Zeph: Cryptographic Enforcement of End-to-End Data Privacy

Lukas Burkhalter*, Nicolas Küchler*, Alexander Viand, Hossein Shafagh, Anwar Hithnawi
Department of Computer Science
ETH Zürich

Abstract

As we increasingly expose sensitive data to gain valuable insights, the need to natively integrate privacy controls in data analytics frameworks is growing in importance. Today, privacy controls are enforced by data curators with full access to data in the clear. However, a plethora of recent data breaches have shown that even widely trusted service providers can be compromised. Additionally, there is no assurance that data processing and handling comply with the claimed privacy policies. This motivates the need for a new approach to data privacy that can provide strong assurance and control to users. This paper presents Zeph, a system that enables users to set privacy preferences on how their data can be *shared* and *processed*. Zeph enforces privacy policies cryptographically and ensures that data available to third-party applications complies with users' privacy policies. Zeph executes privacy-adhering data transformations in real-time and scales to thousands of data sources, allowing it to support large-scale low-latency data stream analytics. We introduce a hybrid cryptographic protocol for privacy-adhering transformations of encrypted data. We develop a prototype of Zeph on Apache Kafka to demonstrate that Zeph can perform large-scale privacy transformations with low overhead and latencies.

1 Introduction

The availability of rich data and the advancement of tools and algorithms to process data at scale has enabled tremendous innovations in various fields ranging from health and retail to agriculture and industrial automation [59, 62, 72, 74]. However, the accumulation of sensitive data has made service providers hosting data lakes a desirable target for attacks. In addition, a surge of incidents of unauthorized data monetization, instrumentation, and sharing has raised societal concerns [46, 78]. This has pushed regulatory bodies to enact data privacy regulations to prevent misuse of private data and ensure the privacy of personal data [2, 3]. Today, the most integral parts of existing data protection systems are security controls such as authentication, authorization, and encryption which protect data by guarding it and limiting unnecessary exposure. Security controls alone, however, are not sufficient. We ultimately need to ensure that user's privacy is respected even by entities authorized to use the data. Thus, privacy solutions that control the extent of what can be *inferred* [12] from data and protect *individuals' privacy* [42] are crucial if we are to continue to extract *utility* from data safely.

Today's Data Privacy Landscape: The advent of new data privacy regulations such as GDPR and CCPA, coupled with the increasing importance of data, has led to growing demand for privacy solutions that protect sensitive data while retaining its value. Despite recent advancements in privacy technologies (e.g., differential privacy [37]), privacy frameworks [24, 39, 40, 55, 69, 76] remain shaped by regulatory requirements that predominately focus on the notion of *notice and consent* [4, 8, 53]. This status quo has three shortcomings that we aim to address with this work: *(i) Trusted data curators:* In the current model, privacy controls are implemented and enforced by data curators who have full access to data in the clear. Frequent data breaches [23, 34, 58] have shown that even trusted providers can be compromised or fall prone to data misuse temptations. Additionally, there are no assurances that data processing actually complies with the stated privacy policies. Consequently, there is a need for built-in data privacy mechanisms that do not require data curators to access data in the clear. *(ii) Lack of user control:* Though privacy regulations mandate services to grant users more control over their data, the materialization of this has been disappointing in practice. Services have been drafting privacy policies that unilaterally dictate how users' data will be used. Users have no option to exert their data privacy preferences except to give blanket consent if they choose to use the service [44, 53]. *(iii) End-to-end privacy:* Privacy solutions today are mostly ad hoc efforts [11] rather than an integral part of the data processing ecosystem. We need a cohesive end-to-end approach to data privacy that follows data from source to downstream. Such solutions should integrate with existing data processing and analytics frameworks and coexist with data protection mechanisms already in place.

Zeph: In this work, we propose Zeph, a new data privacy platform that provides the means to safely extract value from encrypted data while ensuring data confidentiality and privacy by serving only privacy-compliant data. Our design targets data stream analytics/processing pipelines and builds on the typical structure of such systems. Streaming compute tasks are increasingly relevant in various privacy-sensitive sectors [14, 31, 43, 49]. The online nature of stream processing makes low latency and high throughput critical requirements for privacy-preserving stream processing solutions.

*equal contribution

Zeph addresses the above shortcomings with two key ideas: *(i)* a user-centric privacy model that enables users to express their privacy preferences. In Zeph, a user can authorize services to access raw data or privacy-compliant data securely. This aligns with data sharing practices claimed in privacy policies today: e.g., "we share or disclose your personal data with your consent" or "we only provide aggregated statistics and insights" [5, 10]. In addition to this commonly referenced aggregation policy, Zeph supports more advanced privacy-compliant data transformations. For example, transformations that restrict what can be inferred from the data (e.g, generalization techniques [21, 64, 71]) or ensure differential privacy – a mathematically rigorous form of privacy transformations. *(ii)* Zeph cryptographically enforces privacy compliance and executes privacy transformations on-the-fly over encrypted data, ensuring that the generated transformed views conform to users' privacy policies.

Cryptographically Enforced Privacy Transformations. There are three key challenges in designing a data platform that enables privacy-compliant data transformations on encrypted data. First, we need to ensure compatibility with the data flow of existing data processing pipelines (e.g., storage and compute) and meet their strict performance requirements. Second, the platform must enable a wide range of existing privacy transformations and allow for different transformations to be applied to the same underlying data. Finally, in addition to single-source privacy transformations, we need to support transformations that require combining data from multiple users (e.g., differentially private data releases).

Existing practical encrypted data processing systems generally use partially homomorphic encryption schemes that already support the single-source transformation required in our system [29, 35, 48, 60, 61, 73]. However, homomorphic evaluation alone is insufficient to support aggregations across data from different users. Supporting these functions is typically achieved via multi-party computation protocols that are optimized for aggregation operations [13, 35, 56, 65]. These protocols ensure that user inputs remain private and only the aggregation result is revealed to the server. However, these protocols are either limited to specific functions (e.g., updating sketches) or require the data producers to take an active part in the computation.

We address these challenges in Zeph using two key ideas: *(i)* a new approach for encryption that decouples data encryption from privacy transformations. This logical separation of the data and privacy planes allows us to remain compatible with data flows in existing systems. Data producers remain oblivious to the transformations and do not need to encrypt data towards a fixed privacy policy. *(ii)* we introduce the concept of cryptographic *transformation tokens* to realize flexible data transformations. These tokens are, in essence, the necessary cryptographic keying material that enables the respective transformation on encrypted data. Zeph creates these tokens via a hybrid construction of secure multi-party computation

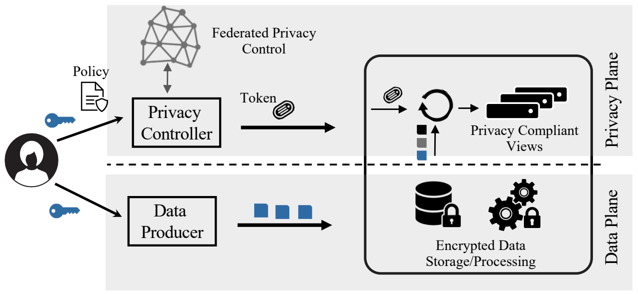


Figure 1: Zeph's end-to-end approach to privacy.

(MPC) and partially homomorphic encryption schemes. Outputs of privacy transformations over encrypted data at the server-side are then released by combining the encrypted data with corresponding cryptographic transformation tokens.

We have built a prototype of Zeph that is interfaced with Apache *Kafka* [18]. Our evaluation results show that Zeph can serve real-time privately transformed streams in different applications with a 2x to 5x latency overhead compared to plaintext. We optimize the interactive part of the underlying MPC protocol with ideas from graph theory to achieve the scalability requirements of Zeph. Our optimization improves performance up to 55x compared to the baseline.

2 Overview

In this section, we discuss the requirements of end-to-end privacy in our setting and give an overview of Zeph, describing our security and privacy model and our assumptions.

2.1 Design Requirements

We begin by defining the core criteria that a practical end-to-end privacy design for stream processing system must achieve.

User-Centric Privacy. Users should be able to set their privacy preferences and define how their data can be accessed and shared. In practice, users' preferences can vary with respect to the trade-offs between increased privacy and utility. While we want to offer users the option of strong privacy guarantees, we also need to provide options for more relaxed privacy guarantees when incentives to do so exist. For example, users might voluntarily share their off-platform shopping activities with a service provider in return for financial incentives [63]. Therefore, a practical system must support a range of privacy preferences and be able to build privacy-compliant views across data covered by heterogeneous policies.

Seamless Integration. A practical privacy solution should integrate seamlessly into existing data pipelines while ensuring the privacy of the underlying data. Additionally, privacy transformations need to respect/adhere to traditional data protection mechanisms already in place (i.e., end-to-end encryption). Therefore, the design needs to offer composability to support a variety of privacy solutions and ensure that privacy solutions can work in encrypted settings.

Performance. Stream processing has emerged as an important form of large-scale data processing [14, 30, 31, 43, 49, 77]. Stream processing functions continuously compute a long-running query over an unbounded stream of data records. This requires low-latency and high-throughput data processing to handle large data ingests from distributed sources in real-time. Therefore, privacy transformations must be scalable, fault-tolerant, and introduce minimal overhead.

Privacy Transformations. Privacy solutions for data analytics are designed to enable extracting utility of data while preserving the privacy of individuals. While there are many proposed solutions in this space (e.g., differential privacy [36], aggregation [26], data masking [71], data minimization [45], approximation [56] or other generalization techniques [21, 64]), they generally fall into three broad categories: *(i)* preserving individual's privacy by combining it with other users' data, *(ii)* limiting what can be inferred from the data by removing, masking, or approximating sensitive aspects of the data, *(iii)* combinations of both approaches. Therefore, for a privacy system to be flexible in regards to the applications and data types it can serve, it needs to support techniques in both approaches and their combinations.

