
- $\mathsf{Add}_{ek}(e_1, \ldots, e_\alpha) \to e$: Given encryptions e_i, output encryption e containing the sum of the encrypted inputs.

Besides the standard correctness and security (IND-CPA) requirements, we also assume that the scheme is α-*additive*, meaning:

$$\mathsf{Dec}_{dk}\big(\mathsf{Add}_{ek}\big(\mathsf{Enc}_{ek}(m_1), \ldots, \mathsf{Enc}_{ek}(m_\alpha)\big)\big) = \sum m_i$$

where the sum operation is over *the integers*.

Construction 2.10 (Stream cipher)**.** Following our definition, stream ciphers combined with a public key primitive, as done in for instance NaCl [BLS12], are 1-homomorphic, with Add being simply the identity function. Their small expansion factor and simple operations might make them relevant when instantiating our protocol, for instance when the computational resources of user devices are very limited.

Construction 2.11 (Packed Paillier)**.** Given two large equal-length primes p, q, the plain Paillier encryption scheme [Pai99] defines a mapping from a plaintext $m \in \mathbb{Z}_n$ to encryption $e \in \mathbb{Z}^\star_{n^2}$, where for security p, q must both be large, say minimum 1024 bits. To satisfy the requirement of α-additivity for a specific value of α we can define a new plaintext space $\mathbb{Z}_B$, keeping B sufficiently smaller than n to "leave room" for simulating integer addition. To furthermore take advantage of the fact that n is large, we may pack several values $m_i \in \mathbb{Z}_B$ into each encryption by leaving empty space between them. Specifically, by using the plain Paillier scheme to encrypt $m = \sum_{i=0}^{\beta-1} m_i \cdot 2^{s \cdot i}$ with $s = \lceil \log_2(\alpha) \rceil + \lceil \log_2(B) \rceil$ we get a packed variant that is α-additive and allows us to pack β values together, as long as $\lfloor \log_2(n) \rfloor \geq \beta \cdot s$.

Construction 2.12 (BV/BGV)**.** The BV [BV11] and BGV [BGV12] somewhat homomorphic encryption schemes canonically allow packing (batching) multiple values in each ciphertext. This happens because the plaintext space of these schemes is a polynomial ring $R_p = \mathbb{F}_p[X]/\Phi(X)$, where $\Phi(X)$ is a cyclotomic polynomial in $\mathbb{Z}[X]$, and p is a prime. In this setup, we can use a CRT representation of R_p, since we can write Φ as $\prod_{i=1}^{\beta} F_i$, where the F_i are irreducible polynomials of degree $s = \deg(\Phi)/\beta$. More in detail, we can decompose R_p as:

$$\begin{aligned} R_p &= \mathbb{F}_p[X]/\Phi = \mathbb{F}_p[X]/(F_1, \ldots, F_\beta) \\ &\cong \mathbb{F}_p[X]/(F_1) \times \cdots \times \mathbb{F}_p[X]/(F_\beta) \\ &\cong \mathbb{F}_{p^s} \times \cdots \times \mathbb{F}_{p^s} = (\mathbb{F}_{p^s})^\beta \end{aligned}$$

which means that we can encode β elements of $\mathbb{F}_p^s$ in R_p before encrypting, and the fact that this encoding is a ring homomorphism implies that operations on $\mathbb{F}_p^s$ are mapped to the corresponding ring operations in R_p, and therefore allows us to batch β elements into a plaintext. For more details we refer to [BV11, BGV12].

2.3 Pseudorandom Generators

A pseudorandom generator PRG is a deterministic polynomial-time algorithm mapping short *seed* strings s of length κ to longer strings of length $\ell(\kappa)$. Furthermore, the output of PRG is indistinguishable from a uniformly random string of length $\ell(\kappa)$ for any efficient adversary; more formally, for any PPT algorithm D,

$$\left| \Pr[D(\mathsf{PRG}(s)) = 1] - \Pr[D(r) = 1] \right|$$

is negligible in κ, where s is drawn uniformly from $\{0,1\}^\kappa$ and r uniformly from $\{0,1\}^{\ell(\kappa)}$.

We note that a PRG over bit strings may be used to efficiently sample uniform elements from any finite set using rejection sampling. In the rest of the paper we shall hence often assume that the output of PRG is a uniform field element.

3 Secure Aggregation Protocol

We here present our protocol for secure aggregation in detail. By introducing a bulletin board we not only avoid the need of direct communication lines between users and clerks, we also improve both communication and computation since a significant amount of work can be outsourced to the company. And by assuming a passively secure bulletin board we may achieve active security against the server and the clerks.

3.1 Secure Aggregation Protocol

Recall that the main idea for aggregation is simple. First, each player sends a one-time pad key to the server, and secret shares an encryption of their input via that key to the clerks in the committee. Then, once enough players has done so, the server computes the sum of all the keys and receives shares of the sum of all one-time pad encryptions from the clerks. At this point the server can reconstruct the sum of all one-time pad encryptions (without any information on each individual encryption revealed), and unmask the sum of the inputs.

A formal description of our protocol $\Pi_{\mathsf{aggregation}}$ is given in Figure 2. To elect the committee we use functionality $\mathcal{F}_{\mathsf{election}}$ described in Figure 3 and parametrised by a distribution $\mathcal{D}$ over clerk candidates; we do not focus on the implementation of $\mathcal{F}_{\mathsf{election}}$ since it may vary based on the application and the desired security guarantees.

Note that in the description of $\Pi_{\mathsf{aggregation}}$ we say that a player posts a message when sending to the bulletin board, and in step 8 that the bulletin board compresses a set of ciphertexts. This latter operation means the following: the bulletin board selects a partition $\mathcal{P} = (A_1, \ldots, A_{\lceil N/\alpha \rceil})$ of $1, \ldots, N$ where $|A_{i'}| \leq \alpha$, and uses the α-additivity of the encryption scheme to compute $e_{I,j} = \mathsf{Add}_{i \in I \in \mathcal{P}}(e_{i,j})$. The result $\mathcal{E}_j = \{e_{I,j}\}_{I \in \mathcal{P}}$ will have size $\lceil N/\alpha \rceil$.

We see that the amount of interaction is minimised: since users may not be online reliably we chose to have them only send data in one step of the protocol, and only to the server and the bulletin board, which are supposed to be online at any time. Likewise,

$\Pi_{\mathsf{aggregation}}$

Let $P = \{P_i\}_{i=1}^N$ denote the set of players in the protocol, $C = \{C_j\}_{j=1}^n$ the committee of clerks, B the bulletin board, and S the server. We assume the parties agreed upon an encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$ which is α-additive for some α. Moreover, $(\mathsf{Share}_t, \mathsf{Recon})$ is a t-private linear secret sharing scheme for n players.

