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

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

Lukas Burkhalter*, Nicolas Küchler*, Alexander Viand, Hossein Shafagh, Anwar Hithnawi

ETH Zürich

Abstract

As increasingly more sensitive data is being collected 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 show 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.

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 [68, 79, 81]. 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 [50, 85]. 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* [15] from data and protect *individuals' privacy* [45] 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 a growing demand for privacy solutions that protect sensitive data while retaining its value. Despite recent advancements in privacy enhancing technologies [41, 78, 84], privacy frameworks [26, 43, 44, 61, 76, 82] remain shaped by regulatory requirements that predominately focus on the notion of *notice and consent* [5, 11, 59]. Though an essential step towards transparency and user control, it is important to emphasize that user consent is not the answer to data privacy. Bad practices in data use and sharing remain pervasive in consent-based systems [48, 57, 67], and often consent does not adequately express the complexities of real-world privacy preferences. The 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 [25, 38, 66] 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

*These authors contributed equally to this work.

to exert their data privacy preferences except to give blanket consent if they choose to use the service [49, 59]. *(iii) End-to-end privacy:* Privacy solutions today are mostly ad hoc efforts [14] 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. 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" [6, 13]. 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 [24, 72, 78]) or ensure differential privacy – a mathematically rigorous definition of privacy. *(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.

The design concepts underpinning Zeph are generic and could be adapted to other systems. In this work, we specifically target data stream analytics/processing pipelines and build on the typical structure of such systems. Hence, we focus on cryptographic building blocks that optimize efficiency for this type of data. Streaming compute tasks are increasingly relevant in various privacy-sensitive sectors [17, 35, 47, 55]. The online nature of stream processing makes low latency and high throughput critical requirements for privacy-preserving stream processing solutions.

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., aggregate private data releases).

Existing practical encrypted data processing systems generally use partially homomorphic encryption schemes that already support the single-source privacy transformations required in our system [33, 39, 54, 69, 70, 80]. 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 [16, 39, 63, 73]. 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 ideas: *(i)* a new approach for encryption that decouples data encryption from privacy transformations. This logical separation of the data and privacy plane 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 *privacy 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 (MPC) and a partially homomorphic encryption scheme. 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[1] that is interfaced with Apache *Kafka* [21]. 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 end-to-end privacy and its requirements, give an overview of Zeph, and describe our security and privacy model.

2.1 End-to-End Privacy

In this work, we investigate a new cohesive end-to-end design for data privacy. Despite being heavily intertwined with users' data, data systems have evolved with design objectives centered around availability, performance, and scalability, while privacy is essentially overlooked. As privacy becomes a more urgent concern, we need system designs that retrofit privacy into existing established data framework designs. Embedding

[1] Zeph's code available at: https://github.com/pps-lab/zeph-artifact

Name	Zeph	Description
		DATA MASKING
Field Redaction [7, 9, 11]	●	Reveal some attributes and hide others
Predicate Redaction [7]	◐	Only reveal data that satisfy a predicate
Det. Pseudonym. [12]	○	Replace value with a deterministic pseudonym
Rand. Pseudonym. [12]	●	Replace value with a random pseudonym
Shifting [4]	●	Shift actual values by a fixed offset
Perturbation [41]	●	Perturb data by adding random noise i.e., additive differential privacy mechanism
		DATA GENERALIZATION
Bucketing [4, 11]	◐	Map values to a coarse space
Time Resolution [33]	●	Aggregate data across time
Population [28, 37, 39, 73]	●	Aggregate data across a population

Table 1: Overview of existing privacy transformations: Data Masking techniques (top), Data Generalization techniques (bottom). We use ● (full support), ◐ (partial support), and ○ (no support) to indicate which of these techniques are currently supported in Zeph.

privacy in the current complex, data-rich systems while ensuring the desired level of utility is, however, challenging. What is considered an appropriate privacy/utility balance in one context might not be a proper trade-off for another context. Therefore, end-to-end system designs for privacy need to account for various privacy solutions and accommodate heterogeneous privacy preferences. Next, we discuss some key design aspects for realizing end-to-end data privacy.

User-Centric Privacy. Users' perception of privacy varies widely across individuals, cultures, and contexts. Therefore, the system needs to provide the means for users to set their privacy preferences and define how their data can be accessed, processed, and shared. In practice, user preferences can also vary with respect to the trade-offs between increased privacy and utility. Their preferences can vary based on the data involved and the target consumer. 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 [71]. Therefore, a practical system needs to support a range of privacy preferences and be able to build privacy-compliant views across data covered by heterogeneous policies.

Retrofit into Existing Data Pipelines. A practical privacy solution should augment 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 with encrypted data. We want to leave the flow of data in end-to-end encrypted systems intact.