Multi-Source Privacy Transformations. While *single-source* transformations already allow limiting what can be inferred from a stream (e.g., data resolution [71], sketching for approximation [56]), many widely used privacy policies inherently require *multi-source* transformations [22]. For example, heart rate measurements from a clinical trial might only be shareable once aggregated across a particular population size. While frequently used in practice, aggregation across users by itself does not provide rigorous privacy guarantees [36]. However, it can be extended to provide *differential privacy*, which ensures that the output is distributed approximately the same, whether a particular user's data was used or not [13, 36]. Hence, we need to additionally provide support for multi-source *differentially private* transformations that introduce carefully crafted noise to the final output of a transformation to achieve these guarantees.

2.2 Zeph in a Nutshell

Zeph is a privacy platform that augments encrypted stream processing pipelines with the means to cryptographically enforce privacy controls. Figure 2 shows an overview of Zeph's design. We aim to enable authorized third-party services to access and process data and to gain insights from it without violating the privacy preferences of the data owners. We design Zeph such that it can encapsulate state-of-the-art privacy solutions (e.g., generalization, differential privacy) without changing existing data streaming pipelines.

We separate the privacy control plane from the underlying data plane that encrypts and forwards data streams. The privacy logic resides and is executed outside the data plane, allowing the *data sources* to continue to write encrypted data streams to a remote stream processing pipeline as before. Zeph exposes an API for *data owners* to set their privacy preferences, which forms the base for users' privacy policies. In the current design, we keep the set of privacy preferences generic across applications and small (i.e., no sharing, share without restrictions, share data in isolation but limit what can be inferred from data, share data only when aggregated across users). These options capture trends in modern privacy solutions (i.e., generalization, data minimization, data masking) and support a wide range of common privacy policies. Users can choose different policies for each stream.

Data owners outsource policy enforcement to their *privacy controller*. The privacy controller does not require access to the data and can be hosted in a location with higher availability guarantees. For example, the controller could run as an on-premise service for a large organization with a strict trust model. The privacy controller is responsible for supplying the privacy transformation tokens that enable privacy-transformations at the server. As some privacy policies require data to be aggregated across different users before being made available, generating tokens specific to these types of transformations require interaction between several privacy controllers in what we refer to as *federated privacy control*. While the tokens generated by the privacy controllers cryptographically enforce the data owners' privacy policies, the server is responsible for composing and executing transformation efficiently.

We distinguish between two types of *data consumers*: (a) services that access the data to provide utility to the user (i.e., personalization), and (b) third-party services, e.g., to provide a utility that is beneficial to the public or the service itself, but not directly to the user (e.g., Strava [9] uses your running data to create a heat map of popular running routes in a privacy-preserving manner or allow your health data records to contribute to a medical study). Enabling direct access to the data for the first type of data consumers is handled by cryptographic access control and is supported in our design, but it is not the focus of this work. In Zeph, we focus instead on the latter with the goal to continue enabling the benefits of these services while respecting user's privacy.

2.3 Threat Model

Zeph aims to protect the confidentiality of data streams from an untrusted cloud platform and any other unauthorized entities and allows data owners to define privacy policies that enable the server to perform a well-defined set of privacy-preserving data transformations. We assume an *honest-but-curious* [61] adversary, i.e., the cloud platform follows the protocol correctly but will analyze all observed data to gain as much information as possible. Furthermore, we consider that the adversary might control some owners, including data producers. We assume the existence of a public-key infrastructure (PKI) for authentication of owners in Zeph. In addition, we assume that at most a fraction α of the parties involved in any aggregation transformation are colluding. Note that this can also include the server. The choice of α depends on the deployment scenario. In (§3.3) we show how the choice of α affects performance. For our evaluation, we use a pessimistic value of $\alpha = 0.5$, but real-world deployments might use significantly lower values. In this setting, Zeph provides the following guarantees:

Confidentiality. In Zeph, data is encrypted at the data producer with a semantically secure encryption scheme. The raw data is never leaked, even if the cloud platform were to be compromised since only the data owner and producer hold the keys used for encryption and decryption.

Policy Complying Transformations. A data owner can optionally define privacy policies for their data streams, allowing the server to integrate the streams into a privacy-preserving data transformation. An adversary learns nothing about the streams except what can be learned from the output of the transformation f (i.e., definition of $\hat{f}$*-privacy* [35]) with some modest leakage function due to encodings (§3). Only transformations that comply with the supplied privacy policy can be executed and only those from authorized parties. Any adversary controlling the server and at most a fraction α of owners is unable to violate the privacy policies of other owners. Zeph currently does not defend against side-channel attacks such as transformation or access pattern analysis.

3 Encryption for Privacy Transformations

In this section, we describe our approach to enable privacy transformations in end-to-end encrypted systems. Our design must serve privacy-compliant transformed views of data without affecting the data flow of an end- to-end encrypted stream-processing system. To meet this goal our design logically decouples privacy transformations and policy enforcement from the generation and storage of data. Data producers remain oblivious to the transformations and do not need to specifically encrypt data towards a fixed privacy policy. The modifications needed for privacy transformation are instead executed outside the data plane (i.e., conventional data flow), working exclusively on encryption keys and producing what we call cryptographic *transformation tokens*. These tokens are, in essence, the necessary cryptographic keying material that enables the respective transformation on encrypted data. Outputs of privacy transformations over encrypted data at the server-side are then released by combining the encrypted data with corresponding cryptographic transformation tokens. Introducing a logical separation between the data plane and privacy plane allows for heterogeneous policies atop the same data and leaves the conventional data flow unaffected. Before we discuss our encryption approach in detail, we present the transformation functions that our system supports.

Privacy Transformation Functions. Zeph supports three core functions over encrypted data, which serve as the building blocks to implement the desired privacy transformations discussed in §2. We show how we use these functions to implement complex privacy policies in real-world applications in §6. The three functions are: (i) $Aggr_S$, which enables ciphertext aggregation within the same data streams. Developers can use this function to implement privacy transformation that limit what can be inferred from data – namely to support generalization techniques that can reduce both the temporal and spatial resolution of data and masking techniques that can hide sensitive parts of data. (ii) $Aggr_M$, which enables ciphertext aggregation across streams from different users, and (iii) $Aggr_{DP}$, which supports noise addition to streams aggregated across multiple users. A privacy transformation F in Zeph can be any chain of these functions. From these, a broad set of transformations can be realized by combining them with user-side encodings [29, 35, 66]. For example, consider an end-to-end encrypted running tracking app that wants to display the most popular running routes to its users. For simplicity, we will assume that all users opt for the same privacy policy allowing use of the data only when rigorous privacy can be guaranteed. At the client application, locations in a given area are encoded into buckets defined by a GPS grid. The client encrypts and sends a route by annotating the respective elements in the bucket vector. The server-side transformation first performs an aggregation over the encrypted events in the desired time-window in each user's stream ($Aggr_S$). Then, it aggregates the encrypted outputs across all runners ($Aggr_M$). Finally, the transformation tokens (i.e., noisy decryption keys), generated in the privacy plane, allow to decrypt the output while adding noise at the same time ($Aggr_{DP}$). By releasing tokens containing the keys for only part of the bucket vector or multiple buckets together, the privacy plane can mask sensitive grid cells (e.g., cells within a restricted area) or reduce the spatial resolution (e.g., increase grid cell size for rural areas with few runners).

3.1 Decoupling Encryption from Privacy Transformations

Existing encrypted data processing systems utilize homomorphic encryption schemes to enable server-side computation on encrypted data [29, 60, 61, 73]. To meet stringent performance requirements, systems typically combine efficient partially homomorphic encryption schemes [25, 29, 35, 48, 60] with specialized client-side encodings to support a wider set of queries. Therefore, such systems generally support in-stream aggregation inherently, i.e., they can evaluate $Aggr_S$.

However, homomorphic evaluation alone is insufficient to support aggregations across streams ($Aggr_M$) that originate from different data owners. Supporting functions like $Aggr_M$ or $Aggr_{DP}$ in the client-server setting is typically achieved via multi-party computation protocols that are optimized for aggregation operations [13, 35, 56, 65]. These protocols ensure that the user inputs remain private and only the result of the aggregation is revealed to the server. However, these protocols are either limited to specific functions (e.g., updating sketches) or require the data producers to take an active part in the computation. For example, using common MPC techniques, chaining an $Aggr_M$ transformation after an inner-stream aggregation $Aggr_S$ would require the data producers to evaluate $Aggr_S$ on their data locally before initiating the $Aggr_M$ transformation protocol. In contrast, we want to remove the need for – potentially resource limited – data producers to take part in or even be aware of privacy transformations.

To decouple encryption from privacy transformations, we draw ideas from Homomorphic Secret Sharing (HSS) [27,28] literature. In essence, HSS allows computing a function F on secret shared messages by combining the outputs of a function $\hat{F}$ applied on the individual secret shares. HHS could be used to split stream events into two shares: one for the data plane (server) and one for the privacy plane. The privacy plane could authorize a transformation F by computing the same function $\hat{F}$ as the server on their local input shares, and releasing the output. Here, $\hat{F}$ supports all of the required core functions, as secret shares can also be aggregated across different data owners. Existing HSS constructions frequently support a much larger selection of functions, but as a result do not offer the performance required in our setting. However, Burkhalter et al. recently proposed a symmetric homomorphic encryption scheme [29] designed specifically for streams. Their scheme resembles a homomorphic secret sharing scheme and offers the required aggregation functions while introducing significantly lower overheads than traditional HSS approaches. The scheme efficiently derives a unique sub-key for each message from a master secret. Encryption is done via modular addition of the key and the message. Therefore, their encryption scheme can, in essence, be seen as an additive sharing of the message where the shares are the (message-specific) sub-key and the encrypted message. This scheme is a suitable choice for our design, as it offers both the required efficiency and the flexibility necessary to decouple the data- and privacy planes. The privacy plane and data producers need to only agree on the master secret and key derivation function. Then, the privacy plane can authorize a transformation F by deriving the involved sub-keys and executing $\hat{F}$ on them, which results in a *transformation token*. This token allows the server to compute and reveal the output of F by performing $\hat{F}$ on the ciphertexts and combining the result with the *transformation token*. If the transformation F spans multiple trust domains, i.e., the privacy plane consists of multiple privacy controllers, the privacy controllers can run an MPC protocol to compute the final transformation token. Note that this does not require the data producers to participate or even be online.