Setup Phase:

1. S and P call $\mathcal{F}_{\mathsf{election}}$ to elect the committee C of clerks
2. C_j runs $(ek_j, dk_j) \leftarrow \mathsf{KeyGen}$ and posts ek_j

Input Phase:

3. P_i picks random s_i and computes $p_i \leftarrow \mathsf{PRG}(s_i)$,
4. P_i computed $r_i \leftarrow x_i + p_i$ and $(\sigma_{i,1}, \ldots, \sigma_{i,n}) \leftarrow \mathsf{Share}_t(r_i)$
5. P_i computes $e_{i,j} = \mathsf{Enc}_{ek_j}(\sigma_{i,j})$ and posts it on B for $j = 1, \ldots, n$
6. P_i sends s_i to S

Output Phase:

7. S asks B and all C_j to report
8. B compresses $\{e_{i,j}\}_{i=1}^N$ into $\mathcal{E}_j = \{e'_{I,j}\}_{I \in \mathcal{P}}$ and posts $\mathcal{E}_j$ for $j = 1, \ldots, n$
9. C_j computes $\sigma_{I,j} \leftarrow \mathsf{Dec}_{dk_j}(e'_{I,j})$ for all $e'_{I,j} \in \mathcal{E}_j$
10. C_j computes $\sigma_j \leftarrow \sum \sigma_{I,j}$ and sends it to S
11. S computes $r \leftarrow \mathsf{Recon}(\sigma_1, \ldots, \sigma_n)$.
12. S computes $p_i \leftarrow \mathsf{PRG}(s_i)$ and $p \leftarrow \sum_{i=1}^N p_i$
13. S outputs $x \leftarrow r - p$

Figure 2: Secure aggregation protocol.

$\mathcal{F}_{\mathsf{election}}(\mathcal{D})$

Setup: The functionality is parametrized by a distribution $\mathcal{D}$ over the subsets of $\{1, \ldots, n\}$.

Functionality: Sample $\{C_j\} \leftarrow \mathcal{D}$ and broadcasts $\{C_j\}$.

Figure 3: Committee election functionality.

there is no communication between the clerks, and both key setup and output generation are one-round protocols.

In terms of security we get that $\Pi_{\mathsf{aggregation}}$ is passively secure against any coalition corrupting the bulletin board, the server, any number of users, and up to t clerks. Moreover, as a result of sharing a one-time pad encryption to the committee instead of, say, their actual inputs, the company only needs to focus on protecting their own server,

meaning that $\Pi_{\mathsf{aggregation}}$ is passively secure against any coalition of parties as long as the server is honest.

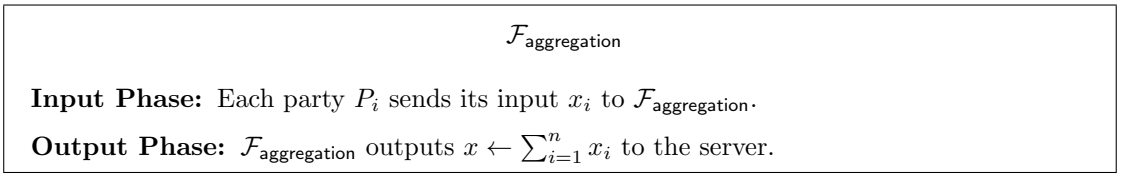

Figure 4: Ideal functionality for statistical computations.

Theorem 3.1. *Protocol* $\Pi_{\mathsf{aggregation}}$ *securely realises ideal functionality* $\mathcal{F}_{\mathsf{aggregation}}$ *given in Figure 4 in the* $\mathcal{F}_{\mathsf{election}}$*-hybrid model, under an adversary* Adv *that corrupts a set* A *of participants as long as* either *of the following conditions is satisfied:*

1. *The server is* not *corrupt* ($S \notin A$)*; or*
2. *The server is corrupt as well as at most* t *clerks* ($S \in A \wedge |A \cap C| \leq t$)

Proof. For simplicity we here give the proof for the hybrid in which the bulletin board has been replaced with secure channels and the PRG with uniformly random pads.

Our proof uses simulator $\mathcal{S}_{\mathsf{aggregation}}$ in Figure 5, which runs a copy of the protocol internally using a fixed constant value (zero) as input for honest users and the actual input for corrupt users. If the server is honest, the simulation of the output phase is trivial: just following the real protocol suffices. On the other hand, if the server is corrupt, after the simulator learns the aggregated value x in the output phase, it furthermore patches the shares sent by the honest clerks so that the server receives a sharing that will make it recover the correct x.

It is trivial to see that if the server is honest then the simulation is indistinguishable from a real run of the protocol, since all values observed by corrupt users and clerks have exactly the same distribution as in the real protocol. More specifically, we may first introduce a hybrid in which the users instead share the one-time pads with the clerks and the one-time padded inputs with the server; indistinguishability follows from the fact that these are identically distributed given only one part. With an honest server it then follows that the adversary now sees identical distributions in the two worlds.

If the server is corrupt we consider two cases. In the first case all users are corrupt, meaning the simulation was in fact done using all the real inputs x_i, in turn making it trivially indistinguishable from the real execution.

In the second case, where $S \in A$ and $U \setminus A \neq \varnothing$, we take advantage of the fact that the number of corrupt clerks is at most t, the privacy threshold of the underlying secret sharing scheme. Intuitively, this means that the adversary is unable to recover the one-time padded inputs of the honest users, and hence is unable to detect that fake values were used during the input phase. Specifically, using share simulatability of the underlying secret sharing scheme the simulator may replace the shares sent by the clerks to the server with fresh shares of $x + p$, constructed such that it agrees with the shares

$\mathcal{S}_{\mathsf{aggregation}}$

Input phase:

- start $\mathcal{F}_{\mathsf{aggregation}}$ and engage its input phase, corrupting the server, the users, and the clerks according to what the adversary does in the real execution of the protocol; this way, the simulator learns the input x_i of each corrupt user P_i
- simulate the input phase of the real protocol pretending $x_i = 0$ for each honest user P_i, and using the leaked x_i for each corrupt user

Output phase:

- if the server is honest ($S \notin A$) or all users are corrupt ($U \setminus A = \varnothing$) then simulate as in the real protocol; otherwise continue as below
- ask the ideal functionality to compute the aggregation, learning x in the case where the server is corrupt and setting $x = 0$ otherwise
- compute all σ_j as in the protocol and get $y \leftarrow \mathsf{Recon}(\sigma_1, \ldots, \sigma_n)$
- let $T = C \cap A$; let x_c be the sum of all corrupt inputs, $x_h = x - x_c$ the sum of all honest inputs, and $\widehat{y} = y + x_h$; then, compute fresh sharing $(\widehat{\sigma_1}, \ldots, \widehat{\sigma_n}) \leftarrow \mathsf{SimShare}_t(\widehat{y}, \sigma_T)$
- continue as in the real protocol, using $\widehat{\sigma_j}$ as the share sent by C_j to the server

Figure 5: The simulator for statistical computations.

already seen by the corrupt clerks. By linearity of the scheme these fresh shares $\widehat{\sigma}$ will be distributed identically to the sum σ of the shares sent by users, meaning that the adversary cannot distinguish between the two executions. □

3.2 Towards Active Security

In the following we assume passive security for the bulletin board and focus on the other parties; if this is judged to be unreasonable we may instead replace it with secure channels between the users and the clerks as mentioned earlier. Moreover, notice that active security against a corrupt server is immediate due to the fact that its only role (perhaps besides committee election) is to receive an output.

