Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

RESEARCH GROUP FOR
Distributed Systems

Master Thesis

Institute for Pervasive Computing, Department of Computer Science, ETH Zurich

Enforcement of Privacy Policies via Encryption for Distributed Unbounded Data

by Nicolas Küchler

ETH student ID:	14-712-129
E-mail address:	kunicola@student.ethz.ch
Supervisors:	Lukas Burkhalter
	Dr. Anwar Hithnawi
	Prof. Dr. Friedemann Mattern
Date of submission:	May 15, 2020

Abstract

The growing demand for gaining valuable insights from a wide variety of sectors leads to the exposure of increasingly sensitive user data. The associated privacy risks prompted regulatory bodies to react with new privacy regulations that create a pressing need to natively integrate privacy controls into data analytics frameworks not only to protect privacy but also to fulfill compliance.

This thesis presents the design of PolicyCrypt, a privacy platform that relies on encryption to enforce users' privacy. PolicyCrypt follows a user-centric model to privacy that allows users to restrict how their data can be stored, exposed, and processed. Users express their preferences with a simple privacy policy language that the platform enforces via encryption.

PolicyCrypt offers a framework to reconcile the privacy interests of users, with the demand for extracting valuable information from sensitive data. Towards this, PolicyCrypt implements a unified interface that provides privacy transformed views of data streams. The data streams remain end-to-end encrypted and the platform only exposes transformed streams that comply with users' privacy policies to applications and data analytics frameworks.

PolicyCrypt executes privacy-preserving data transformations in real-time and scales to thousands of distributed data sources, which enables large-scale low-latency data streaming analytics. This work introduces a hybrid MPC-PHE approach to support scalable and efficient privacy-preserving data transformations over multiple encrypted data streams across users. As part of this approach, this thesis proposes a new optimized variant of a secure aggregation protocol, tailored explicitly to the recurring nature of streaming analytics. Finally, a prototype implementation of the platform demonstrates PolicyCrypt's practicality and performance in realistic application scenarios.

Acknowledgements

For the last six months, I had the opportunity to work with several people on the design of PolicyCrypt. Without their valuable insight, expertise, and last but not least, their support, my thesis on the intersection of distributed systems, privacy, and cryptography would not have been possible. I want to start by thanking my supervisors Lukas Burkhalter and Dr. Anwar Hithnawi. The numerous discussions with Lukas were instrumental in the design of PolicyCrypt and Anwar's helpful advice and constructive feedback further enhanced the thesis. I truly enjoyed the collaboration with both of them because they provided me with advice and courage to pursue my interests and ideas.

Furthermore, I would like to thank Dr. Hossein Shafagh and Alexander Viand for their valuable feedback on aspects of the PolicyCrypt design and Dominik Bründler for the feedback on the written thesis. Also, I would like to thank Prof. Friedemann Mattern for the opportunity to conduct the master thesis in his group.

Moreover, I would like to express my gratitude to Valentina Betschart, for all her love and support. Last but not least, a special thanks goes to my family for creating an environment that allowed me to thrive and for their unwavering support. My journey of education and life would not have been possible without them.

Contents

1 Introduction

Recent years have seen unprecedented growth in the collection of sensitive data. New data is generated at a tremendous rate, and volume and data sources become more and more pervasive. This development is primarily attributed to the recognition that there is immense value in extracting information from data, especially when data from different sources is brought together.

An important emerging type of data is streaming data. The collection of these unbounded streams of data is increasingly prevalent across a wide range of systems in diverse domains such as health, agriculture, transportation, operational insight, and smart cities [13, 31]. The growth of streaming data is largely associated with the rising demand for instrumentation. Individuals and organizations are continuously logging various metrics that report the state of systems or organisms for better diagnoses, forecasting, decision making, and resource allocation in timely manners. The growth of streaming data is only expected to continue with further advances in sensor technology, data analytics, machine learning, and cloud computing.