3.2 Transformation Tokens

We now describe the process of creating the *transformation tokens* that authorize the release of privacy transformation results. We start with a description of a simplified version of Zeph that assumes a single privacy controller and extend our description to consider multiple privacy controllers in §3.3. Let a data stream be a continuous stream of events $\{e_0, e_1, ..., e_i, e_{i+1}, ...\}$ for events $e_i := (t_i, m_i)$ consisting of a message and a timestamp. Each message m_i is an integer modulo M and is annotated with a discrete timestamp $t_i \in I$. We assume events are ordered by their timestamps and created in-order. We now explain the homomorphic encryption scheme that data producers use to encrypt such a stream.

Symmetric Homomorphic Stream Encryption. We use the symmetric homomorphic stream encryption scheme as described in [29]. In the setup phase, a master secret k is generated, the group size M is defined (e.g., 2^{64}), and a keyed pseudo-random function (PRF) $F_k : I \rightarrow [0, M-1]$ that outputs a fresh pseudo-random key k_i for timestamp t_i is selected. To encrypt an event e_i, the data producer uses the event timestamp from the last encrypted message t_{i-1} and computes:

$$SEnc(k, t_{i-1}, e_i) = (t_i, t_{i-1}, m_i + k_i - k_{i-1} \mod M) \quad (1)$$

where $k_i = F_k(t_i)$, $k_{i-1} = F_k(t_{i-1})$. This scheme is additively homomorphic: ciphertexts can be aggregated via modular additions. The client can efficiently decrypt an aggregation over the time-window $[t_i, t_j]$ by deriving only the two outer keys $k_i = F_k(t_i)$ and $k_j = F_k(t_j)$.

Authorizing Transformations. We can observe that this encryption scheme hides the inputs from the server and allows the server to perform aggregations among the streams without accessing the plaintext data. The intuition in Zeph is that the encryption scheme essentially splits a message m_i into two additive secret shares: the key $-k_i + k_{i-1}$ and the ciphertext c_i, with $m_i = c_i + (-k_i + k_{i-1}) \mod M$. Therefore, any transformation F consisting of the three core aggregate operations can be performed independently on both the ciphertexts and the keys using modular additions. The latter produces a *transformation token* τ_F, which the server can use to reveal the output o_F of a transformation F by computing $o_F + \tau_F \mod M$. Hence, a privacy controller that is in possession of the master keys of the streams can authorize a transformation F by deriving the necessary keys and performing the transformation on top of them to produce a matching *transformation token* τ_F. In the following, we assume that all additions are performed modulo the parameter M.

Single-Stream Transformation Tokens. We now describe how a privacy controller can create transformation tokens for $Aggr_S$ transformations, e.g., only reveal the approximate locations aggregated over a month. In a window aggregation, the server adds values within the specified time window t_i to t_{i+w} , where w is the window size. As long as data producers submit a value on each window border, the resulting ciphertext of the window aggregation on the server shares has the form $c_w = m_{aggr} + k_{i+w} - k_{i-1}$. The privacy controller can compute the transformation token for this window $\tau = -k_{i+w} + k_{i-1}$ by deriving only the two outer keys $k_i = F_k(t_i)$ and $k_j = F_k(t_j)$ as the inner-keys cancel each other out [29, 60]. With this token, the server can decrypt the window aggregation if and only if the correct windows were aggregated, as the keys directly encode the window range. For aggregations within events, the privacy controller uses modular addition to add the respective sub-keys to create the transformation token.

Multi-Stream Transformation Tokens. Multi-stream transformation tokens reveal the output of $Aggr_M$ transformations, which aggregate data over multiple streams, e.g., only reveal the approximate location aggregated among multiple users. In multi-stream aggregation, the server sums a fixed window t_i to t_{i+w} across different streams. Let S be the set of streams in the aggregation. For each stream $j \in S$ we have a window aggregated share $c_w^{(j)} = m_{aggr}^{(j)} + k_{aggr}^{(j)}$ where $k_{aggr}^{(j)} = k_{i+w}^{(j)} - k_{i-1}^{(j)}$. The aggregation over all streams in S results in the sum of all window aggregates and the sum of all window share keys. Hence, a privacy controller can compute the transformation token by aggregating the window keys $\tau^{(j)} = -\sum_{j \in S} k_{aggr}^{(j)}$.

Differentially-Private Transformations. Differential Privacy [36] provides formal bounds on the leakage of an individual's private information in aggregate statistics. The most common technique to achieve a differentially private release of information is to add carefully calibrated noise. Zeph supports *noisy* transformations (i.e., $Aggr_{DP}$) on multi-stream window transformations, but could be extended to single-stream. The privacy controllers add carefully calibrated noise to the keys (i.e., submit noisy keys): $\tilde{\tau}_j = \tau_j + \eta_j$ where η_j is the noise distribution. Zeph therefore supports all additive noise mechanisms from the Differential Privacy literature [37] with noise drawn from a divisible distribution. However, mechanisms like the *Sparse Vector Technique* [38] that require access to the underlying data cannot be applied this way. In previous work, noise is added to plaintexts prior to encryption rather than to decryption keys, as done in Zeph. Cryptographically, our approach is equivalent. Practically, however, it has significant advantages since the encrypted data remains unchanged. As a result, the same data is reusable for encrypted storage and to facilitate one or multiple differentially private privacy transformations.

3.3 Transformations Across Different Trust Domains

Until now we assumed a single privacy controller that is in control of all streams. We now discuss how Zeph enables multiple privacy controllers that are each responsible for a distinct subset of streams. Data owners should trust their privacy controller since it could reveal their private information by colluding with the server. However, different data owners might not want to trust the same controller. In such a multi-controller setting, the server needs to interact with all privacy controllers involved in a transformation. Hence, when aggregating across streams the server needs to request a transformation token from each privacy controller. In a naïve approach, the privacy controllers might simply create the transformation tokens as in the single-controller setting and send a combined token for the aggregation of streams under their control. However, this approach leaks the result over each controller's subset of data to the server. Instead, we need the individual tokens to reveal no additional information while still enabling correct decryption of the total aggregation. We solve this using secure aggregation [13, 26], a specialized secure MPC protocol in the client-server setting. We require a secure aggregation protocol that is *(i)* lightweight in terms of computation for privacy controllers and *(ii)* can be efficiently executed multiple times with similar participants. Based on these requirements, Zeph builds on the secure aggregation protocol from Ács et al. [13] to create transformation tokens over multiple parties. The protocol goes hand in hand with the design of the transformation tokens, as it also relies on additive masking. In the following, we outline the protocol and then describe our optimizations.

Core Protocol. We consider a set $\mathcal{P}$ consisting of N privacy controllers and a server that aggregates the inputs. Each privacy controller $p \in \mathcal{P}$ owns a token τ_p that is constructed by aggregating the tokens for the corresponding $Aggr_S$ transformation for each stream under their control. The goal of the protocol is to compute $\tau = -\sum_{p \in \mathcal{P}} \tau_p$ without revealing the individual inputs τ_p to the server or the other privacy controllers. Each privacy controller masks its input τ_p with a nonce k_p, i.e., it computes $Enc(\tau_p, k_p, M)$. The nonces are constructed such that the sum over all nonces results in $\sum_{p \in \mathcal{P}} k_p = 0$. As a consequence, the sum over all encrypted inputs results in the sum of inputs:

$$\sum_{p \in \mathcal{P}} \tau_p + \sum_{p \in \mathcal{P}} k_p \mod M = \sum_{p \in \mathcal{P}} \tau_p \mod M \tag{2}$$

To construct the canceling nonce, each privacy controller establishes $N-1$ pairwise shared secrets $k'_{p,q}$ with all other privacy controllers which are aggregated to form the nonce k_p In particular, if $p > q$, then the controller p adds $-k'_{p,q}$ else $k'_{p,q}$.

$$k_p = \sum_{p>q} -k'_{p,q} + \sum_{p<q} k'_{p,q} \mod M \tag{3}$$

Hence, the pairwise secrets cancel each other out when the masks are combined in the aggregation. For conciseness, we refer to Ács et al. [13] for a description of dropout handling.

Constructing Canceling Nonces. In Zeph, the secure aggregation protocol is run repeatedly for multiple rounds. Thus, privacy controllers require an efficient method to establish many pairwise shared secrets. The standard protocol achieves this with a setup phase where the parties create pairwise shared secrets $k_{p,q}$ using a Diffie-Hellman key exchange. These pairwise secrets then serve as seeds (or keys) for a PRF to establish nonces for each round r: $k^r_{p,q} = PRF(k_{p,q}, r)$.

Even though PRF computations are significantly more efficient than a Diffie-Hellman key exchange, this protocol still requires each privacy controller to evaluate $O(N)$ PRF's and additions to create the blinding nonce k_p for a single transformation token, which can be expensive for large N.

To improve this theoretical overhead, we view the complexity of creating a shared blinding nonce as a graph $G = (V, E)$ with the set of vertices V representing the involved parties ($|V| = N$), and the set of edges E denoting the pairwise canceling masks $k'_{p,q}$. In the standard form described above, the graph G forms a Clique because every privacy controller includes a pairwise mask $k'_{p,q}$ with every other privacy controller. To reduce the number of PRF evaluations in the online phase for a privacy controller (i.e., reduce the number of edges in the graph G), we propose an optimization that leverages the fact that the protocol is repeating over a longer period of time with similar participants, i.e., the long-running nature of streaming queries.

Online Phase Optimization. The high-level idea is that a privacy controller's nonce includes only a small random subset of the pairwise-secrets in each round. In graph terms, this corresponds to a small expected degree of each vertex. As long as the graph remains connected[1], confidentiality is guaranteed. We divide the online phase into epochs consisting of t rounds. At the beginning of each epoch, we use $N-1$ evaluations of the PRF to bootstrap the secure aggregation graphs for the epoch. A privacy controller assigns each edge to a small number of rounds, based on the output of a PRF evaluated on the shared secrets. More specifically, we divide the output of $PRF(k_{p,q}, r)$, where r is a public epoch-identifier, into b-bit segments. Each segment assigns the edge $e_{p,q}$ to one of 2^b graphs using the number encoded in the b-bit segment.