We next consider two options for active security against corrupt clerks. The first options works regardless of the encryption scheme used but puts extra conditions on the choice of parameters that may be used for the secret sharing scheme.

Corollary 3.2. *By using robust Shamir for* $(\mathsf{Share}, \mathsf{Recon})$*, Theorem 3.1 holds under its original assumptions, even adding the extra property that at most* $(n - t)/2$ *among the corrupt committee members* $A \cap C$ *are actively corrupt.*

For the second option we assume that a homomorphic encryption scheme is used such that decryption is the only task clerks have to perform, and that there exists efficient proofs for proving correct decryption. As mentioned earlier this is for instance the

case for Paillier where the randomness of a ciphertext may be extracted as part of the decryption process and sent along as the proof.

Corollary 3.3. *By using an N-additive homormophic encryption that allows proof of correct decryption (e.g. Paillier), Step 8 compresses to a single ciphertext $\mathcal{E}_j$; having each clerk C_j send a proof π_j of correct decryption of $\mathcal{E}_j$ in Step 10 and having S check this proof before proceeding, makes Theorem 3.1 secure even if any of the clerks are actively corrupt.*

We leave active security against corrupt users as future work. Here, one is primarily interested in guaranteeing correct secret sharing. Moreover, to avoid "answer pollution" one would often also be interested in range proofs guaranteeing that the inputs of users are from a subset of $\mathbb{Z}_p^k$.

3.3 Performance Analysis

We here give a quick theoretical performance analysis of the communication cost of the aggregation protocol in terms of upload for users and download for clerks. We return to this empirically in Section 5.

Note first that the setting with only a single clerk may be seen as using a trivial $(n = 1, r = 1, t = 0, k = 1)$-secret sharing scheme. By doing so we may compare the different settings through the following (relative) parameters:

- *reliability*, or the overhead of total number of shares against shares needed to reconstruct, $\rho = \frac{n}{r}$
- *privacy*, or the ratio of corrupt clerks allowed, $\tau = \frac{t}{n}$
- *sharing expansion*, or the ratio between total number of shares and number of packed secrets, $\epsilon = \frac{n}{k}$

with the aim of minimising the latter and maximising the other.

For instance, with trivial secret sharing we have $\rho = 1$, $\tau = 0$, and $\epsilon = 1$, the latter being the only good property, while using an $(n = 26, r = 15, t = 5, k = 10)$-scheme we have $\rho = 1.7$, $\tau = .2$, and $\epsilon = 2.6$, meaning better reliability and privacy at the price of more than twice the size.

Moving on to the encryption side of the scheme, given a bound p on the (absolute) value of shares, we define the *encryption content* β as the number of shares that can be packed into a single encryption, and the *encryption expansion* γ as the relative encryption length per share (analogous to the sharing expansion). For example, if we assume 4-byte shares and instantiate the Paillier encryption scheme with a 2048-bit modulus we have

$$\beta = \left\lfloor \frac{2048}{\lceil \log_2(\alpha) \rceil + \lceil \log_2(p) \rceil} \right\rfloor = \left\lfloor \frac{2048}{20 + 32} \right\rfloor = 39$$

when we leave room for more than a million additions (2^{20}). This gives an *encryption expansion* factor of

$$\gamma = \frac{\mu}{\beta \cdot \lceil \log_2(p) \rceil} = \frac{4096}{39 \cdot 32} \approx 3.3$$

where μ is the bit size of encryptions. As another example, if we use a typical stream cipher then we get $\alpha = \beta = 1$, and $\gamma \approx 1$ if we assume $\mu = \lceil \log_2(p) \rceil + \mathcal{O}(1^\kappa)$.

The number of bits each user has to upload is then

$$\mathcal{O}(1^\kappa) + \left\lceil \frac{\lceil \frac{d}{k} \rceil}{\beta} \right\rceil \cdot n \cdot \mu$$

and the number of bits each clerk has to download will be

$$\left\lceil \frac{\lceil \frac{d}{k} \rceil}{\beta} \right\rceil \cdot \left\lceil \frac{N}{\alpha} \right\rceil \cdot \mu$$

which may be approximated by respectively

$$\frac{d}{\beta} \cdot \frac{n}{k} \cdot \mu = d \cdot \log_2(p) \cdot \epsilon \cdot \gamma$$

and

$$\frac{d}{\beta} \cdot \frac{N}{\alpha \cdot k} \cdot \mu = d \cdot \log_2(p) \cdot \frac{N \cdot \gamma}{\alpha \cdot k}$$

which, for the case of upload, shows the expected overhead of $\epsilon \cdot \gamma$ compared to e.g. the local DP model where all (noised) input values are sent in the clear. Regarding download, we see that we have a reduction by a factor of k; when α is small (in which case γ is also typically small, e.g. $\alpha = \gamma = 1$) this is a way to reduce download while keeping essentially the same upload, by simultaneously increasing both n and k to keep the same sharing expansion (in general, $N \geq n$); however, when $\alpha \approx N$ reducing download cost becomes less interesting since upload will quickly become the limiting factor anyway, with the upload-to-download rate being $\epsilon \cdot k \cdot \frac{\alpha}{N} \approx n$.

In summary, for sufficiently large values of d and N, using a good additive homomorphic scheme is the obvious choice, limited primarily by the upload size.

4 Obtaining Differential Privacy

In this section we extend the aggregation protocol of Section 3 to further protect the privacy of the input providers: while the previous protocol ensures privacy of intermediate values, nothing so far has been done to prevent learning too much by seeing the final aggregated output. Following standard practice, we switch from the absolute privacy guarantee of secure computation to a quantified measure provided through the notion of *differential privacy*. In this framework, uncertainty is used to reason about the privacy loss endured by any single individual, in one interpretation ensuring plausible deniability. Randomness drawn according to certain distributions are typically added to the output as a means of achieving this, which will indeed also be the approach we take here. However, for privacy it is crucial that this randomness, or *noise*, remains unknown and hence cannot simply be generated and added by for instance the company.

We here focus on noise drawn from a Binomial distribution due to its discrete nature, but note that processes for sampling from other distributions (see e.g. [ÁC11, SCR+11, EDG14]) would also work.