However, with more data being collected and processed, reports on unauthorized selling and sharing of data are rising [16, 28, 73]. These growing concerns regarding data privacy have led regulatory bodies to haste to work on data privacy regulations to mitigate private data misuse [3,86]. A legitimate concern is that privacy regulations are leading to more siloed sensitive data, hence, diminishing the potential of data in advancing many vital sectors. It thus remains unclear how data can be safely brought together to derive valuable insights and simultaneously respect individuals' rights to privacy.

While mechanisms for data security targeting authorization, access control, and data protection have matured and are natively integrated into many systems and data analytics frameworks [9–11], their equivalence for data privacy remains primitive. Today there is a shortage of technical tools for privacy in data analytics platforms.

Data Privacy – Status Quo The current data privacy model builds on privacy policies that describe how data is collected, processed, and shared. Privacy policies are documents written by legal departments to comply with existing privacy regulations. Users are left with no choice but to consent to the target service's privacy policy if they wish to use it. Despite the new privacy regulations, most of the auditing and enforcement of compliance is done manually by organizations. Even though there have been some efforts to integrate privacy mechanisms in products that collect sensitive data, these remain the exception and are done by organizations with significant privacy and security teams [6–8].

We identify three main issues with the status quo: *(i)* users lack both control over their data and expressing their privacy preferences, *(ii)* there is a lack of privacy tools and frameworks for automatic compliance and privacy enforcement, *(iii)* resulting in data curators being trusted to enforce privacy, which often leaves data vulnerable to data breaches.

In essence, what is missing is a user-centric end-to-end approach to data privacy that starts at the source of the data. To be effective, such an approach must easily integrate into existing data processing pipelines and can coexist with data protection mechanisms in place. Moreover, a compelling approach should democratize privacy in the current data frameworks and put users in control of how and in what form their data can be used. Also, the manual enforcement of compliance should be replaced with automatic enforcement of privacy providing rigorous guarantees.

The work outlined in this thesis addresses the simultaneous simplification of privacy compliance and protection of the user's privacy. Doing so might help towards unlocking valuable but currently inaccessible data silos that could help to tackle crucial problems in domains such as health, transportation, and smart cities.

1.1 Approach

This thesis presents PolicyCrypt, a design of a new privacy platform that integrates privacy-preserving transformations into existing streaming analytics platforms while empowering users with unparalleled control over their data (Figure 1.1).

PolicyCrypt is the missing link between users and service providers that allows for privacy protection while automatically enforcing compliance. To do so, PolicyCrypt offers users an interface to express their data stream privacy preferences in the form of an expressive yet straightforward privacy policy language. Furthermore, PolicyCrypt provides a streaming query language that service providers can use to express their data requirements. Last, PolicyCrypt outlines a simple matching algorithm to reconcile the service provider's interest in specific streaming queries with user-defined privacy policies.

The platform follows an end-to-end encryption paradigm where data is encrypted at the source and remains protected until the data is in a form that respects the

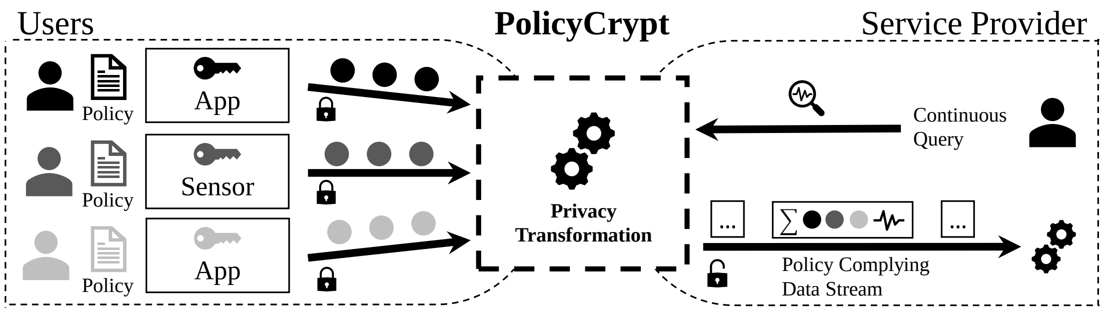


Figure 1.1: Approach of the PolicyCrypt privacy platform.

user-defined privacy preferences. In PolicyCrypt, the user-defined privacy policies are enforced with encryption, which provides users with strong data protection guarantees. This approach enables performing privacy transformations in an untrusted cloud environment without granting access to raw data.