Assuming a 128-bit output size of a PRF (e.g., AES), an epoch consists of $t = \lfloor 128/b \rfloor \cdot 2^b$ rounds. In comparison, the protocol of Ács et al. [13] uses the same $N-1$ PRF evaluations to create only a single secure aggregation graph (i.e., epoch size of one). Ideally, we want to create as many graphs as possible, i.e., select a large b, since with increasing b, an epoch consists of more rounds. However, with increasing b, each of the associated graphs has fewer edges, which increases the risk of a graph being disconnected. In the formal supplementary material, we show how to select b so that the probability of any honest subset of nodes being isolated in any of the t generated graphs is bounded by δ, assuming a fraction of at most α parties collude.

For example, for 10k privacy controllers, assuming that up to half are colluding ($\alpha = 0.5$), and bounding the failure probability by $\delta = 1 \times 10^{-9}$, allows for $b = 7$, which results in an epoch consisting of 2304 rounds where each vertex has a expected degree of 78. As a consequence, our optimization requires 190k PRF evaluations and 180k additions for constructing all 2304 blinding nonces of an epoch. In comparison, the basic protocol requires 23 million PRF evaluations and additions while the protocol from Ács et al. [13] requires 23.2 million PRF evaluations and 180k additions[2].

[1] More specifically, the subgraph of *honest* nodes must remain connected.

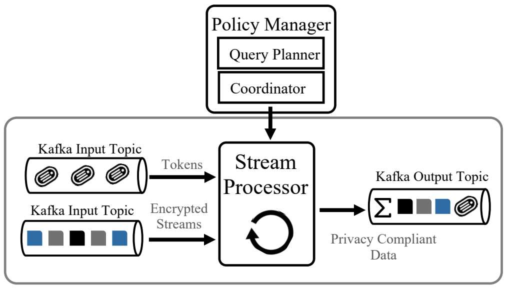


Figure 2: Architecture of the Zeph microservice that runs on top of a streaming pipeline.

4 Zeph System Design

Zeph is a privacy platform that cryptographically enforces user-defined privacy preferences in streaming platforms by sharing only transformed privacy-compliant views of the underlying encrypted data. So far, we have described the cryptographic building blocks that enable privacy transformations in Zeph. Here, we describe how we overcome the system challenges that need to be addressed to allow practical deployment. Zeph augments existing stream processing pipelines, similar to existing frameworks operating on data in-the-clear [57]: *(i)* On data producers, Zeph adds a proxy module for encoding and encryption. *(ii)* On the server, Zeph adds a microservice running in the existing stream processing platform. This microservice transforms the incoming encrypted streams into privacy-compliant output streams (Figure 2), which can then be consumed by existing stream processing queries for arbitrary post-processing.

4.1 User API and Privacy Policies

Before introducing the Zeph components in detail, we discuss aspects related to users' interaction with Zeph.

Privacy Preferences. Zeph provides the capabilities for users to set their privacy preferences and the means to cryptographically enforce various privacy policies in a unified system. In this paper, we do not consider the question of what this set of privacy preferences should be. Nevertheless, we suggest and implement a sensible set of options. In the current design, data owners can set their preferences as follows: *(i)* do not share my data, *(ii)* share my data without restrictions, *(iii)* share my data only when aggregated with other users, and *(iv)* share only generalized views of my data and/or mask sensitive data, i.e., share but limit inference of sensitive information from my data. The realization of these preferences in practice is application- and data-dependent (i.e., generalization and

[2] All results assume that τ_p is at most 128-bit long and hence a single evaluation of AES is sufficient for encryption.

data minimization techniques can differ depending on the data type, e.g., image, location, heart rate).

Data Stream Schema. Zeph's schema language builds on the *Avro* [17] schema language (Figure 3). Using our extended schema language, developers can translate users' privacy options to configurations, encodings, and transformations for their application. In addition, the schema contains meta-information about the stream and the contents of events within a stream. This enables seamless integration into existing streaming services employing schema registries to store structural information about the events flowing through the system. A Zeph stream schema contains: *(i) Metadata attributes* describing static fields that remain constant for an extended period of time and are public information. Zeph's microservice uses these metadata tags to group and filter streams for transformations over different populations (§4.3). *(ii) Stream attributes* describe the private contents of an event message and are annotated with all possible supported queries. These explicit annotations are required to derive the necessary encodings to execute queries using the three core functions (§3). *(iii)* The *privacy options* for stream attributes. A privacy option describes the set of transformations that the service can perform to reveal an output. The options *stream-aggregate*, *aggregate*, and *dp-aggregate* directly correspond to the three core functions defined in §3. In addition, *private* does not allow any transformations on the stream while *public* allows access to the raw data. For each transformation set, one can add further constraints, e.g., defining a minimum population size, specifying a lower temporal resolution by aggregating over time or providing a privacy budget for the transformation.

Annotating Streams. The Zeph schema for a particular application can be accessed by all privacy controllers. A user's privacy selection in the application triggers the responsible privacy controller to create a matching stream annotation and share it with the server. A stream annotation contains the selected privacy option along with values of the metadata attributes and additional information about the stream (Figure 3). This information later allows Zeph's server to identify suitable streams to include in privacy transformations. Stream annotations contain an identifier of the data owner (e.g., the hash of their public key) that maps to the data owner's public key in the PKI.

4.2 Writing Encrypted Data Streams

Data producers submit streams of events to the pipeline where each event conforms to a data schema in the schema registry, as in standard streaming pipelines. However, Zeph augments them with a proxy module to handle encoding and encryption.

Setup. To initialize a new data stream matching a Zeph schema, the data producer generates a master secret and shares both the schema and the master secret with the associated privacy controller. After the initial setup phase, the data producer can start sending encrypted data to the server without any further coordination with the privacy controller.

```
name: MedicalSensor
metadataAttributes:
  - name: ageGroup
    type: [enum, optional]
    symbols: [young, middle-
              aged, senior]
  - name: region
    type: string
streamAttributes:
  - name: heart-rate
    type: integer
    aggregations: [var]
  - name: hrv
    type: integer
streamPolicyOptions:
  - name: aggr
    option: aggregate
    clients: [medium, large]
    window: [1hr]
  - name: priv
    option: private
```

```
id: 235632224234
ownerID: 2474b75564b
serviceID: app.com
validFrom: 2020-04-20
validTo: 2021-04-20
stream:
  type: MedicalSensor
  metadataAttributes:
    ageGroup: middle-aged
    region : California
  privacyPolicy:
  - heartrate:
     option: aggr
     clients: medium
     window: 1hr
  - hrv:
     option: priv
```

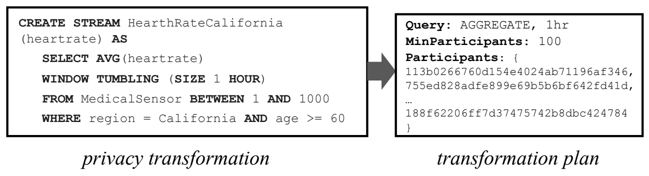
Figure 3: An example privacy policy schema of a medical sensor (*left*) and a stream annotation for this schema (*right*). (YAML format for display)

Encrypting Data Streams. The proxy module encrypts each record with the symmetric homomorphic encryption scheme (§3), using the master secret from the setup phase. Moreover, in regular intervals (e.g., every minute), the data producer sends an encrypted neutral value that does not affect the result of computations but is required for $Aggr_S$ transformations across time. In essence, these messages allow the privacy controller to derive a transformation token without observing the data (§3). An additional benefit of these messages is that they allow the Zeph microservice to detect and handle dropped data producers (e.g. due to network interruptions).

4.3 Matching Queries with Privacy Policies

Zeph's microservice contains a policy manager component that manages active streams, privacy controllers and transformations in the streaming pipeline. It provides a query interface for launching new transformations and matches queries with available streams by considering their chosen privacy options. Privacy transformations are chains of the core operations $Aggr_S$, $Aggr_M$, and $Aggr_{DP}$ and are executed as stream processing queries running continuously on a set of encrypted streams. The challenge here is that Zeph needs to ensure that the query complies with all the stream's selected policy options to receive the required transformation tokens from the privacy controllers. To address this, Zeph's policy manager includes a query planner that leverages the fact that privacy transformation queries follow the same structure, which we discuss in more detail below.

Query Language. The query language of Zeph builds on *ksql* [47], an SQL-like query language for expressing continuous queries on data streams. Any authorized service can express privacy transformations that follow the pattern explained above. Figure 4 shows an example query, which



```
CREATE STREAM HearthRateCalifornia
(heartrate) AS
   SELECT AVG(heartrate)
   WINDOW TUMBLING (SIZE 1 HOUR)
   FROM MedicalSensor BETWEEN 1 AND 1000
   WHERE region = California AND age >= 60
```

```
Query: AGGREGATE, 1hr
MinParticipants: 100
Participants: {
113b0266760d154e4024ab71196af346,
755ed828adfe899e69b5b6bf642fd41d,
…
188f62206ff7d37475742b8dbc424784
}
```

Figure 4: The query planner converts privacy transformations into transformation plans with complying data streams.

creates a transformed stream for the hourly average heart rate of seniors in California, including at most 1k streams.

Query Planner. The query planner executes queries from authorized services in three steps: *(i)* streams are filtered by their metadata attributes (e.g., all medical sensor streams in California). *(ii)* an $Aggr_S$ operation using a time-window is performed on certain attributes of each selected stream (e.g., average heart rate over 1 hour). The query planner checks for each selected stream that the transformation complies with the annotated privacy options for the attributes used, else the stream is excluded. *(iii)* If more than one stream is selected, an $Aggr_M$ or $Aggr_{DP}$ operation is performed on the results of the previous step. The query planner checks for each remaining stream that the transformation complies with the privacy option and checks that the population constraints are met (e.g., minimum population size), or otherwise excludes the stream. These compliance checks are necessary, as privacy controllers would not provide the required tokens for a stream where the privacy options do not allow the query. To prevent an attacker from combining outputs of different transformations to violate privacy policies, any stream attribute can be matched to only one query, and is removed from the set of queriable streams for this attribute as long as the stream is part of the running transformation. After processing the query, the query planner outputs a *transformation plan* that encodes the list of streams in the transformation, information on fault tolerance (i.e., how many participants are allowed to drop), and the operations that have to be performed (Figure 4).