4.1 Differential Privacy

To simplify matters we here instantiate the standard definitions of differential privacy [DR14] to match our aggregation protocol. We first assume that our base field is $\mathbb{Z}_p = \left[-\lfloor \frac{p}{2} \rfloor, \lfloor \frac{p}{2} \rfloor\right]$ for some odd prime p much large than required by a given application, such that wrapping occurs only with an insignificant probability even when (a small amount of) noise is added. This allows us to assume that all operations are in $\mathbb{Z}$ below. We then consider the generalised aggregation protocol where the inputs are vectors $x \in \mathcal{X} \subseteq \mathbb{Z}^d$ and the *datasets* $D, D' \in \mathbb{N}^{\mathcal{X}}$ are represented by a count of distinct values. Note that the aggregation protocol can naturally be seen as computing a function $f : \mathbb{N}^{\mathcal{X}} \to \mathbb{Z}^d$ on these domains.

The standard definition of differential privacy then ensures that what is released about a dataset by a randomized process, or *mechanism*, only changes the probability of any particular event by a small amount, as quantified by the ϵ parameter. For a mechanism $\mathcal{M}$ with domain $\mathbb{N}^{\mathcal{X}}$ and range $\mathbb{Z}^d$ we have:

Definition 4.1 (Differential Privacy)**.** A mechanism $\mathcal{M}$ is (ϵ, δ)-*differentially private* if for all $S \subseteq \mathbb{Z}^d$ and all $D, D' \in \mathbb{N}^{\mathcal{X}}$ we have

$$\Pr\left[\mathcal{M}(D) \in S\right] \leq \mathbf{e}^{\epsilon} \Pr\left[\mathcal{M}(D') \in S\right] + \delta$$

when D, D' are *adjacent* datasets, i.e. $\|D - D'\|_1 \leq 1$. Here the additional δ parameter intuitively denotes a small probability by which the privacy guarantee breaks down.

A simple but useful fact about the definition of differential privacy is that it holds under post-processing, meaning that if a subprocess ensures differential privacy, then this guarantee will also hold no matter what happens in the remaining part of the process.

Proposition 4.2 (Post-Processing)**.** *Let $\mathcal{M}$ be a mechanism that is (ϵ, δ)-differentially private, and let $\mathcal{A}$ be a randomized algorithm. Then $\mathcal{A} \circ \mathcal{M}$ is (ϵ, δ)-differentially private.*

Turning to concrete constructions later, a common way of obtaining differential privacy fitting with our protocol is by simply adding noise to the output of the aggregation function.

Construction 4.3 (Abstract Additive Mechanism)**.** For function f, the *additive mechanism* $\mathcal{M}_{\mathcal{D}}$ on input $D \in \mathbb{N}^{\mathcal{X}}$ is given by the process that first computes $f(D)$, samples $\nu \leftarrow \mathcal{D}$, and returns their sum $f(x) + \nu \in \mathbb{Z}^d$.

Typical noise distributions for the additive mechanism include Gaussian and Laplace, yet here we focus on the discrete case and instead consider the Binomial distribution as in [DKM+06]. An important step in proving privacy of the additive mechanism for a specific distribution is to reason about the magnitude of the noise with respect to the *sensitivity* $\Delta_\ell f = \max \|f(D) - f(D')\|_\ell$ of function f, for adjacent datasets D, D'; intuitively, the magnitude of the noise must match the impact a single individual can have on the aggregation.

4.2 Private Aggregation Protocol

Extending the protocol from Section 3 to ensure differential privacy is relatively straightforward, by for instance having the clerks collectively generate a secret sharing of an (partially) unknown noise sample, that is then mixed in as part of the normal aggregation of inputs[5]. Note that noise generation may furthermore be done independently, such as in a parallel or offline phase, offering some flexibility in the requirements put on the involved parties.

To generalise the treatment slightly, in Figure 7 we extend the aggregation protocol with a call to the ideal functionality $\mathcal{F}_{\mathsf{noise}}$ given in Figure 6 for generating noise shares. This functionality simply shares a private noise sample from the specified distribution, added together with an extra noise component determined and known by the adversary. Note that both are parameterised by a noise distribution $\mathcal{D}$ (as well as parameters n, t, and k left out for simplicity) that ultimately determines the guaranteed privacy level.

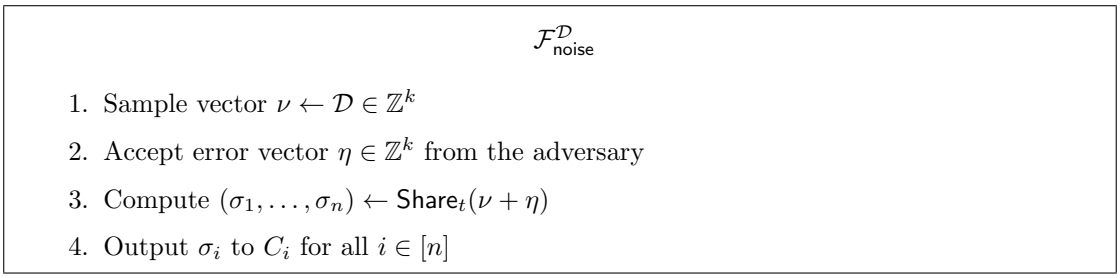

Figure 6: General noise generation functionality.

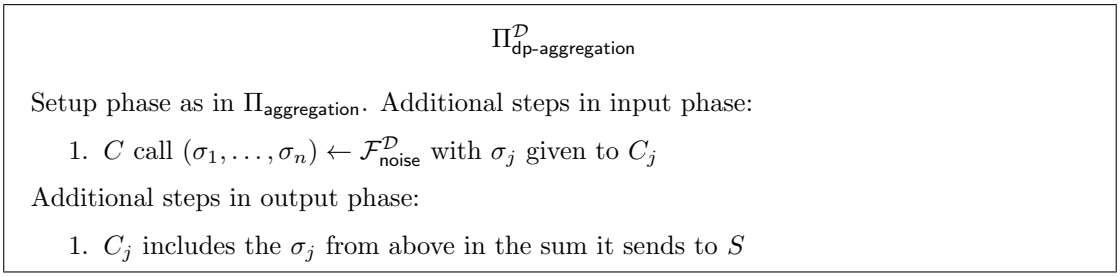

Figure 7: Our extended protocol that ensures differential privacy.

To see why $\Pi_{\mathsf{dp\text{-}aggregation}}$ ensures differential privacy when instantiated with appropriate distributions, note that the pertubated output is of form $f(x) + \mathcal{D} + \mathcal{E}$ for some independent error distribution $\mathcal{E}$ controlled by the adversary. By assumption $f(x) + \mathcal{D}$ is differentially private, and hence by the post-processing lemma so is the pertubated output.

[5] In practice, one difference compared to input sharings is that care has to be taken to prevent the bulletin board from leaving the noise sharing out of the aggregation. A possible remedy is to have the clerks sign their sharings and reject the computation unless enough valid noise shares are included.