1.2 Challenges

A compelling privacy platform for streaming data must overcome several system, privacy, and security challenges. On the privacy end, the main challenge is that such a platform must provide cryptographic guarantees regarding data protection, which ensure that only users themselves have access as long as the data is not in a privacy-preserving form.

Furthermore, a challenge that all user-centric privacy approaches share is that users can express different privacy preferences. As a result, a privacy platform must be able to handle a diverse set of privacy policies where conflicts may arise.

On the system side, modern stream processing pipelines process data in motion in real-time. A privacy platform for streaming data must thus be optimized to achieve low-latency while simultaneously scale to thousands of resource-constrained data sources. The design of PolicyCrypt addresses all these challenges.

1.3 Contributions

The contributions of this thesis are:

- The design of PolicyCrypt, a new privacy platform for streaming data that relies on encryption to enforce user-defined privacy policies.

- A new privacy policy language and an adaption of an existing streaming query language for users and service providers to express their interests.

- A new approach that decouples privacy transformations from encryption at the source. This decoupling allows performing transformations corresponding to various privacy policies without requiring re-encryption of data.

- A new technique that enhances an existing secure aggregation protocol from Ács et al. [12]. The optimization is specifically tailored for recurring stream processing workloads and helps PolicyCrypt to meet the tight latency requirements of target applications.

- A prototype implementation of PolicyCrypt on top of Apache Kafka [46].

- A comprehensive system evaluation to quantify the overheads of PolicyCrypt in terms of computation time, throughput, latency, bandwidth, and storage against a reference implementation operating on unencrypted data. Furthermore, the thesis provides an evaluation of the optimized secure aggregation protocol that compares the performance to the original protocol.

1.4 Thesis Outline

The remainder of this thesis is structured as follows: The thesis starts with presenting background material that is necessary to understand the design rationale of PolicyCrypt (Chapter 2). Then, it presents a discussion on the related work (Chapter 3). Since this thesis proposes a new approach to privacy with PolicyCrypt, the discussion in related work targets aspects close to this work. The thesis continuous by presenting an overview of the system design (Chapter 4), describes the cryptographic constructions of this work (Chapter 5), and explains system design aspects of PolicyCrypt (Chapter 6). It further covers the implementation of the PolicyCrypt prototype (Chapter 7) and evaluates the performance of PolicyCrypt (Chapter 8). Finally, the thesis concludes and outlines possible future directions for this work (Chapter 9).

2 Background

This chapter discusses the relevant background for the privacy platform PolicyCrypt presented in this thesis. The chapter starts with an introduction to the data privacy landscape. Then, known secure computation techniques that help to facilitate privacy-preserving transformations are reviewed. Lastly, the background chapter presents data stream analytics, which is the scenario that the privacy platform targets.

2.1 Data Privacy

The rate and volume of generated data is unprecedented and data collection techniques have become more pervasive. This development is primarily driven by *(i)* the realization that there is enormous value in data, and *(ii)* by the fact that advances in data analytics put the privacy of an individual at risk, mainly because most of the collected data is highly sensitive (e.g. health, location, or financial data). This section starts by describing the status quo and later introduces existing techniques that allow deriving valuable insight while protecting the individual's privacy.

2.1.1 Privacy Regulations and Policies

Regulatory bodies in the European Union and California reacted to the increasing concerns over data protection and privacy by introducing new privacy laws. Since May 2018, all companies operating within the European Union are subject to the General Data Protection Regulation (GDPR). While all Californians are protected by the California Consumer Privacy Act (CCPA) since January 2020. Both of these data protection rules aim to give people control over their data [3, 18]. Services that fail to comply with these privacy laws run at the risk of receiving hefty fines.