4.4 Coordinating Privacy Transformations

Once the query planner outputs a transformation plan, Zeph executes the privacy transformation in the streaming pipeline. Zeph provides a customized stream processor that handles the required coordination between the transformation job running in the streaming pipeline and the privacy controllers. In addition to handling data, it consumes event messages (i.e., tokens) from privacy controllers and writes events about the state of the transformation back to the privacy controllers.

Transformation Setup. Zeph introduces a coordinator component that initiates the setup based on transformation plans provided by the query planner. In order to initialize a new job, the coordinator first determines the involved privacy controllers and distributes the transformation plan to them. This step enables the privacy controllers to verify the compliance of the transformation against the user-defined privacy option. The verification involves checking the privacy policy based on the included attributes, window size, aggregation size, and/or noise configurations. If the transformation plan includes multiple data owners, each privacy controller needs to verify the identities involved in the transformation plan by fetching their certificates from the PKI. Afterwards, each privacy controller initiates the setup phase of the secure aggregation protocol (§3.3) among the involved privacy controllers. Once all privacy controllers agree, the coordinator initiates the transformation job in the streaming pipeline.

Transformation Execution. The stream processor continuously aggregates incoming encrypted events into windows and applies the transformation tokens received from the privacy controllers. Zeph runs an interactive protocol with the privacy controllers once per window, to robustly adjust to failures of both data producers and privacy controllers. At the end of each window, the stream requests a heartbeat from all privacy controllers in the transformation. Note that data producer dropouts can be detected by the absence of their events. After a specified timeout, the data transformer computes the intersection of available data producers and privacy controllers and broadcasts a membership delta in comparison to the previous window to all involved privacy controllers.

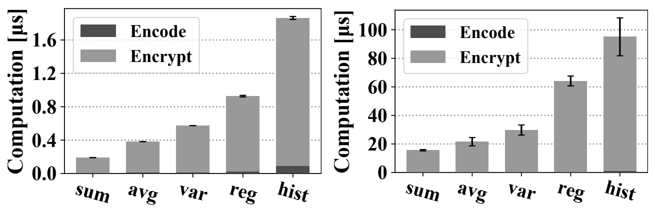
After receiving an update, the privacy controllers verify that the transformation still complies with the selected privacy options and update the tokens they send to match the new transformation. Once all transformation tokens have arrived, Zeph can complete the transformation and output the result.

5 Implementation

Our prototype of Zeph is implemented on top of Apache *Kafka* [18], consisting of roughly 4500 SLOC and 5500 SLOC for benchmarks. We provide a data producer proxy library written in Java that relies on the Bouncy Castle library [52] for cryptographic operations, and *Avro* [17] for serialization. The privacy controller is implemented in Java but, via the Java native interface (JNI), calls native code in Rust for the secure aggregation protocol. For the PRF, we rely on CPU-based AES-NI using the `AES` Rust crate [67], and for the ECDH key exchanges we use the `secp256r1` elliptic curve from Bouncy Castle [52]. We use the Apache *Kafka* Streams framework [19] to implement the stream processor for the privacy transformations. We emulate the policy manager with a configurable Ansible [16] playbook.

6 Evaluation

Meeting the performance requirements of data stream processing is a key goal of Zeph's design. Therefore our experimental evaluation is designed to validate this and more concretely answer the following two questions: *(i)* what is the cost of enforcing privacy policies with encryption in Zeph?, and *(ii)* can Zeph provide the means to support practical privacy for various applications in an acceptable overhead?



(a) EC2 instance (b) Raspberry Pi

Figure 5: The computation cost at the data producer for encryption and different stream encodings. The encoding for hist has ten buckets.

6.1 Experimental Setup

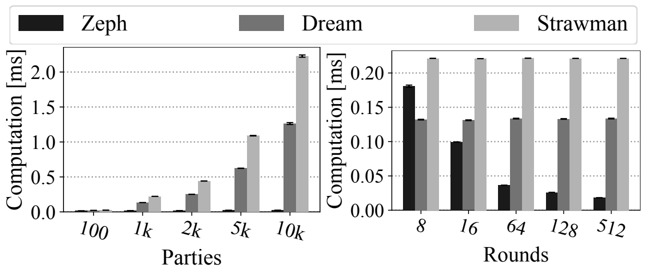
The experimental evaluation consists of two parts. First, we quantify the overhead of Zeph components with microbenchmarks. We start by quantifying the performance of our proposed secure aggregation optimization compared to *Strawman* with no optimization (§3.3) and the optimized protocol by Ács et al. *Dream* [13]. The second part of the evaluation aims to quantify the end-to-end performance of Zeph as we integrate it into three applications with different privacy options. Moreover, we show how various data-dependent privacy logic can be realized in Zeph. In these experiments, we consider a setting where each data producer has a separate privacy controller; this represents the worst-case scenario – the number of privacy controller involved in the MPC protocol is equal to the number of data streams.

Compute. We run the microbenchmarks on Amazon EC2 machines (m5.xlarge, 4 vCPU, 16 GiB, Ubuntu Server 18.04 LTS). Additionally, we run the data producer microbenchmarks on a Raspberry Pi 3 B to analyze the performance on more resource-constrained edge devices. For the end-to-end evaluation, we employ Amazon MSK [15], which provides a *Kafka* cluster as a fully managed service. The *Kafka* cluster contains two broker nodes (m5.xlarge) spread over two availability zones in Frankfurt. The stream processor application spreads over a set of two EC2 machines (m5.2xlarge) using *Kafka* streams. Data producers and privacy controllers are grouped into partitions of up to 100 entities. A single producer- or controller-partition runs on one EC2 machine (m5.large). We run the partitions in three different regions London, Paris, and Stockholm, to simulate federation.

Configuration. In the microbenchmarks, Zeph uses an event with a single stream attribute x encoded as $\vec{x} = [x, x^2, 1]$ while for the end-to-end setup, we use application-specific encodings. Throughout the evaluation, Zeph's optimized secure aggregation assumes that up to half the participants are colluding (i.e., $\alpha = 0.5$), and that the failure probability is below $\delta = 1e-7$. For the end-to-end evaluation the data producer uses a Poisson process with a mean of 0.5 to time inserts (i.e., an average of 2 inserts/s).

Privacy Controllers	100	1k	10k	100k
Bandwidth	9.0 KB	91 KB	910 KB	9.1 MB
Bandwidth Total	901 KB	91 MB	9.1 GB	910 GB
Shared Keys	3.2 KB	32 KB	0.3 MB	3.2 MB
ECDH	25 ms	249 ms	2.5 sec	25 sec
ECDH Total	2.5 sec	4 min	7 h	693 h

Table 1: The computation and bandwidth costs for the privacy controller in the *setup phase* of a multi-stream transformation. The total amount consists of the sum of all cost involved over all privacy controllers of the transformation, versus the costs for a single privacy controller. The Elliptic-curve Diffie–Hellman (ECDH) key exchanges dominate the computation and bandwidth costs.



(a) Average per round (b) Varied rounds for 1k parties

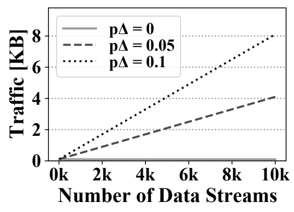
Figure 6: Computation costs for privacy controllers in the *privacy transformation phase* to execute multi-stream queries. A round corresponds to a transformation of a single time window.

6.2 Data Producer

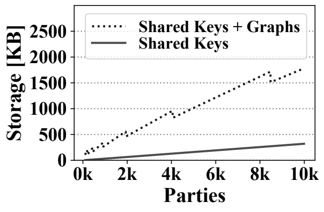
We now discuss Zeph's overhead at the data producers.

Computation. The encryption cost for a single record with *SEnc* is low with 0.19μs on EC2 and 16μs on a Raspberry Pi, as the encryption scheme only relies on symmetric primitives (i.e., efficient AES). Figure 5 shows the encryption latency for different encodings. A data producer can maintain a throughput in the range of 5.3m to 524k records per second (rps) depending on the encoding. Even on a Raspberry Pi, the computation cost is moderate, and a throughput of 7.7k to 76.6k rps can be observed. To accommodate for window borders, the data producer has to additionally submit a ciphertext per-window, which increases the cost at a fixed rate.

Bandwidth. Compared to plaintext, Zeph's aggregation-based encodings and timestamp introduce a ciphertext expansion which manifests itself in increased bandwidth requirements. The expansion varies from 24 bytes (1.5x) with one encoding to 96 bytes (6x) with 10 encodings, i.e., grows by 8 bytes per encoding. Besides this, the window border ciphertexts increase bandwidth with an additional constant factor.



(a) Bandwidth for transformation phase depending on delta probability p_Δ.



(b) Memory costs for a privacy controller during privacy transformation phase

Figure 7: Bandwidth and memory costs for privacy controllers in the *privacy transformation phase*.

6.3 Privacy Controllers

The cost of the privacy controller depends on the executed transformations on the service side. Single-stream window transformations are efficient both in computation and bandwidth because no MPC is involved. The privacy controller computes the transformation tokens on a per-window basis from the master secret with a computation cost of around 0.2μs and bandwidth cost of 8 bytes per token.

For the multi-stream case, the privacy controller engages in the secure aggregation protocol (§3.3). We quantify the overhead by running the secure aggregation protocol for different numbers of privacy controllers and compare it against the strawman approach. As a first step, all these protocols require a *setup phase* to establish pairwise shared secrets with all involved parties. Afterward, the *privacy transformation phase* starts, during which the privacy controllers create the required transformation tokens at the end of each window.