Theorem 4.4. *Let $\mathcal{D}$ be a distribution such that the additive mechanism drawing noise from $\mathcal{D}$ is (ϵ, δ)-differentially private. If there are at most t colluding clerks then $\Pi^{\mathcal{D}}_{\mathsf{dp\text{-}aggregation}}$ ensures (ϵ, δ)-differential privacy.*

4.3 Binomial Realisation

Following [DKM+06, CRFG12] we realise $\mathcal{F}_{\mathsf{noise}}$ for our setting using noise drawn from the (shifted) Binomial distribution $\mathcal{B}(s)$, approximating the Gaussian distribution $\mathcal{N}(\frac{s}{2})$, and ensuring differential privacy under suitable choices of ϵ and δ depending on the sensitivity of f. As done there we use the following process for drawing samples from this distribution:

Construction 4.5 (Binomial Sampling)**.** Let $b_1, \ldots, b_{2s} \in \{-1, 1\}$ be independent samples of an unbiased coin. Then $\sum_i b_i$ is distributed according to $\mathcal{B}(s)$ with median zero.

Hence, to obtain a sample of $\mathcal{B}(s)$ we let each of the n clerks sample $\frac{2s}{n-t}$ coins and secret share their sum with the other clerks. This produces some "left-over" noise but accounts for the fact that up to t of the sample sums may be known and generated by clerks colluding with the server[6].

Our Binomial noise generation protocol Π_{binomial} is shown in Figure 8. While it requires one synchronisation between the clerks to exchange shares, note that this does not increase the total number of synchronisation points in the executing of an aggregation since it can be run in parallel with the input providers delivering their inputs.

$\Pi^{s}_{\mathsf{binomial}}$

1. Let $\hat{s} = \left\lceil \frac{2s}{n-t} \right\rceil$ and $\mathcal{C}$ the uniform distribution over $\{-1, 1\}$
2. C_j samples $\nu_{j,\ell} \leftarrow \mathcal{C}^k$ for $\ell \in [\hat{s}]$
3. C_j computes $\nu_j = \sum_\ell \nu_{j,\ell}$
4. C_j computes $(\sigma_{j,1}, \ldots, \sigma_{j,n}) \leftarrow \mathsf{Share}_t(\nu_j)$
5. C_j sends $\sigma_{j,i}$ to C_i, for all $i \in [n]$

Figure 8: Binomial generation protocol.

Theorem 4.6. *$\Pi^{s}_{\mathsf{binomial}}$ realises $\mathcal{F}^{\mathcal{B}(s)}_{\mathsf{noise}}$ in the passive model when at most t clerks are corrupt.*

Proof. Assuming at most t corrupt clerks we consider the following simulator for showing passive security; in fact we show something slightly stronger, namely that we can get a

[6] To account for instability of the clerks we may wish to avoid insisting on receiving samples from $n - t$ clerks. To obtain the same guarantees we may than instead ask each clerk to sample $\frac{2s}{m}$ coins for some $1 \leq m \leq n - t$, let the server pick $m + t$ of the received sharings, and proceed by summing these. Note however, that the left-over noise grows as m decreases.

weaker form of active security under the assumption that secret sharing is always done honestly, making extraction possible.

The simulator runs the real protocol internally, keeping a record of the noise each clerk has shared. If asked to generate shares for an honest clerk it proceeds with coin sampling and secret sharing as in the protocol; if asked to distribute shares for a corrupt clerk it asks the adversary for a set of shares from which it extracts an error $\eta_i \in \mathbb{Z}^k$ via the reconstruction algorithm; in both cases it keeps a record of the shared value. When finally asked by the functionality for the error η, it returns the sum of the stored values from the $\hat{t} \leq t$ corrupt clerks and $t - \hat{t}$ arbitrarily picked honest clerks. □

5 Applications

Since the purpose of this work is to obtain a practical solution to secure aggregation we next discuss applications of our protocol and the concrete performance numbers it achieves on realistic datasets. We assume inputs from $\mathbb{Z}_p^d$ and focus on cases where the dimension d is required to be somewhat large, running the aggregation protocol $\lceil \frac{d}{k} \rceil$ times in parallel when needed. We see that the k parameter of the secret sharing scheme serves as a key element in scaling the work across clerks, both in terms of communication and computation.

Our main metric of interest is the amount of data uploaded by users and downloaded by clerks. Occasionally we also investigate the computational load based on results from our implementation outlined in Section 6. Throughout the section we consider the different secret sharing parameters shown in Table 2, which all use a modulus size less than 32 bits and ensure $r \leq \lfloor \frac{4n}{5} \rfloor$ and $t \geq \lfloor \frac{n}{5} \rfloor$. All data sizes are calculated based on Section 3.3, and all cryptographic computations are measured[7] on a Raspberry Pi (model 3), an iPhone 6, and a MacBook Pro (13 inch 2015), and are reported for one core, meaning parallelisation will cut the reported times depending on the number of available cores.

	n	**r**	**t**	**k**
small	26	15	5	10
medium	80	63	16	47
large	728	511	145	366

Table 2: Parameters considered for secret sharing

Finally, since summing does not naturally preserve correlation between features, we first briefly outline how an input from a set of correlated features may be represented in a way suitable for aggregation when needed by the application. Note however that for applications such as analytics, the natural representation without expansion is sufficient.

[7] Benchmarking code available at https://github.com/mortendahl/DPP17.

5.1 Correlation-preserving Representation

In general, the input of a user is given as a vector

$$f = (f_1, \ldots, f_\ell) \in \mathcal{F}_1 \times \cdots \times \mathcal{F}_\ell$$

taking values from a set of ℓ features $\mathcal{F}_j = \{f_{j,1}, \ldots, f_{j,|\mathcal{F}_j|}\}$. When it is essential for the application to maintain correlation between (some of) these features through the aggregation, it is convenient to first expand the input vectors into their unary representation as described next. This increases the size of the vectors significantly but may be needed in order to extract enough information from the aggregated sum output.

As an example, assume we have two binary features InUS and IsFrench as illustrated in the upper half of Table 3. In their natural representation these inputs are simply 2-dimensional vectors $[b_{\mathsf{InUS}}, b_{\mathsf{IsFrench}}]$ where each component is a bit. However, if we aggregate using this representation we loose their correlation and can for instance only extract that three of the users live in the US, but not how many of them both live in the US *and* are French nationals.

In order to maintain this correlation, each user can instead first map his input vector x_i to a vector $\hat{x}_i$ of length $\prod_j |\mathcal{F}_j|$, with 0 everywhere except for a single 1 at the entry corresponding to the index of x_i in $\mathcal{F}_1 \times \cdots \times \mathcal{F}_\ell$. Doing this for our example above is illustrated in the lower half of Table 3.

Name	**Lives in the US?**	**Is French?**
Alice	Yes	No
Bob	Yes	No
Charlie	No	Yes
David	Yes	Yes

⇓

Name	**No/No**	**No/Yes**	**Yes/No**	**Yes/Yes**
Alice	0	0	1	0
Bob	0	0	1	0
Charlie	0	1	0	0
David	0	0	0	1

Table 3: Comparison between original inputs and preprocessed ones.