A central point of the privacy regulations is that services must describe in a privacy policy (or privacy notice) how user data is used and processed. Today, the majority of existing privacy policies are legal statements written in natural language, which rely on notice and consent to protect privacy. A single sentence from the Airbnb privacy policy in Listing 2.1 gives an example that highlights the complexity of some of the existing privacy policies. The idea of notice and consent is that a user decides on using services based on such privacy policies.

Listing 2.1: Extract from Airbnb's Privacy Policy [14].

```
In jurisdictions where Airbnb facilitates the
 Collection and Remittance of Occupancy Taxes
 as described in the Taxes section of the
Terms of Service, Hosts and Guests, where
legally permissible according to applicable
law, expressly grant us permission, without
further notice, to disclose Hosts and Guests
data and other information relating to them
or to their transactions, bookings,
Accommodations and Occupancy Taxes to the
relevant tax authority, including, but not
limited to, the Hosts or Guests name, Listing
 addresses, transaction dates and amounts,
tax identification number(s), the amount of
taxes received (or due) by Hosts from Guests,
 and contact information.
```

The next two paragraphs discuss the shortcomings of the status quo from both the user- and service provider viewpoint.

User Privacy In combination with the paradigm of notice and consent, today's privacy policies lack expressiveness, scalability, transparency, and enforcement [17]. To a large degree, these problems remain unsolved even under GDPR and CCPA because they are inherent to the current form of privacy policies written in natural language.

- *Expressiveness:* In notice and consent, there is only an all-or-nothing choice for users. There is no fine-grained mechanism that allows users to limit what data can be used and in which form.
- *Scalability:* The assumption behind notice and consent is that users read the privacy policy of each service that they use and then decides whether to consent or to look for an alternative. A study from 2008 [68] suggests that an average American internet user would have to invest 244 hours in reading all privacy policies from services with which they interact within a single year. Consequently, it is no surprise that most users skip reading the privacy policy and directly give uninformed consent. The main problem here is not the large number of services that people use but rather the length of privacy policies. An experiment from 2019 shows that some privacy policies take almost 35 minutes to read [66].
- *Transparency:* The requirement for informed consent must be that the user understands the privacy policy. However, evidence suggests that verbosity and the amount of legal jargon make many privacy policies hard to understand,

even for people with excellent education [66]. Moreover, a high-level policy defined in human-readable language inevitably passes over details that might be relevant. A low-level policy with all details might be too complicated for a user to understand.

- *Enforcement:* The enforcement of privacy policies relies on trust in the service provider. There is no convincing mechanism for enforcing privacy policies. Instead, the mechanism to enforce privacy policies is based on the idea of legal concerns over hefty fines. However, the number of data breaches reported under the GDPR obligation to notify demonstrates that these concerns are not sufficient. The latest published numbers show that in the first year, 89’271 data breaches were reported across Europe [19].

Privacy compliance The number of reported data breaches under GDPR is also an indicator that services struggle to fulfill compliance with the new regulations. Having a GDPR or CCPA compliant privacy policy is only the first step. The next and arguably more difficult step is to also adhere to the privacy policy. The problem highlighted by the number of data breaches is not that companies willingly violate their privacy policy. Instead, the lack of enforcement mechanisms in combination with access to raw data results in a situation where small mistakes in handling data can have far-reaching consequences. The potential fines and the loss of reputation pose a significant threat to companies.

It becomes evident that a paradigm shift is necessary because the current system is neither adequate to provide privacy to users nor to help services fulfill compliance regulations. The next segment discusses existing technologies that enable privacy-preserving transformations on user data, which are the fundamental building blocks for the new privacy paradigm proposed in the thesis.