Setup Phase. The *setup phase* overhead increases quadratically with the number of privacy controller, i.e., $O(N^2)$, however, we assume that N stays below 10k for most transformations in Zeph. This assumptions stems from the fact that privacy transformations are only pre-processing steps for advanced queries, i.e., a service can split a query over larger populations into multiple privacy tranformations. Table 1 shows a quadratic increase of the bandwidth and computation costs for running the setup phase with the ECDH key exchanges. However, the overall amount is reasonable even for 10k participants, setting with 910 KB bandwidth and 2.5 sec computation cost per privacy controller. Note that the setup phase has to be performed only once a new transformation query is created. In terms of memory, the privacy controllers need to store their private-key (i.e., 150 bytes) and the established shared secrets of the current privacy transformation. Each shared key requires 32 bytes, e.g., 3.2MB for 100k shared keys.

Privacy Transformation Phase (Optimization). Zeph optimizes the cost of the secure aggregation protocol per round by computing the shares in random sub-groups (§3.3). In the initial phase, the controllers have to invest more resources to compute the random subgroup for the upcoming rounds (i.e., epoch). After a few rounds, the additional work performed at the beginning of an epoch is amortized and, therefore, the overall cost of the computation reduces significantly in the long run, as depicted in Figure 6. With 1k participants, the computation costs for the first window is 1 ms for a privacy controller, while in the following windows Zeph reduces the computation cost by 2.6x. Already for 8 and 16 windows for 10k and 1k participants, respectively, the Zeph optimization is more efficient on average and the amortized performance improvement increase linearly with the number of rounds the transformation runs, as shown in Figure 6a. For 10k privacy controllers, an individual participant requires less than 2 MB to store the shared keys and the secure aggregation graphs of the epoch (Figure 7b). As a result, even though the overhead increases in the number of privacy controllers, the total memory remains acceptable. In case memory is scarce (e.g., because a privacy controller is in charge of large number of data streams), a privacy controller can resort to storing a fraction of the secure aggregation graphs and recalculate the next batch of graphs at the required time.

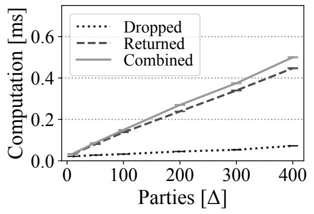


Figure 8: Computation cost for a privacy controller to adapt to Δ dropping or joining parties. In the combined case, Δ members dropped and Δ other members returned.

Drop-Outs. In Zeph, privacy controllers can dynamically join or leave in the transformation phase, which increases both the computational cost and the required bandwidth due to the additional communication, as depicted in Figure 8. The computation and bandwidth costs for adapting the transformation token are linear in the number of returning participants as well as dropping participants. These costs are modest, even for the extreme fraction of dropping and joining users (i.e., 400 each), the induced cost remains below 0.5 ms. In terms of bandwidth, a privacy controller observes less than 10KB bandwidth, even under the assumption of a 10% fluctuation of dropping participants (Figure 7a).

6.4 End-to-End Application Scenarios

This section evaluates the end-to-end overhead of Zeph and its effectiveness in supporting a variety of privacy policies relevant to real-world applications. We develop three applications with Zeph that represent different complexities of privacy transformations. We evaluate each application with 300 and 1200 active data producers, each producing two events per second with a window size of 10 seconds. Each data producer has its own privacy controller and we set $\alpha = 0.5$ as usual.

Fitness Application. We consider the Polar App [7] which

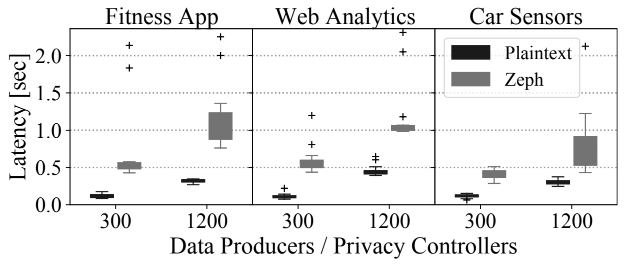


Figure 9: Computation cost for Plaintext (no encryption) and Zeph for different Applications. The latency measures the time after the grace period (5s) of a window is over until the result of the transformation is available.

collects data during users' sports activities. Recorded data includes heart-rate, altitude and weather information, among others. We consider a privacy policy that limits the resolutions of sensor data temporally and/or spatially. In our evaluation, we gather statistics about the average heart-rate of a population organized into per-altitude buckets with a maximum resolution of 5 meters. Each exercise event consists of 18 attributes that are encoded in 683 values in Zeph.

Web Analytics. We implement Zeph on a subset of statistics from the Matamo [6] web analytics platform for gathering website statistics such as page views, user flows, and click maps. Here we evaluate aggregation queries using a privacy policy that translates to only differentially private (i.e., noised) information aggregated over all users being made available to a third-party service. To enable this functionality in Zeph, we encode the 24 attributes into 956 values.

Car Predictive Maintenance. We consider a car metric data platform that contains a predictive maintenance service [1]. We consider a setting where users allow a third-party service to observe sensor readings only if they are out of the ordinary or differ too much from long-term aggregates across different cars. Therefore we compute both the long-term aggregates across many users and individual histograms for each user. The application records 23 different attributes from car sensors and encodes them into 169 values.

Performance. Figure 9 shows the observed stream transformation latencies for the different applications compared against plaintext. The latency overhead varies between 2x and 5x for the different applications. Note that Zeph always completes processing the current window well before the next one needs to be processed. With this, we show that Zeph is capable of performing real-time privacy transformations atop encrypted streams for a variety of application scenarios.

7 Related Work

Privacy Policy Enforcement. The most common approach to tackle the problem of enforcing privacy policies automatically in real-world data processing systems is to leverage information flow control (IFC) to check and constrain how information flows through the system [24, 39, 40, 55, 69, 76]. These systems feature different variations on how IFC rules can be expressed and who enforces these rules in application code. In contrast to Zeph, these approaches rely on a trusted service or trusted hardware and do not comply with the end-to-end encryption paradigm. Riverbed [76] is a practical IFC system that enforces user-defined privacy policies with information flow techniques by grouping users with similar policies into separately running containers (i.e., universes). Ancile [20] introduces a trusted data processing library that automatically enforces user-defined privacy preferences on passively generated data by only releasing policy complying transformations of data to applications. In a *multiverse database* [54, 68], global privacy policies are enforced by only exposing materialized views of the database to each user in an application. A multiverse database is fully trusted to enforce privacy policies correctly. Pyramid [51] exposes privacy-compliant transformed data to services for analytics. However, users cannot express their individual privacy preferences.

Private Aggregate Statistics. Zeph leverages techniques from existing private data-collection systems [13, 26, 50] to compute sums over private user data to facilitate privacy-preserving transformations on data. However, no existing system fulfills the requirement of supporting aggregations across data streams with heterogeneous user-defined privacy policies. In addition, they require data producers to actively participate in the aggregation protocols. Various protocols are based on homomorphic encryption [32, 65] or secret sharing [13, 26, 35] that enable an untrusted aggregator to collect encrypted inputs while ensuring that only the sum of the inputs can be decrypted. Prio [35] enables a service to collect private aggregate statistics from user data while allowing the server to check that inputs are not malformed, but it relies on non-colluding servers. Bonawitz et al. introduce a secure aggregation protocol to support large-scale private machine learning workloads [26], which has slightly different requirements than our application. Even when securely computed, plain aggregations can still reveal sensitive information. Private data-collection systems [33, 42] based on differential privacy [36, 38] therefore add carefully-crafted noise to the data inputs. Several systems [13, 56, 65, 70] combine differential privacy techniques with secure aggregation in a way that minimizes the amount of added noise. This line of work is orthogonal to Zeph, and some can be integrated with Zeph.

8 Conclusion

The practice of massive data collection is not likely to diminish anytime soon. Corporations across all sectors consider data as a valuable asset that has enormous value to their business. However, as we accumulate more and more sensitive data, protecting individuals' privacy is gaining critical urgency. Today's privacy landscape presents a unique set of challenges and opportunities that make this an auspicious time to reshape our data ecosystems for privacy. Adequately addressing privacy in the current complex computing landscape is an acute challenge and is vital to avoid the pitfalls of big data. The path for achieving this necessitates developing privacy tools that can easily be implemented in existing data pipelines. In this paper, we propose a new end-to-end design for privacy. A design that empowers users with more control with a user-centric model to privacy and that ensures strong data protection and compliance assurance with a cryptographic enforcement approach to privacy policies.

References

[1] Bosch Predictive Maintenance. Online: https://www.bosch-mobility-solutions.com/en/products-and-services/mobility-services/predictive-diagnostics/.

[2] California Consumer Privacy Act (CCPA). CCPA, Online: https://oag.ca.gov/privacy/ccpa.

[3] General Data Protection Regulation: GDPR. GDPR, Online: https://gdpr-info.eu/.

[4] Immuta Platform. Online: https://www.immuta.com/.

[5] Instagram Data Policy. Online: https://help.instagram.com/519522125107875. Accessed on 09-12-2020.

[6] Matomo Web Analytics. Online: https://matomo.org/.

[7] Polar Platform. Online: https://www.polar.com/accesslink-api/#detailed-sport-info-values-in-exercise-entity.

[8] Privatar Privacy Platform. Online: https://www.privitar.com/.

[9] Strava. Online: https://www.strava.com/.

[10] Twitter Privacy Policy. Online: https://twitter.com/en/privacy. Accessed on 09-12-2020.

[11] Gartner Says Just Four in 10 Privacy Executives Are Confident About Adapting to New Regulations. Gartner, Online: https://www.gartner.com/en/newsroom/press-releases/2019-04-23-gartner-says-just-four-in-10-privacy-executives-are-confident-about-adapting-to-new-regulations, April 2019.

[12] John M. Abowd. The U.S. Census Bureau Adopts Differential Privacy. In *ACM SIGKDD*, 2018.

[13] Gergely Ács and Claude Castelluccia. I Have a DREAM! (DiffeRentially privatE smArt Metering). In *International Workshop on Information Hiding*, pages 118–132. Springer, 2011.

[14] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. MillWheel: Fault-Tolerant Stream Processing at Internet Scale. *VLDB*, 6(11):1033–1044, 2013.

[15] Amazone MSK. Online: https://aws.amazon.com/de/msk/.

[16] Ansible. Online: https://www.ansible.com/.

[17] Apache Avro. Online: https://avro.apache.org/.