Formally, for expansion we associate a matrix $F_j = \mathbb{I}_{|\mathcal{F}_j|}$ with each feature and replace an input $f = (f_{1,i_1}, \ldots, f_{\ell,i_\ell})$ with the unit vector at row $(i_1, \ldots, i_\ell)$ of $F_1 \otimes \ldots \otimes F_\ell$. This unit vector hence becomes the new input. By doing so, from the aggregated output $\hat{x}$ of a set of such unit vectors we may then for instance evaluate

$$\left(\mathbf{u}_3^\intercal \otimes F_2 \otimes \left[\sum \mathbf{u}_i^\intercal\right] \otimes \ldots \otimes \left[\sum \mathbf{u}_i^\intercal\right]\right) \cdot \hat{x}$$

to extract the distribution over $\mathcal{F}_2$ for users with $\mathcal{F}_1 = f_{1,3}$.

This representation blows up the input size exponentially (an input vector over e.g. d binary features gets compiled into one with 2^d components), which will be problematic in case the number of original attributes is large. Below we investigate several ways to mitigate this dimensionality issue.

5.2 Analytics

A simple but widely used method in many markets, not least web and mobile, is that of data-driven product development, where the actions of users are recorded and used to evaluate the performance of new features. Typically, these event logs are stored on a centralised server, but using the aggregation protocol to provide a private alternative.

We here assume an application counting the occurrences of 100 different events, meaning users hold input vectors of this dimension. For efficiency on web and mobile (focusing on the input providers' upload and computation time), we restrict ourselves from using homomorphic encryption and instead rely on stream ciphers; as such, the main bottleneck will be the individual download size (and decryption time) for the clerks. However, the clerks' work can be done as a background task, which means that our efficiency requirements here are less strict. Notice also that our test cases highlight both reasonable and interesting scenarios in which our solution works efficiently, but also scenarios that show its limits.

The communication sizes for various numbers of users are shown in Tables 4. If we aim for a clerk download size of no more than 1MB then we see that the protocol supports up to $25,000$ users with only 26 clerks. And as the number of users increases the download size can be kept below our limit by increasing the number of clerks; as expected this does not imply any significant change in the upload size of each user.

users	scheme	upload	download
$25,000$	small	1KB	977KB
$80,000$	medium	1KB	938KB
$250,000$	large	3KB	977KB

Table 4: Communication for analytics

As for computation, Table 5 shows that even users running Raspberry Pis could easily perform the operations needed for processing inputs: encryption is taking most of the time, yet still less than a few milliseconds. The story is slightly different for clerks of course, where decrypting the shares from $250,000$ users will take roughly 6 minutes; however, if running as a background task then this is still within reason.

device	users	scheme	share	encrypt	decrypt
RP	25,000	small	.840ms	53ms	37,275ms
RP	80,000	medium	.622ms	163ms	119,120ms
RP	250,000	large	1.599ms	1,485ms	372,259ms
MBP	25,000	small	.153ms	3ms	1,575ms
MBP	80,000	medium	.096ms	10ms	5,040ms
MBP	250,000	large	.242ms	94ms	15,500ms

Table 5: Computation for analytics (using one core)

5.3 Drug-use Survey

For this application we consider a real-world analysis performed on the drug use among different age groups [BF15]:

"The availability and public perception of different drugs change, making it difficult to compare the drug use of a given age group over time. We were, however, able to analyze a survey on drug use ... to see what drugs baby boomers are taking now. Are their patterns of use different from other age groups? How similar are people within the baby-boomer cohort when it comes to drug use?"

Given the sensitive nature of these questions, surveys for building the underlying data set seem likely to benefit from strong privacy guarantees. Here we ask under which parameters our protocol would have been efficient enough to carry out a data collection process of this scale (multiply features and more than 55,000 users). Concretely, the study considers:

- an age group feature $\mathcal{A}$ of cardinality 17,
- binary drug-use features $\mathcal{D}_1, \ldots, \mathcal{D}_{13}$,

meaning each user holds an input $f \in \mathcal{A} \times \mathcal{D}_1 \times \cdots \times \mathcal{D}_{13}$. Below we then consider how a table similar to that used in the study may be extracted from an aggregate of these inputs.

In the general case where we wish to answer queries about any correlation between the features, we are forced to perform the expansion described above. This however results in an expansion of the dimension of the input vector from 14 to $17 \cdot 2^{13} = 139,264$; with 55,268 users this means more than 28GB of unencrypted data alone, and hence it is no surprise that it leads to an unrealistic load on any reasonably sized set of clerks. But note that if the data collection had been done using this representation we would actually obtain more information than they have. Below we outline different approaches for dealing with this dimensionality issue while still being able to answer our target questions.

technique	dimension	scheme	upload	download
correlation	442	small	5KB	9.5MB
correlation	442	medium	4KB	2.1MB
sensitivity	8192	medium	55KB	36.9MB
sensitivity	8192	large	65KB	4.8MB
formulation	4862	medium	33KB	21.9MB
formulation	4862	large	40KB	2.7MB

Table 6: Communication for drug-use survey

Table 6 shows the communication sizes for the three dimensionality reduction techniques, again showing the significant impact on clerk download size achieved by switching to a scheme with a higher k parameter. Following the nature of such surveys we assume users and clerks running in web browsers, and hence rule out homomorphic encryption, relying instead on stream ciphers. This means that also here will the main bottleneck be the individual download size for clerk.

Feature Correlation If we are not interested in the correlation between certain features we can reduce the dimension of the problem by decoupling the unwanted features. Concretely, we may only be interested in the relationship between age and each drug, and not for instance between the use of two drugs. Hence, by instead running 13 aggregations of dimension $17 \cdot 2$ in parallel (or one with dimension 442) the medium setting with 80 clerks might be deemed plausible.

Feature Sensitivity Another way to reduce the dimension is to decide that one of the features is not sensitive and may be leaked. To obtain the same result, for each value of the feature we run a separate aggregation, with users only participating in the one matching their input. In this specific application, we may for instance decide that the age is not sensitive, and hence essentially run 17 aggregations of dimension $2^{13} = 8192$ in parallel, with each user only responding to the one matching his age. Note that this increase in dimension would most likely force us to switch to a large scheme with at least 728 clerks.

Question Reformulation Alternatively, to reduce dimensionality we may also simply reformulate the question. In this specific application, instead of asking which drugs each user has used, we may ask *"what are your top three drugs used?"*. This reduces the dimension to $17 \cdot \binom{13}{3} = 4862$. Note that we are again most likely forced to consider a large scheme.

5.4 Location Heatmaps

The places a user visits may naturally be used to enrich his profile, including learning e.g. music or food preferences, as well as more sensitive features. By using an external data source for places this may be done directly on mobile devices, limiting the privacy impact on users. However, it may also be interesting to have information flow in the other direction, allowing e.g. a company to learn properties about a place based on visits by its clientele. Specifically, assuming mobile phones holding the preferences of their users we may use the aggregation protocol to privately learn places frequented by a specific demographic of users.