Privacy Transformations. Privacy solutions for data analytics focus on allowing the use of data or computation on data subject to privacy restrictions specified by users (e.g., restrict

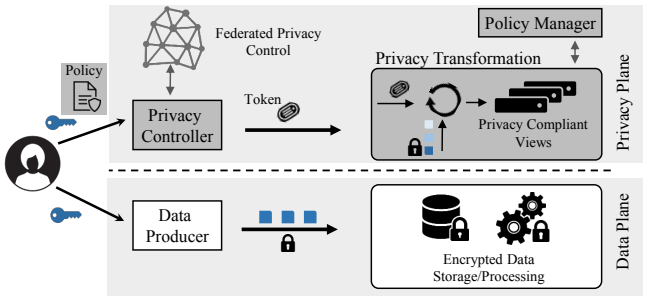


Figure 1: Overview of Zeph's end-to-end approach to privacy.

what can be inferred from the data). They are designed to enable extracting the utility from data while preserving individual's privacy preferences. This is often achieved through a range of data modifications that we refer to as privacy transformations, i.e., functions applied to the data to limit and control the extent of sensitive information revealed to authorized parties. Solutions in this space can be grouped into three broad classes (Table 1): *(i)* data masking techniques that obfuscate sensitive parts of the data, *(ii)* generalization techniques that reduce data fidelity, e.g., by aggregating data, *(iii)* combinations of *(i)* and *(ii)* which can realize complex transformations by chaining masking and generalization techniques. Privacy transformations are the primary tools to safely release data, achieving either a range of privacy guarantees common in practice (e.g., as in aggregate statistics) or formal privacy definitions such as k-anonymity [78] or differential privacy [40]. A useful end-to-end system design for privacy therefore needs to support a broad set of existing privacy transformations.

2.2 Zeph in a Nutshell

Zeph is a privacy platform that augments encrypted stream processing pipelines with the means to enforce privacy controls cryptographically. Figure 1 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 to encapsulate state-of-the-art privacy solutions (e.g., generalization, differential privacy) while preserving the data flow in existing streaming pipelines.

Privacy Plane. To illustrate a deployment of Zeph, we consider a health monitoring provider that stores health-related data from wearable devices such as heart rate and other metrics. We assume that the data streams are already end-to-end encrypted, i.e., the wearables encrypt data before uploading, while applications (e.g., health dashboard) query encrypted data and locally decrypt the result [33]. We refer to this data flow through a streaming platform as the *data plane*. The privacy logic resides and is executed outside of the data plane, allowing *data sources* to continue writing 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. A Zeph deployment augments the data plane with a *privacy plane* that enables the provider to extract information from the encrypted data streams through privacy transformers. Hence, the service gets access to privacy-compliant transformed views of the underlying raw data streams. For example, the service might collect the average heart-rate per day for different age-groups (i.e., population aggregate transformations). To collect these statistics, Zeph allows the service to express privacy options for data stream attributes through a modified data schema (§4.1), which describes a set of possible privacy transformations for attributes. Upon registering with the services, *data owners* set their privacy preferences for each stream based on these options, forming the base for users' privacy policies. For example, they can indicate if the service is allowed to include their heart rate stream in the specified aggregate transformations. The additional logic for handling privacy options and coordinating transformations is handled by an additional server component, the *policy manager*. The *policy manager* offers an API to handle privacy options per data stream and coordinates privacy transformations as *stream processors* in the streaming pipeline.

Privacy Controller. Policy enforcement in Zeph is handled by the *privacy controller*. The privacy controller is responsible for supplying the cryptographic 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 transformations efficiently. The *privacy controller* does not require access to the data and can be hosted in a location with higher availability guarantees. Zeph allows users to choose a variety of deployment scenarios for privacy controllers. Privacy controllers could be self-hosted, hosted on-premise for corporations, or outsourced to a trusted provider (e.g., OpenID identity providers).

Data Consumers. 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., 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 users' privacy.

2.3 Threat Model

Zeph enforces users' privacy preferences cryptographically, i.e., users are guaranteed that the data is transformed with the privacy transformation corresponding to their privacy preference before it is released to applications. Meanwhile, their original data remains end-to-end encrypted.

Setting. We assume an *honest-but-curious* [70] server, i.e., the server performs the computations correctly but will analyze all observed data to gain as much information as possible. We also assume the existence of a public-key infrastructure (PKI) for authentication of privacy controllers/data producers. In this setting, Zeph ensures *data confidentiality*, more specifically *input privacy*, guaranteeing that the adversary learns nothing about the raw data streams except what can be learned from the output of the transformation F (i.e., $\hat{F}$*-privacy* [39]) with some modest leakage function due to encodings (§3). Zeph also ensures that an adversary controlling the server and at most a fraction α of privacy controllers is unable to violate the privacy policies of other data owners.