[18] Apache Kafka. Online: https://kafka.apache.org/.

[19] Apache Kafka Streams. Online: https://kafka.apache.org/documentation/streams/.

[20] Eugene Bagdasaryan, Griffin Berlstein, Jason Waterman, Eleanor Birrell, Nate Foster, Fred B. Schneider, and Deborah Estrin. Ancile: Enhancing Privacy for Ubiquitous Computing with Use-Based Privacy. In *ACM WPES*, page 111–124, 2019.

[21] E. Balsa, C. Troncoso, and C. Diaz. Ob-pws: Obfuscation-based private web search. In *2012 IEEE Symposium on Security and Privacy*, pages 491–505, 2012.

[22] Vinayshekhar Bannihatti Kumar, Roger Iyengar, Namita Nisal, Yuanyuan Feng, Hana Habib, Peter Story, Sushain Cherivirala, Margaret Hagan, Lorrie Cranor, Shomir Wilson, Florian Schaub, and Norman Sadeh. Finding a Choice in a Haystack: Automatic Extraction of Opt-Out Statements from Privacy Policy Text. In *WWW*, page 1943–1954, 2020.

[23] John Biggs. It's time to build our own Equifax with blackjack and crypto. Online. http://tcrn.ch/2wNCgXu, September 2017.

[24] Eleanor Birrell, Anders Gjerdrum, Robbert van Renesse, Håvard Johansen, Dag Johansen, and Fred B. Schneider. SGX Enforcement of Use-Based Privacy. In *ACM WPES*, page 155–167, 2018.

[25] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, and Shafi Goldwasser. Secure large-scale genome-wide association studies using homomorphic encryption. *Proceedings of the National Academy of Sciences*, 117(21):11608–11613, 2020.

[26] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *ACM CCS*, 2017.

[27] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: optimizations and applications. In *ACM CCS*, 2017.

[28] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In *ITCS*, 2018.

[29] Lukas Burkhalter, Anwar Hithnawi, Alexander Viand, Hossein Shafagh, and Sylvia Ratnasamy. TimeCrypt: Encrypted Data Stream Processing at Scale with Cryptographic Access Control. In *USENIX NSDI*, 2020.

[30] Paris Carbone, Vasiliki Kalavri, Marios Fragkoulis, and Asterios Katsifodimos. Beyond Analytics: the Evolution of Stream Processing Systems. In *ACM SIGMOD Tutorial*, 2020.

[31] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering*, 36(4), 2015.

[32] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient Aggregation of Encrypted Data in Wireless Sensor Networks. In *ACM MobiQuitous*, July 2005.

[33] Yan Chen, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. PeGaSus: Data-Adaptive Differentially Private Stream Processing. In *ACM CCS*, 2017.

[34] Long Cheng, Fang Liu, and Danfeng Daphne Yao. Enterprise Data Breach: Causes, Challenges, Prevention, and future Directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(5), 2017.

[35] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *USENIX NSDI*, 2017.

[36] Cynthia Dwork. Differential Privacy. In *ICALP*, Lecture Notes in Computer Science, 2006.

[37] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*, pages 265–284. Springer Berlin Heidelberg, 2006.

[38] Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.*, 9:211–407, 2014.

[39] Eslam Elnikety, Aastha Mehta, Anjo Vahldiek-Oberwagner, Deepak Garg, and Peter Druschel. Thoth: Comprehensive Policy Compliance in Data Retrieval Systems. In *USENIX Security*, 2016.

[40] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *USENIX OSDI*, 2010.

[41] Paul Erdos and Alfred Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hungary. Acad. Sci.*, 5:17–61, 1960.

[42] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *ACM CCS*, 2014.

[43] Jon Gjengset, Malte Schwarzkopf, Jonathan Behrens, Lara Timbó Araújo, Martin Ek, Eddie Kohler, M. Frans Kaashoek, and Robert Morris. Noria: dynamic, partially-stateful data-flow for high-performance web applications. In *USENIX OSDI*, 2018.

[44] Stephanie Hare. These new rules were meant to protect our privacy. They don?t work. The Guardian, Online: https://www.theguardian.com/commentisfree/2019/nov/10/these-new-rules-were-meant-to-protect-our-privacy-they-dont-work, November 2019.

[45] Baik Hoh, Marco Gruteser, Hui Xiong, and Ansaf Alrabady. Preserving Privacy in Gps Traces via Uncertainty-Aware Path Cloaking. In *ACM CCS*, 2007.

[46] Amnesty International. The google-fitbit merger must include human rights risks. Online. https://www.amnesty.eu/wp-content/uploads/2020/11/Google-Fitbit-merger-complaint-to-the-EU-Commission-FINAL.pdf, November 2020.

[47] Hojjat Jafarpour and Rohan Desai. KSQL: Streaming SQL Engine for Apache Kafka. In *EDBT*, pages 524–533, 2019.

[48] Alan F Karr, Xiaodong Lin, Ashish P Sanil, and Jerome P Reiter. Secure regression on distributed databases. *Journal of Computational and Graphical Statistics*, 14(2):263–279, 2005.

[49] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, and Siddarth Taneja. Twitter Heron: Stream Processing at Scale. In *ACM SIGMOD*, 2015.

[50] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. Privacy-Friendly Aggregation for the Smart-Grid. In *PoPETS*, pages 175—-191, 2011.

[51] Mathias Lecuyer, Riley Spahn, Roxana Geambasu, Tzu-Kuo Huang, and Siddhartha Sen. Pyramid: Enhancing Selectivity in Big Data Protection with Count Featurization. In *IEEE Symposium on Security and Privacy*, 2017.

[52] Java BouncyCastle Cryptograpy Library. Online: https://www.bouncycastle.org/.

[53] Kevin Litman-Navarro. We Read 150 Privacy Policies. They Were an Incomprehensible Disaster. nytimes, Online: https://www.nytimes.com/interactive/2019/06/12/opinion/facebook-google-privacy-policies.html, June 2019.

[54] Alana Marzoev, Lara Timbó Araújo, Malte Schwarzkopf, Samyukta Yagati, Eddie Kohler, Robert Morris, M. Frans Kaashoek, and Sam Madden. Towards Multiverse Databases. In *ACM HotOS*, 2019.

[55] Miti Mazmudar and Ian Goldberg. Mitigator: Privacy Policy Compliance using Trusted Hardware. In *PoPETS*, number 3, page 204–221, 2020.

[56] Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient Private Statistics with Succinct Sketches. In *NDSS*, 2016.

[57] David Millman. Blog: Data Privacy, Security, and Compliance for Apache Kafka. Online: https://www.confluent.io/blog/kafka-data-privacy-security-and-compliance/.

[58] Lily Hay Newman. The Alleged Capital One Hacker Didn't Cover Her Tracks. WIRED, Online: https://www.wired.com/story/capital-one-hack-credit-card-application-data/, July 2019.

[59] Oracle. Innovation in Retail: Using Machine Learning to Optimize Retail Performance. Online: http://www.oracle.com/us/industries/retail/data-analytics-retail-perform-info-4124126.pdf.

[60] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. Big Data Analytics over Encrypted Datasets with Seabed. In *USENIX OSDI*, 2016.

[61] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *ACM SOSP*, 2011.

[62] Deloitte University Press. Intelligent automation: A new era of innovation. Online: https://www2.deloitte.com/content/dam/insights/us/articles/intelligent-automation-a-new-era-of-innovation/DUP703_IntelligentAutomation.pdf.

[63] PYMNTS. Amazon to pay consumers for their shopping data. https://www.pymnts.com/amazon/2020/amazon-to-pay-consumers-for-their-shopping-data/, 21 October 2020. Accessed: 2020-12-10.

[64] J. L. Raisaro, J. R. Troncoso-Pastoriza, M. Misbach, J. S. Sousa, S. Pradervand, E. Missiaglia, O. Michielin, B. Ford, and J. P. Hubaux. Medco: Enabling secure and privacy-preserving exploration of distributed clinical and genomic data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(4):1328–1341, 2019.

[65] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. Honeycrisp: Large-Scale Differentially Private Aggregation without a Trusted Core. In *ACM SOSP*, 2019.

[66] Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C Pierce. Orchard: Differentially private analytics at scale. In *USENIX OSDI*, 2020.

[67] Rust AES Crate. Online: https://docs.rs/aes/0.3.2/aes/.

[68] Malte Schwarzkopf, Eddie Kohler, M. Frans Kaashoek, and Robert Tappan Morris. Position: GDPR Compliance by Construction. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB 2019 Workshops*, pages 39–53, 2019.

[69] Shayak Sen, Saikat Guha, Anupam Datta, Sriram K. Rajamani, Janice Tsai, and Jeannette M. Wing. Bootstrapping Privacy Compliance in Big Data Systems. In *IEEE Symposium on Security and Privacy*, page 327–342, 2014.

[70] Elaine Shi, Richard Chow, T-H. Hubert Chan, Dawn Song, and Eleanor Rieffel. Privacy-preserving Aggregation of Time-series Data. In *NDSS*, 2011.

[71] Latanya Sweeney. Achieving k-Anonymity Privacy Protection Using Generalization and Suppression. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):571–588, 2002.

[72] Jeroen Tas. Going virtual to combat COVID-19. Online: https://www.philips.com/a-w/about/news/archive/blogs/innovation-matters/2020/20200403-going-virtual-to-combat-covid-19.html, April 2020.

[73] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing Analytical Queries over Encrypted Data. *VLDB*, 6(5):289–300, 2013.

[74] Iowa State University. Iowa State University scientists propose a new strategy to accelerate plant breeding by turbocharging gene banks. Online: https://www.news.iastate.edu/news/2016/10/03/sorghumgenebanks, October 2016.

[75] Dieter van Melkebeek and Seeun Umboh. CS 880: Advanced Complexity Theory - Lecture 23: Threshold Phenomena. Online: http://pages.cs.wisc.edu/~dieter/Courses/2008s-CS880/Scribes/lecture23.pdf.

[76] Frank Wang, Ronny Ko, and James Mickens. Riverbed: Enforcing User-defined Privacy Constraints in Distributed Web Services. In *USENIX NSDI*, 2019.

[77] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In *ACM SOSP*, 2013.