For the concrete application we seek to discover areas of specific interest in a region the size of New York City. We do this by dividing the region into a grid of $160,000$ equally sized boxes and use the location data of our demographic to essentially vote on visited boxes, in the end obtaining a global popularity rating for each box without knowing where any particular user have been. This results in a height and width of each box of approximately 40 meters.

The main difficulty here lies in dimensionality, in that we deem it too heavy for users to send a vector of dimension $160,000$. As a remedy we use a Min-Count Sketch [CM05] to effectively reduce the number of boxes to $20,000$. This is possible since each user may generate a sketch locally, that may then be combined using the aggregation protocol. Moreover, while the compression introduces noise into the computation, as shown below it is still possible to recover the "heavy-hitter", i.e. the most popular places.

To obtain experimental results on the effectiveness of this approach we use a Foursquare dataset of places around New York City to model the data reported by users. For instance, we take the approximately $75,000$ places marked as shopping in this region and increment the counter by one for the containing box; one possible interpretation of this is that $75,000$ users all voted on their favourite location. The comparison between the real and compressed heat map is shown in Figure 9 with the real heatmap on the left, the heatmap recovered from the sketch in the middle, and the additive noise between the two on the right; the lower row shows a zoom of a select area. From the recovered heatmap it is clear that we can extract the heavy hitters with good accuracy (see also [MDC16] for a similar use-case with a more detailed analysis of accuracy).

As might be expected, not using homomorphic encryption for this dimensionality and number of users makes even our most efficient parameters from above result in a 16MB download for each clerk. By switching to the Paillier encryption scheme with parameters as described in Section 3.3 we see in Table 7 that a setting with 26 clerks increases the upload size for users slightly (but reasonably, assuming that the data collection happens on e.g. modern mobile phones) while eliminating the download bottleneck for clerks. Another benefit of course is that the bulletin board has to store much less data, by combining the Paillier encryptions immediately as they arrive.

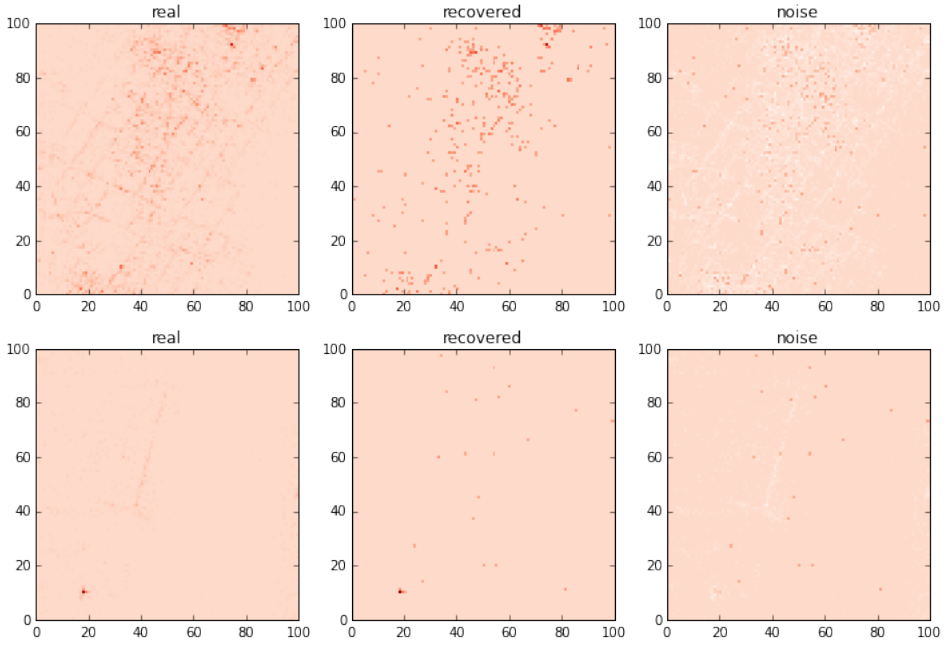

Figure 9: Heatmaps

encryption	scheme	upload	download
Stream	large	156KB	16,113KB
Paillier	small	676KB	26KB

Table 7: Communication for heatmaps

5.5 Movie Ratings

As a stress-test we here consider the large scale Netflix Prize use case [DCZ10], where roughly 100 million ratings for 17,770 movies are aggregated. We assume each vote to be between 0 and 5, and use an extra binary value to simultaneously count the number of ratings each movie has, in turn allowing us to derive an average. In other words, 100 million users holding input vectors of dimension 35,400.

For the setting with 26 clerks, each input vector is split into 3540 sharings. To avoid wrapping we leave additional room in the Paillier packing compared to Section 3.3, meaning each component will be $\lceil \log_2(5 * 100,000,000) \rceil + 32 = 61$ bits; using a 2048 bit plaintext modulus means packing $\lfloor \frac{2048}{61} \rfloor = 33$ values into each ciphertext, for a total of $\lceil \frac{3540}{33} \rceil = 108$ ciphertexts per clerk. This means an upload of roughly 1.4MB for each user and a download of 54KB for each clerk.

Leaving aside the feasibility of this in terms of communication, the fact that it

device	scheme	share	decrypt
RP	small	299ms	21,564ms
RP	medium	157ms	4,593ms
RP	large	158ms	599ms
iPhone	small	321ms	6,181ms
iPhone	medium	289ms	1,318ms
iPhone	large	327ms	172ms
MBP	small	59ms	981ms
MBP	medium	25ms	208ms
MBP	large	25ms	27ms

Table 8: Computation for movie ratings (using one core)

results in 108 decryption operations for the clerks may imply a bottleneck in terms of computation. Table 8 shows these computational requirements, and again we see that by increasing the k parameter of the scheme we can lower the investment each individual clerk has to make, to under one second even in the Raspberry Pi case. At the same time, this does not significantly change the time it takes each user to process her input.

6 Implementation

We here discuss a few details of our implementation. We have used the Rust language for efficiency and portability across different platforms and architectures. As a stream cipher we use the Sodium library[8].

6.1 Efficient Packed Secret Sharing

Our implementation of secret sharing[9] is as follows. Let $x_1, \ldots, x_{t+k}, y_1, \ldots, y_n$ be distinct elements in a finite field. To generate shares for the packed secret sharing scheme, where we want to simultaneously share secrets $m_1, \ldots, m_k$ into shares $\sigma_1, \ldots, \sigma_n$ with privacy threshold t, we first sample a uniformly random degree $t + k - 1$ polynomial

$$f(X) = \sum_{i=1}^{t+k} f_i X^{i-1}$$

such that $f(x_i) = m_i$. Then, to obtain the shares we simply evaluate this polynomial at points $y_1, \ldots, y_n$.

[8] Available at https://libsodium.org.
[9] Available at https://github.com/snipsco/rust-threshold-secret-sharing.