Data Plane. In Zeph, data streams are encrypted at the source with a semantically secure encryption scheme, while the metadata (e.g., timestamps) is sent in plaintext. Decryption keys are never disclosed to the server; therefore, raw data confidentiality is guaranteed even in the case of a server compromise. If an adversary gains control over a data producer or the responsible privacy controller, only the data associated with that producer/controller is revealed.

Privacy Plane. Zeph ensures input privacy for honest data owners even if the stream processor executing a privacy transformation or the policy manager coordinating it is compromised by an adversary. For the case where F is an aggregation function involving data from different privacy controllers (i.e., federated privacy control), we assume that at most a fraction of α of the entities in the aggregation transformation are controlled by the adversary. Note that this can also include the server. The choice of α depends on the deployment scenario. In (§3.4), we show how this choice affects performance. For our evaluation, we use a pessimistic value of $\alpha = 0.5$, but real-world deployments might use significantly lower values.

Robustness. While Zeph can handle various failures in practice, formal robustness against misconfigured or malicious privacy controllers or data producers is out of scope for this design. A privacy controller sending corrupted tokens cannot compromise privacy but could alter the output of a transformation or prevent a transformation from completing.

3 Encryption for Privacy Transformations

In this section, we describe our approach to enable privacy transformations in end-to-end encrypted systems. Our design serves 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 encrypt data towards a fixed privacy policy. The modifications needed for privacy transformations are instead executed outside the data plane (i.e., conventional data flow), working exclusively on encryption keys to generate 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. In this realization of Zeph we focus on streaming data. Hence, we focus on cryptographic building blocks that optimize efficiency for this type of data. The design concepts underpinning Zeph are generic and could be adapted to other systems using other cryptographic building blocks. These, however, can introduce their own trade-offs between computation expressiveness and performance.

3.1 Decoupling Encryption from Privacy Transformations

This design requires an encryption approach that supports *homomorphic evaluation* in a variety of settings. Namely, it needs to support: *(i)* evaluation on encrypted data, i.e., encrypted data processing, *(ii)* construction of cryptographic *transformation tokens*, and *(iii)* combining the encrypted data with the matching cryptographic transformation tokens for selective release of privacy-compliant transformed data views.

Encrypted Data Processing. Existing encrypted data processing systems utilize homomorphic encryption schemes to enable server-side computation on encrypted data [33, 69, 70, 80]. To meet applications' stringent performance requirements, systems typically combine efficient partially homomorphic encryption schemes [27, 33, 39, 54, 69] with specialized client-side encodings to support a wider set of queries. However, standard homomorphic encryption schemes do not lend themselves to support selective release of data (i.e., handing out the decryption key allows to decrypt all data) or support privacy transformations that require data evaluation across different users (i.e., different trust domains). Supporting functions across populations in the multi-client/single-server setting is typically achieved via specialized multi-party computation protocols [16, 39, 63, 73]. These existing protocols ensure that the user inputs remain private and only the output of the function evaluation is revealed to the server. However, they require active participation by the data producers and are often limited to specific functions. We want to remove the need for – potentially resource-limited – data producers to take part in or even be aware of privacy transformations.

Homomorphic Secret Sharing. To decouple encryption from privacy transformations, we draw on ideas from the Homomorphic Secret Sharing (HSS) [31, 32] 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. HSS 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. Applying standard HSS in our setting raises two issues: *(i)* general-purpose HSS incurs non-negligible overhead [31], and *(ii)* with this approach privacy controllers remain dependent on data producers as they continue to receive a secret share for each new stream event.

To address the first issue, we employ *additively* homomorphic secret sharing. This is considerably more efficient than general-purpose HSS, allowing our system to sustain the high throughput needed for streaming data workloads. Used naively, additively homomorphic secret sharing can significantly limit expressiveness. However, as we show in the next section, using carefully selected data encodings allows us to support a wide set of privacy transformations.

To break the dependency between privacy controllers and data producers, we enable the privacy controller to independently derive the tokens (i.e., its shares) based only on metadata about the stream. Given a shared common master secret, the data producer and privacy controller never have to communicate or even be online at the same time. We introduce our scheme in more detail in §3.3. Our tailored scheme offers both the required efficiency and the flexibility necessary to decouple the data plane from the privacy plane. The data producer and the responsible privacy controller need to only agree on a shared master secret. Then, the privacy plane can authorize a transformation F by deriving the involved shares 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 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. Next, we show how we can support a broad set of privacy transformations with this scheme.

3.2 Privacy Transformation Functions

Broadly, privacy transformations (§2.1) generally involve computation/perturbation of individual user's data, computations across different users' data, or combinations of the two.