[78] Shoshana Zuboff. *The Age of Surveillance Capitalism*. Profile Books, 2019.

A Appendix

A.1 Zeph Secure Aggregation Optimization

The following section gives more details and security arguments on the optimization of the secure aggregation protocol [13] in Zeph. First, a paragraph reframes the complexity of the protocol as a graph problem, which helps to reason about possible optimizations. Afterward, the later parts build up the Zeph optimization starting from a Strawman and the original formulation by Ács et al. [13].

Secure Aggregation Graph. This paragraph introduces a graph formulation that models the complexity of the protocol. Let the secure aggregation graph $G := (V,E)$ model the symmetric protocol with N parties. The set of vertices V represents the involved parties ($|V| = N$), and the set of edges E denotes the pairwise canceling masks.

In the presence of colluding nodes, there is no better option than to aggregate the sum of non-colluding clients securely. The reason for this is that the server can always subtract the contributions from colluding clients from the total sum, which only leaves the sum of non-colluding clients.

Zeph assumes that colluding nodes are indistinguishable from non-colluding nodes. However, the total number of colluding nodes is bounded by a constant parameter $0 < \alpha \leq 1$ which guarantees at least $n \geq \alpha \cdot N$ non-colluding nodes.

More formally, let the colluding and non-colluding nodes be denoted by V^- and V^+ respectively ($V = V^+ \cup V^-$ and $V^+ \cap V^- = \emptyset$) and let $E^+ := \{(u,v) | u \in V^+ \wedge v \in V^+\}$ denote the set of edges for which both incident vertices are non-colluding ($E^- = E \setminus E^+$). The pairwise dummy keys which involve at least one colluding node serve no purpose for security. Consequently, removing all colluding nodes from V^- along with their edges E^- from the graph G does not affect the aggregation's security. This leaves the graph consisting only of non-colluding nodes $G^+ := (V^+, E^+)$ with $|V^+| = n$, which is relevant for secure aggregation.

The semantic security of the encryption scheme for generating the pairwise masks, ensures that the aggregation is secure as long as the graph of non-colluding clients $G^+ := (V^+, E^+)$ is connected (i.e., there is only a single connected component). Given that the graph is connected, an attacker cannot isolate a subgroup of non-colluding clients to reveal the aggregate of the smaller subgroup. As a result, the only option to decrypt a sum of values is by adding up all contributions. Otherwise, at least a single mask does not cancel out.

Strawman. The strawman solution for performing secure aggregation among N nodes based on pairwise canceling masks involves sharing a dummy key with all $N-1$ other nodes. As a result, the secure aggregation graph $G := (V, E)$ forms a clique, which leads to an overall complexity of $O(N^2)$. This N^2 is an upper bound on the number of required edges because, as previously shown, as long as all non-colluding clients form a single connected component, the aggregation is secure. From the perspective of each participant, the time complexity is $O(N)$ because all participants have to evaluate a PRF $N-1$ times and add together $N-1$ dummy keys.

Dream. With the goal of reducing the $O(N^2)$ complexity, Ács et al. [13] propose a distributed protocol for randomly selecting a subset of edges $E_c \subseteq E$ such that if node v_i selects v_j then node v_j also selects node v_i.

They leverage the pseudo-randomness of a PRF to create this random graph. More specifically, v_i selects v_j if $PRF(k_{ij}, r_1) \leq c$ for a constant threshold c, where r_1 is a changing public value. As a result, each edge is included with probability $p = \frac{c}{2^{128}}$ assuming a 128-bit output size of the PRF. The process of creating a random graph where each edge is independently present with a fixed probability p is also known as the Erdős–Rényi model $G(n, p)$.

An additional neat property from their process is that an attacker does not know the structure of the graph $G^+ := (V^+, E^+)$ among the non-colluding nodes and consequently cannot target specific nodes to break the graph in smaller components.

In order to prevent leaking more than one value in the unlikely event that an attacker manages to control all neighbors of a non-colluding node, the selected nodes change in every round by repeating the selection process.

However, this leads to a problem that almost nullifies the benefits of having a lower degree in the graph. To select the neighbors, each node has to evaluate the PRF $N-1$ times. Let us assume that $\ell << N$ nodes were selected. Consequently, the node has to re-evaluate the PRF among all ℓ selected nodes with a different public changing value r_2 to generate the dummy key. In total, this leads to $N-1+\ell$ PRF evaluations for every round, which is even more than in the Strawman version. The only benefit is that now only ℓ instead of $N-1$ dummy keys need to be added up.

Zeph Optimization. The idea behind the optimization in Zeph is to reduce the number of PRF evaluations by using the output of a single evaluation more efficiently. More specifically, the output of the PRF constructs W random graphs $G(n, p)$ via the Erdős–Rényi model instead of only one for the use in later rounds.

The effect of the Zeph optimization is that for W rounds, with N participants and expected number of selected neighbors $\ell << N$, the optimization must evaluate the PRF only $N-1+W\cdot\ell$ times compared to $W\cdot(N-1)$ and $W\cdot(N-1)+W\cdot\ell$ in the Strawman and Dream respectively. Note that the large N is only an additive factor in Zeph while it is to a multiplicative factor in both the Strawman and Dream.

Sharp Connectivity Threshold. Recall that the requirement for a secure aggregation is that the non-colluding nodes form a single connected component, which means that no part of the graph is isolated. In the Erdős–Rényi model, there are a fixed number of vertices n, and each edge is in the graph with probability p independently. Erdős and Rényi studied the probability that the graph $G(n, p)$ is connected as a function of p [41]. Intuitively, for very small p, $G(n, p)$ consists of mostly isolated vertices, and for large p, $G(n, p)$ is connected with high probability. It turns out that the change from disconnected to connected with high probability is quite sudden at the critical threshold p_c:

$$p_c = \frac{\ln(n)}{n}$$

If p is slightly above the threshold, that is $p \geq (1+\varepsilon)p_c$ for some $\varepsilon > 0$, then the probability that the graph is connected converges to 1 as $n \to \infty$. One can show [75] that the probability that a specific graph $G(n, p)$ is disconnected is bounded by:

$$P[G(n,p) \text{ is disconnected }] \leq \sum_{j=1}^{n/2} \left(\frac{e \cdot n}{j} \, (1-p)^{n-j} \right)^j$$

Let B_W be the event that at least one of the Erdős–Rényi graphs is disconnected. There are W random graphs and let A_i be the event that the i-th Erdős–Rényi graph $G_i(n, p)$ is disconnected. Applying the union bound results in:

$$P(B_W) = P(\bigcup_{i=1}^{W} A_i) \leq \sum_{i=1}^{W} P(A_i) = W \cdot P(A_i)$$

Given a maximal error δ and an aggregation size n, this allows to identify a W and a p such that the probability of failure (i.e. that not all W graphs are connected) is bounded from above by δ:

$$P(B_W) \leq W \cdot P(A_i) = W \cdot \sum_{j=1}^{n/2} \left(\frac{e \cdot n}{j} \, (1-p)^{n-j} \right)^j \leq \delta$$

Graph Construction from PRF. The following paragraph describes a distributed protocol for generating W secret random graphs via the Erdős–Rényi model among n vertices based on evaluating a PRF. Towards this goal, the protocol divides the output of the PRF into b-bit segments. For the moment, the explanation focuses only on a single segment (e.g., the first b bits).

As in Ács et al. [13] a node u evaluates $PRF(k_{uv}, r_1)$ for every other node v to determine the random graph structure. However, instead of only constructing a single random graph, the goal is to construct $w = 2^b$ random graphs. Towards this, the protocol assigns edge (u, v) to the i-th graph where i is the number encoded in the b-bit segment of the PRF output. The probability that a graph contains a specific edge is 2^{-b}. After repeating the procedure with every node, a vertex has in expectation a degree of $\frac{N-1}{2^b}$.

This process generates w graphs $G_i(n, 2^{-b})$ according to the Erdős–Rényi model. Note that the graphs are highly dependent on each other since each edge can only be present in one of the w graphs. Nevertheless, each graph individually satisfies the requirements for an Erdős–Rényi graph (i.e., every edge is present with probability $p = 2^{-b}$ independent of the other edges of the same graph).

Remember that so far, the protocol only used a b-bit segment of the PRF output to generate $w = 2^b$ graphs. In a PRF output size of 128 bits (e.g. AES), there are $\lfloor \frac{128}{b} \rfloor$ segments and hence using all segments allows to generate $W = \lfloor \frac{128}{b} \rfloor \cdot 2^b$ graphs.

The target is to generate as many graphs as possible (i.e., large b). A large b leads to both more rounds and a smaller expected node degree in each round, which improves performance. However, increasing b also leads to a higher probability that a graph of non-colluding nodes is disconnected, which results in a smaller aggregation size. Given the number of members N, the fraction of non-colluding members α, and a maximum error threshold δ, the optimal b is the solution to a constrained optimization problem. Let $n = \alpha \cdot N$, $p = 2^{-b}$ and find b which maximizes

$$W = \lfloor 128/b \rfloor \cdot 2^b$$

subject to the constraint

$$W \cdot \sum_{j=1}^{n/2} \left(\frac{e \cdot n}{j} \, (1-p)^{n-j} \right)^j \leq \delta$$

Note that since the space of possible b is very small (i.e., integers in between 1 and the PRF output size), Zeph finds the optimal b via brute-force. As in the protocol of Ács et al. [13], the same random graph can stay the same over t rounds, which would result in $t \cdot W$ rounds in total.

In order to get an idea of how many rounds (i.e., random graphs) are possible from performing the procedure a single time, Table 2 shows a few examples of different numbers of participants to demonstrate the effect of the optimization. The number of graphs W and the expected degree of each vertex $\mathbb{E}[\text{degree}]$ holds under the assumption that up to half the nodes are colluding $\alpha := 0.5$ and a probability of success higher than $1 - \delta$ with 1×10^{-7}.

N	W	$\mathbb{E}$[degree]
100	256	49.5
1000	512	62.4
5000	1344	78.1
10000	2304	78.1

Table 2: Expected degree of a node in a secure aggregation graph for different number of participants N and for different number of rounds W.