Sampling f and obtaining its coefficient representation $(f_1, \ldots, f_{t+k})$ may be done by interpolating point-value pairs

$$\big((x_1, m_1), \ldots, (x_k, m_k), (x_{1+k}, r_1), \ldots, (x_{t+k}, r_t)\big)$$

for uniformly random field elements $r_1, \ldots, r_t$. To do both interpolation and evaluation efficiently we use the Fast Fourier Transform (FFT) over finite fields in respectively its backward and forward direction, yielding a running time of $\widetilde{\mathcal{O}}(t+k)$ and $\widetilde{\mathcal{O}}(n)$. Note that we may need to append extra zero coefficients to f before evaluation since $n \geq k+t$ in general.

For the above applications of the FFT, we make some additional assumptions on the field $\mathbb{F}$, the points $x_1, \ldots, x_{t+k}$ and $y_1, \ldots, y_n$, and the form of $t+k$ and n. Specifically, we assume that $\mathbb{F} = \mathbb{Z}_p$ for some prime p of form $2^a \cdot 3^b \cdot c + 1$, which guarantees the existence[10] of suitable ω_x and ω_y as respectively 2^a-th and 3^b-th primitive roots of unity such that their respective multiplicative subgroups only have the identity in common as needed for security. Moreover, we assume that the parameters for the scheme is chosen such as $t + k = 2^a - 1$ and $n \leq 3^b - 1$, the former requirement on equality being due to the fact that the sampled polynomial will have degree 2^a and we want this to match the expected reconstruction threshold. Under these assumptions we let $x_i = \omega_x^i$ and $y_j = \omega_y^j$ so that we can apply the two FFTs with efficiency as claimed above. Note that finding a set of suitable parameters may be precomputed and shared publicly.

As for reconstruction, we unfortunately cannot use the FFT since it implicitly assumes that the polynomial has degree n, and hence given only as many shares as required by the reconstruction threshold we cannot simply "patch up" the missing shares with zero. Moreover, we cannot use the FFT used for sampling either, since it considers a different set of points. However, since reconstruction is only done once and on the server, efficiency is not of big importance and we default to the Newton/Neville algorithm.

6.2 Paillier

Our implementation of the Paillier scheme[11] is somewhat straight-forward, using either GMP or RAMP as the underlying arbitrary precision arithmetic library.

We use the standard optimisations for efficiency, choosing $g = 1 + n$ as the generator and the simplifications it brings [DJ01], and decrypting using the Chinese Remainder Theorem [Pai99]. At this point in time not much has been done to adapt the implementation to our protocol, and the repeated nature of our application of the scheme could likely yield a further reduction in the reported computation times. Moreover, by splitting encryption into an online and offline phase (computing respectively g^m and r^n) we could significantly improve the time users spent in the input phase.

[10] Given a generator g of $\mathbb{Z}_p^\star$ we may construct them as $\omega_x = g^{(p-1)/2^a}$ and $\omega_y = g^{(p-1)/3^b}$. Of course, other choices of primes would also work.

[11] Available at `https://github.com/snipsco/rust-paillier`.

References

[ÁC11] Gergely Ács and Claude Castelluccia. *I Have a DREAM! (DiffeRentially privatE smArt Metering)*, pages 118–132. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2009.

[BF15] Anna Maria Barry-Jester and Andrew Flowers. How baby boomers get high, 2015. http://fivethirtyeight.com/datalab/how-baby-boomers-get-high/.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Simon [Sim88], pages 1–10.

[BLS12] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. *The Security Impact of a New Cryptographic Library*, pages 159–176. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 97–106, Washington, DC, USA, 2011. IEEE Computer Society.

[CAF13] Ruichuan Chen, Istemi Ekin Akkus, and Paul Francis. Splitx: High-performance private analytics. *SIGCOMM Comput. Commun. Rev.*, 43(4):315–326, August 2013.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In Simon [Sim88], pages 11–19.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2000.

[CM05] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58 – 75, 2005.

[CRFG12] Ruichuan Chen, Alexey Reznichenko, Paul Francis, and Johanes Gehrke. Towards statistical queries over distributed private user data. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 169–182, San Jose, CA, 2012. USENIX.

[CSS12] T. H. Hubert Chan, Elaine Shi, and Dawn Song. *Privacy-Preserving Stream Aggregation with Fault Tolerance*, pages 200–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[DCZ10] Yitao Duan, John Canny, and Justin Zhan. P4p: Practical large-scale privacy-preserving distributed computation robust against malicious users. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 14–14, Berkeley, CA, USA, 2010. USENIX Association.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2005.

[DIK+08] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam D. Smith. Scalable multiparty computation with nearly optimal work and resilience. In David Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 241–261. Springer, 2008.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, PKC '01, pages 119–136, London, UK, UK, 2001. Springer-Verlag.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure mpc for dishonest majority - or: Breaking the spdz limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

[DKM+06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. *Our Data, Ourselves: Privacy Via Distributed Noise Generation*, pages 486–503. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(34):211–407, 2014.

[EDG14] Tariq Elahi, George Danezis, and Ian Goldberg. Privex: Private collection of traffic statistics for anonymous communication networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1068–1079, New York, NY, USA, 2014. ACM.

[EPK14] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1054–1067, New York, NY, USA, 2014. ACM.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 554–563. ACM, 1994.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. http://eprint.iacr.org/2012/144.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *STOC*, pages 218–229. ACM, 1987.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *Proceedings of the 31st Annual Conference on Advances in Cryptology*, CRYPTO'11, pages 132–150, Berlin, Heidelberg, 2011. Springer-Verlag.

[HMRT12] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient rsa key generation and threshold paillier in the two-party setting. In *Proceedings of the 12th Conference on Topics in Cryptology*, CT-RSA'12, pages 313–331, Berlin, Heidelberg, 2012. Springer-Verlag.

[HN12] Michaela Hardt and Suman Nath. Privacy-aware personalization for mobile advertising. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 662–673. ACM, 2012.

[KMR11] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multiparty computation. *IACR Cryptology ePrint Archive*, 2011:272, 2011.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. *IACR Cryptology ePrint Archive*, 2016:505, 2016.

[LP11] Richard Lindner and Chris Peikert. *Better Key Sizes (and Attacks) for LWE-Based Encryption*, pages 319–339. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1219–1234, New York, NY, USA, 2012. ACM.

[MDC16] Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient private statistics with succinct sketches. *NDSS*, 2016.

[MS81] Robert J. McEliece and Dilip V. Sarwate. On sharing secrets and reed-solomon codes. *Commun. ACM*, 24(9):583–584, 1981.

[Pai99] Pascal Paillier. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*, pages 223–238. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

[RN10] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 735–746, New York, NY, USA, 2010. ACM.

[SCR+11] Elaine Shi, T-H. Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. 2011.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[Sim88] Janos Simon, editor. *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. ACM, 1988.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250. Springer, 2015.