Based on this insight, Zeph exposes three core functions for developers that allow for privacy transformations in the encrypted setting: *(i)* Σ_S, which enable ciphertext aggregation operations within the same user's data streams. *(ii)* Σ_M, which enables ciphertext aggregation across streams from a population of users, *(iii)* Σ_{DP}, which supports perturbation via noise addition to streams aggregated across multiple users.

Privacy Transformations in Zeph. A privacy transformation F in Zeph is realized by combining a chain of core functions and/or withholding certain shares when creating a token. *(i) Data Masking.* Data masking techniques such as field-redaction and randomized pseudonymization are directly supported by the secrecy properties of our scheme. The privacy controller redacts or pseudonymizes a field by withholding the corresponding shares from the transformation token. Shifting and perturbation are realized with Σ_S by adding a constant or calibrated random offset to the transformation token. Zeph supports a subset of predicate redactions using client-side encodings that represent a value as a vector of elements. A privacy controller can then construct a transformation token that only reveals a subset of elements in the vector or a certain sum of the elements (Σ_S). For example, to enable predicate redaction based on a threshold, the client would encode the value as a vector of two elements. If the value is above the threshold, the client stores the value as the first element in the vector or else as the second element. To only reveal the values above the threshold, the privacy controller can disclose the first elements of the vectors with the tokens.

(ii) Data Generalization. Bucketing similarly builds on client-side encodings that map a value to a one-hot vector representing the whole domain. Instead of releasing a token for all elements, the privacy controller uses Σ_S to release a sum of shares for elements mapping to the same bucket. For values with a large domain, we can approximate the frequency count with a histogram using a larger bin width. Zeph supports data generalization over time with Σ_S and population with Σ_M. Moreover, Zeph provides Σ_{DP} to release a differentially private aggregate across a population. To extend the supported aggregation functions, we leverage existing encoding techniques [27, 33, 39, 54, 69, 83]. In essence, these encodings map a value to a vector with different statistics that allows the computing platform to execute functions by performing element-wise addition. The aggregate functions sum and count are inherently additions. With a vector of sum and count, a party can obtain the mean by dividing the sum by the count. By adding the square of a value to the encoding vector, a party can calculate the variance using that $Var(x) = \mathbb{E}(x^2) - \mathbb{E}(x)^2$. Moreover, with the one-hot encoding, constructing a histogram corresponds to the element-wise sum of a set of one-hot vectors. Given a histogram, a party can compute the median or other percentiles, min, max, mode, range, or topk. Prior work [39] presents further encoding techniques for other functions that we support in Zeph.

3.3 Transformation Tokens

In Zeph, we build upon a symmetric homomorphic encryption scheme [33] explicitly designed for streaming workloads. We use this scheme to realize efficient additively homomorphic secret sharing for our setting. The scheme efficiently derives a unique sub-key for each message from a master secret and encryption is performed via modular addition of the key and the message. Here, the encrypted message and the (message-specific) sub-key can be seen as additive shares of the message. Since encryption and decryption are linear operations, the scheme supports linear aggregation by computing the function on both the sub-keys and the encrypted messages independently. *Transformation tokens*, which authorize the release of privacy transformation results, are derived from the sub-keys via aggregations. We now describe how these are constructed in our system. 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 §3.4.

Symmetric Homomorphic Stream Encryption. First, we give a brief summary of the symmetric homomorphic stream encryption scheme [33] we build upon. 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.

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:

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

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. Keys can be aggregated the same way, but for a time-window $[t_i, t_j]$, the client can decrypt more efficiently by deriving only the two outer keys $k_i = f_k(t_i)$ and $k_j = f_k(t_j)$ because the inner keys cancel out. This encryption scheme hides the inputs from the server and allows the server to perform aggregations among the streams without accessing the plaintext data.

Authorizing Transformations. 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 Σ_S transformations, e.g., only reveal the approximate locations aggregated over a month. We start with a window aggregation to reduce the time resolution. 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 [33, 69]. 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. The privacy controller can construct transformation tokens for values with encodings (§3.2) by selectively releasing the sub-keys of certain elements in the encoding vector or by aggregating sub-keys of elements in the vector.

Multi-Stream Transformation Tokens. Multi-stream transformation tokens reveal the output of Σ_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 [40] 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., Σ_{DP}) on multi-stream window transformations, but could be extended to the single-stream setting. 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. Zeph therefore supports all additive noise mechanisms from the Differential Privacy literature [41] with noise drawn from a divisible distribution. However, mechanisms like the *Sparse Vector Technique* [42] that require access to the underlying data cannot be applied this way. In previous work, noise is added to plaintexts prior to encryption, whereas in Zeph noise is added to the decryption keys. The two approaches are cryptographically equivalent. However, previous work requires deciding on the noise to add at encryption time. Our approach has the advantage of allowing noise to be added to data that was previously encrypted without consideration for noise. This also means, that the same data is reusable for encrypted storage and to facilitate one or multiple differentially private privacy transformations.

3.4 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. While we assume that data owners trust their own privacy controller, different data owners might not want to trust the *same* controller. In such multi-trust 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 send a combined token for the aggregation of the streams under their control. However, this leaks the intermediate result from each controller to the server. Instead, we need the individual tokens to reveal no additional information while still enabling correct decryption of the transformation output. We enable this in Zeph using secure aggregation [16, 28], a specialized secure MPC protocol. For our system, 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. [16] 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 core protocol and then describe our optimizations that reduce the computation cost for privacy controllers.

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 Σ_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 to the other privacy controllers. Each privacy controller masks its input τ_p with a nonce k_p, i.e., it computes $\tau_p + k_p \mod M$. 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:

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

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 $k'_{p,q}$.

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

Hence, the pairwise secrets cancel each other out when the masks are combined in the aggregation. For conciseness, we refer to Ács et al. [16] for a description of dropout handling.

Constructing Canceling Nonces. In Zeph, the secure aggregation protocol is run repeatedly for multiple rounds due to the continuous nature of streaming queries. 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 repeated over a long period of time with similar participants, i.e., the long-running nature of streaming queries.

Online Phase Optimization. We reduce the communication overhead during the online phase by choosing privacy controller's nonce as to only include 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[2], 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. [16] 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 extended version of this paper [34], 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. [16] requires 23.2 million PRF evaluations and 180k additions[3].

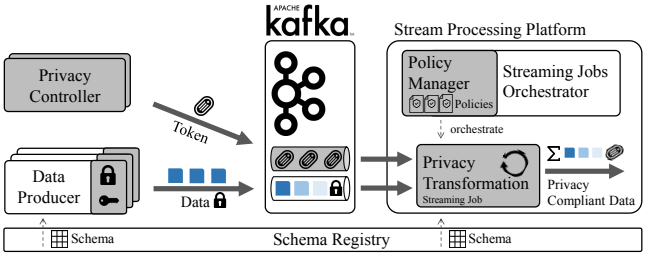


Figure 2: Overview of Zeph's architecture and integration into existing data streaming pipelines. Zeph's components are highlighted in gray.

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.

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

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

Zeph augments existing stream processing pipelines, similar to existing frameworks operating on data in-the-clear [64]: *(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 (i.e., user-centric privacy) 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 to demonstrate how Zeph can be used in practice. 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 data minimization techniques can differ depending on the data type, e.g., image, location, heart rate).

Data Stream Schema. In Zeph, developers can translate user preferences to an application-specific set of transformations by mapping them in a schema language. Zeph's schema language builds on the *Avro* [20] 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). For example, the region where a data stream originates from (Figure 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). For example, a heart rate sensor might have two stream attributes such as heart-rate and heart-rate variability (Figure 3). The heart-rate is annotated to support aggregates with variance statistics. *(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* (Σ_S), *aggregate* (Σ_M), and *dp-aggregate* (Σ_{DP}) 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.

```
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)

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 data producers 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.

Encrypting Data Streams. The proxy module encrypts each record with a symmetric homomorphic encryption scheme (§3), using the master secret from the setup phase. In order to allow the privacy controller to derive a transformation token without observing the data (§3), the data producer sends a neutral value at regular intervals (e.g., every minute) to terminate the window. This does not affect the result of computations but is required for efficient Σ_S transformations across time. Additionally, these messages allow Zeph's microservice to detect and handle dropout of data producers (e.g., due to network interruptions).

4.3 Matching Queries with Privacy Policies

Zeph's microservice contains a policy manager that maintains a global view of the system and coordinates 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 constructed from chains of the core operations (§3.2) and are executed as stream processing queries running continuously on a set of encrypted streams.

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. The policy manager needs to ensure that queries comply with all the stream's selected policy options. Otherwise, it will not receive the required transformation tokens from the privacy controllers.

Query Language. The query language of Zeph builds on *ksql* [51], 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 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 Σ_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 Σ_M or Σ_{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 transformation, and is removed from the set of queriable streams for this attribute as long as the stream is part of the running transformation. The privacy controller generally only supplies a single transformation token for each window in a given stream, preventing differencing or re-use attacks. For DP aggregations, a stream value can contribute to multiple transformations if allowed by its current privacy budget. The privacy controller maintains the privacy budget and suppresses transformation tokens if the privacy budget is used up. After processing the query, the query planner outputs a *transformation plan* that encodes the list of streams in the transformation, fault tolerance details (i.e., number of participants dropout the system can handle), and the sequence of operations the system need to perform (Figure 4).

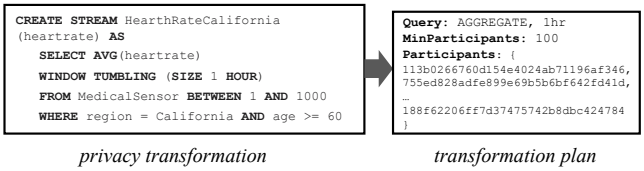


Figure 4: The query planner converts privacy transformations into transformation plans with complying data streams.

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 (§3.4) 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.

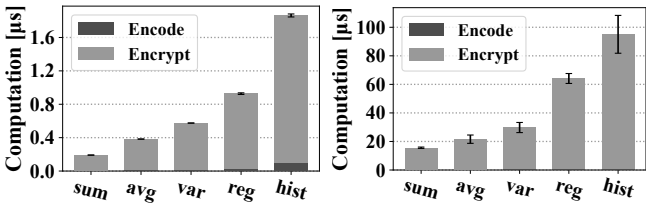
After receiving an update, the privacy controllers verify that the transformation still complies with the selected privacy options and update the tokens they send to match the new transformation. Upon the arrival of all transformation tokens, Zeph can complete the transformation and output the result.

5 Implementation

Our prototype of Zeph is implemented on top of Apache *Kafka* [21], consisting of roughly 4500 SLOC for Zeph and and additional 5500 SLOC for benchmarks. We provide a data producer proxy library written in Java that relies on the Bouncy Castle library [58] for cryptographic operations, and *Avro* [20] 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 [74], and for the ECDH key exchanges we use the `secp256r1` elliptic curve from Bouncy Castle [58]. We use the Apache *Kafka* Streams [22] to implement the stream processor for the privacy transformations. We emulate the policy manager with a configurable Ansible [19] 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: sum, average, variance, linear regression, histogramm. The encoding for the histogramm has ten buckets.

6.1 Experimental Setup

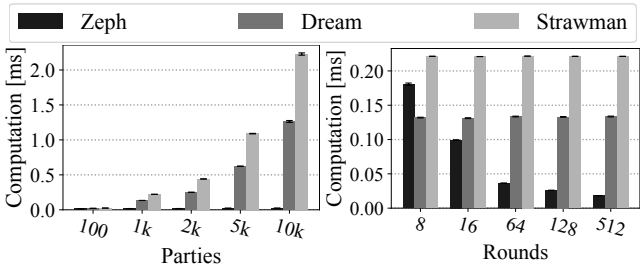
The experimental evaluation consists of two parts. First, we quantify the overhead of Zeph components with microbenchmarks. We start by quantifying the performance of our proposed secure aggregation optimization compared to a *Strawman* with no optimizations (§3.4) and the optimized protocol by Ács et al. *Dream* [16]. 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 [18], 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 2: 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 exchange dominates 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 *Enc* is 0.19μs on EC2 and 16μs on a Raspberry Pi, the cost is low because the encryption scheme relies on symmetric primitives (i.e., efficient AES). Figure 5 shows the encryption latency for different encodings. A data producer can encrypt events at a rate 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.

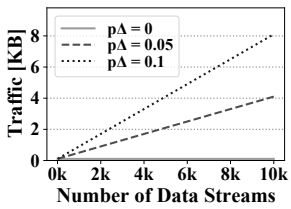
6.3 Privacy Controllers

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.

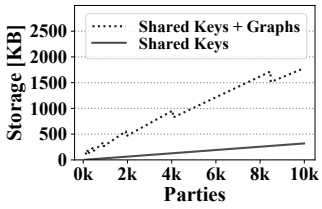
For the multi-stream case, the privacy controller engages in the secure aggregation protocol (§3.4). We quantify the overhead by running the secure aggregation protocol for different numbers of privacy controllers and compare it against the strawman approach. As a first step, all these protocols require 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 controllers, i.e., $O(N^2)$. However, we assume that realistic deployments will feature at most a few thousand privacy controllers in a single aggregation. Beyond this point, further scalability should be realized through hierarchical transformations. In our evaluation, we explore aggregations with up to 10k privacy controllers, which is the current limit of feasibility without resorting to hierarchies. Table 2 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 when 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 (§3.4). 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



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



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

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

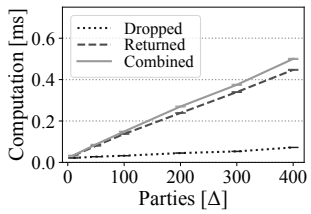


Figure 8: Computation cost for a privacy controller to adapt to Δ dropping or joining parties. In the combined case, Δ members dropped out and Δ other members returned.

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.

Dropout. 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 dropout 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 dropout 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.

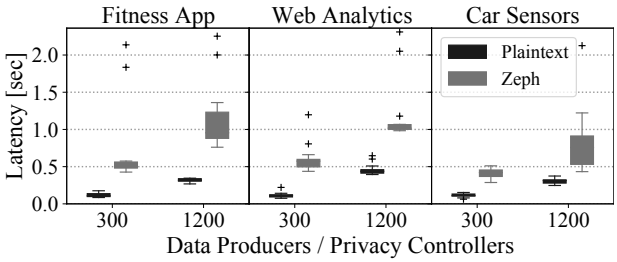


Figure 9: Computation cost for Plaintext (no encryption) and Zeph for different Applications. The latency measures the time after the grace period (5s) of a window is over until the result of the transformation is available.

Fitness Application. We consider the Polar App [10] which 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 [8] 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. Zeph completes processing the current window 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. Enforcing privacy policies automatically in real-world data processing systems is often

achieved by resorting to Information Flow Control (IFC) to check and constrain how information flows through the system [26, 43, 44, 61, 76, 82]. 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 for privacy enforcement. Riverbed [82] 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 [23] 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* [60, 75], 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. Qapla [62] allows a policy compliance team to associate a set of policies to database schemas, which a trusted reference monitor then enforces.

Private Aggregate Statistics. Secure aggregation protocols have been used in a variety of private system designs, largely to enable services to collect statistics over users' data without accessing individual data [16, 28, 36, 39, 56, 63, 73, 77]. Compared to Zeph, these systems require data producers to actively participate in the aggregation protocol, keep data local, and do not support a wide range of privacy transformations. While we utilize a secure aggregation protocol [16, 28] to construct privacy transformation tokens that require inputs from multiple trust domains, this does not impact data producers in Zeph design. Several systems [16, 63, 73, 77] combine differential privacy techniques [40, 42] (i.e., by adding noise to inputs) with secure aggregation in a way that minimizes the amount of added noise. This line of work is orthogonal to this work, and some can be integrated with Zeph.

Private Outsourced Computation. A different line of work investigates how to protect the confidentiality of data while allowing a server to compute on encrypted data either with homomorphic encryption [33, 69, 70, 80] or secret sharing [52, 53]. This line of work is orthogonal to Zeph, and the goal of Zeph is to augment these systems with the capability to selectively release encrypted data following an evaluation of a privacy transformation. Encrypted processing systems can be adapted to perform privacy transformations but then require clients first to decrypt the outputs. Zeph supports both direct release of privacy-compliant views of data and privacy transformations to a targeted authorized party.

Functional Encryption. Another closely related line of work is functional encryption [29, 30, 46] (FE). Functional encryption allows a data owner to issue restricted secret keys that enable the key holder to learn only the output of a specific function. Existing constructions are currently not yet efficient enough for practical systems. Additionally, some of the privacy transformations in Zeph require functions on multiple inputs from multiple trust domains, which requires techniques that are even more complex than standard FE [65].

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.

Acknowledgments

We thank our shepherd Amit Levy, the anonymous reviewers, Hidde Lycklama, and Emanuel Opel for their valuable feedback. This work was supported in part by the SNSF Ambizione Grant No. 186050 and an ETH Grant.

References

[1] Bosch Predictive Maintenance. Online: https://www.bosch-mobility-solutions.com/en/products-and-services/mobility-services/predictive-diagnostics/. Accessed: 09-12-2020.

[2] California Consumer Privacy Act (CCPA). CCPA, Online: https://oag.ca.gov/privacy/ccpa. Accessed: 09-12-2020.

[3] General Data Protection Regulation: GDPR. GDPR, Online: https://gdpr-info.eu/. Accessed: 09-12-2020.

[4] Google Cloud De-identification. Online: https://cloud.google.com/dlp/docs/classification-redaction. Accessed: 09-12-2020.

[5] Immuta Platform. Online: https://www.immuta.com/. Accessed: 09-12-2020.

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

[7] IRI Total Data Management Redaction. Online: https://www.iri.com/blog/data-protection/redaction-options-for-data-privacy/. Accessed: 09-12-2020.

[8] Matomo Web Analytics. Online: https://matomo.org/. Accessed: 09-12-2020.

[9] Oracle Responsys Data Redaction. Online: https://docs.oracle.com/en/cloud/saas/marketing/responsys-user/DataRedaction.htm. Accessed: 09-12-2020.

[10] Polar Platform. Online: https://www.polar.com/accesslink-api/#detailed-sport-info-values-in-exercise-entity. Accessed: 09-12-2020.

[11] Privitar Privacy Platform. Online: https://www.privitar.com/. Accessed: 09-12-2020.

[12] Pseudonymisation techniques and best practices. Online: https://www.enisa.europa.eu/publications/pseudonymisation-techniques-and-best-practices/. Accessed: 09-12-2020.

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

[14] 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.

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

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

[17] 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.

[18] Amazone MSK. Online: https://aws.amazon.com/de/msk/. Accessed: 09-12-2020.

[19] Ansible. Online: https://www.ansible.com/. Accessed: 09-12-2020.

[20] Apache Avro. Online: https://avro.apache.org/. Accessed: 09-12-2020.

[21] Apache Kafka. Online: https://kafka.apache.org/. Accessed: 09-12-2020.

[22] Apache Kafka Streams. Online: https://kafka.apache.org/documentation/streams/. Accessed: 09-12-2020.

[23] 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*, 2019.

[24] E. Balsa, C. Troncoso, and C. Diaz. OB-PWS: Obfuscation-Based Private Web Search. In *IEEE Symposium on Security and Privacy*, 2012.

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

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

[27] 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.

[28] 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.

[29] Dan Boneh, Amit Sahai, and Brent Waters. Functional Encryption: Definitions and Challenges. In *TCC*. Springer, 2011.

[30] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*. Springer, 2014.

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

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

[33] 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.

[34] Lukas Burkhalter, Nicolas Küchler, Alexander Viand, Hossein Shafagh, and Anwar Hithnawi. [Extended Version] Zeph: Cryptographic Enforcement of End-to-End Data Privacy. In *arXiv*, 2021.

[35] 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.

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

[37] Claude Castelluccia, Aldar CF Chan, Einar Mykletun, and Gene Tsudik. Efficient and Provably Secure Aggregation of Encrypted Data in Wireless Sensor Networks. *ACM TOSN*, 2009.

[38] 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.

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

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

[41] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*. Springer, 2006.

[42] Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.*, 2014.

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

[44] 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.

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

[46] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *IEEE FOCS*, 2013.

[47] 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.

[48] Eloise Gratton. Beyond Consent-based Privacy Protection. Online: https://www.eloisegratton.com/files/sites/4/2016/07/Gratton_Beyond-Consent-based-Privacy-Protection_-July2016.pdf, July 2016.

[49] 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.

[50] 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.

[51] Hojjat Jafarpour and Rohan Desai. KSQL: Streaming SQL Engine for Apache Kafka. In *EDBT*, 2019.

[52] Thomas P. Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A Framework for Outsourcing of Secure Computation. In *ACM CCSW*, 2014.

[53] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing Multi-Party Computation. *IACR Cryptol. ePrint Arch. Report 2011/272*, 2011.

[54] 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.

[55] 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.

[56] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. Privacy-Friendly Aggregation for the Smart-Grid. In *PoPETS*, 2011.

[57] Crystal Lee and Jonathan Zong. Consent Is Not an Ethical Rubber Stamp. Online: https://slate.com/technology/2019/08/consent-facial-recognition-data-privacy-technology.html, August 2019.

[58] Java BouncyCastle Cryptograpy Library. Online: https://www.bouncycastle.org/. Accessed: 28-04-2020.

[59] 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.

[60] 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.

[61] Miti Mazmudar and Ian Goldberg. Mitigator: Privacy Policy Compliance using Trusted Hardware. In *PoPETS*, 2020.

[62] Aastha Mehta, Eslam Elnikety, Katura Harvey, Deepak Garg, and Peter Druschel. Qapla: Policy Compliance for Database-Backed Systems. In *USENIX Security*, 2017.

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

[64] David Millman. Blog: Data Privacy, Security, and Compliance for Apache Kafka. Online: https://www.confluent.io/blog/kafka-data-privacy-security-and-compliance/. Accessed: 09-12-2020.

[65] Muhammad Naveed, Shashank Agrawal, Manoj Prabhakaran, XiaoFeng Wang, Erman Ayday, Jean-Pierre Hubaux, and Carl Gunter. Controlled Functional Encryption. In *ACM CCS*, 2014.

[66] 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.

[67] Cristina Onose. 10 privacy issues and trends. Online: https://www.pwc.com/ca/en/services/consulting/privacy/privacy-canadian-business-hub/2020-and-beyond-10-privacy-issues-and-trends-part-1.html, January 2020.

[68] 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. Accessed: 09-12-2020.

[69] 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.

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

[71] 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.

[72] 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.

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

[74] Rust AES Crate. Online: https://docs.rs/aes/0.3.2/aes/. Accessed: 09-12-2020.

[75] 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.

[76] 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*, 2014.

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

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

[79] 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.

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

[81] 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.

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

[83] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical Private Queries on Public Data. In *USENIX NSDI*, 2017.

[84] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-Scale Secure Multiparty Computation. In *ACM CCS*, 2017.

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