2.1.2 Data Privacy Technologies

Two of the most straightforward privacy-preserving transformations are anonymization and pseudonymization. The idea of anonymization is to mask personal data by removing identifying attributes such as name or address before publishing data. At first sight, this transformation appears to be irreversible. The idea behind pseudonymization is similar; however, the masking of private information should be reversible given a decryption key. However, without access to the decryption key, this transformation also looks irreversible at first. Unfortunately, many examples show that both anonymization and pseudonymization do not work as well as initially thought [35, 75, 84]. The problem is that the combination of insensitive attributes often acts as a unique fingerprint of a user. An attacker can link this fingerprint to public information and, as a result, often completely reverse the transformation and identify an individual.

Another privacy-preserving transformation is aggregation. One of the most common phrases in privacy policies is that the service uses or shares data in aggregate form. A search for the term “aggregat”, which covers both the terms “aggregate” and “aggregation”, in the 2020 opt-out choice dataset [62] reveals that it appears in almost half of the privacy policies (371 of 767) . However, most of the time, it is unclear what aggregation entails. For example, even though technically, a sum of two participants is data in aggregated form, it is intuitively clear that this form of aggregation provides little privacy.

In general, it must be stated that aggregated data alone does not provide any formal privacy guarantees. However, in many cases, aggregation of comparable data over a sufficient number of participants provides practical privacy. For example, consider a vote or an election, where this form of privacy is applied for hundreds of years. At the very least, an aggregation over many participants preserves plausible deniability for each individual.

Furthermore, many types of data are highly sensitive at a high resolution but much less so aggregated over a certain period, for example, the household’s power consumption or the number of steps of an individual. This example is a form of practical privacy through aggregation over time.

The next section introduces a technique which can be used in situations where it is unclear whether practical privacy via aggregation is sufficient.

2.1.3 Differential Privacy

Differential privacy (DP) provides a more rigorous notion of privacy by formalizing the goal of protecting an individual’s privacy. Intuitively, DP preserves privacy by ensuring that the output of the analysis remains approximately the same, independent of whether the dataset includes the individual or not. In such a situation, the rationale is that an outsider learns nothing new about individuals if they participate in the database. As a consequence, individuals can participate without compromising their privacy.

More formally, let the ℓ_1 distance between two databases x and y $\|x - y\|_1$ be the measure of how many records differ between the databases. Two databases are neighbouring if $\|x - y\|_1 = 1$ or in other words if they differ only by a single record [38].

Definition 2.1 (Differential Privacy) *Let $\mathcal{O}$ be the set of all possible outputs. A mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{O}$ is (ϵ, δ)-differentially private if for all sets $\mathcal{S} \subseteq \mathcal{O}$ and for every pair of neighbouring databases $x, y \in \mathcal{D}$*

$$Pr[\mathcal{M}(x) \in \mathcal{S}] \leq e^{\epsilon} \cdot Pr[\mathcal{M}(y) \in \mathcal{S}] + \delta$$

This definition captures that a mechanism $\mathcal{M}$ behaves similarly on neighboring databases.

Laplace Mechanism One of the most widely applicable mechanisms is the Laplace mechanism because it allows answering numeric queries while preserving differential privacy [38]. The idea is to protect privacy by adding Laplacian noise to the query result. For hiding the contribution of a single individual in a numeric query f, the mechanism needs to know the magnitude by which the data of an individual can change the function f in the worst case, as this is proportional to the amount of noise which is necessary. The ℓ_1-sensitivity of a function$f$: $\mathcal{D} \to \mathbb{R}^d$ captures precisely this for neighbouring databases x and y

$$\Delta f = \max_{x,y \in \mathcal{D} \| x-y \|_1 = 1} \| f(x) - f(y) \|_1$$

Definition 2.2 (Laplace Mechanism) *Given any function f: $\mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^d$, the Laplace mechanism is defined as:*

$$\mathcal{M}_L(x, f(\cdot), \epsilon) = f(x) + (Y_1, \cdots, Y_d)$$

where Y_i are i.i.d random variables drawn from the Laplace distribution Lap$(\Delta f/\epsilon)$.

The Laplace distribution (centered at 0) with scale b is the distribution with the probability density function:

$$Lap(x|b) = \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right)$$

One can show that the Laplace mechanism preserves $(\epsilon, 0)$-differential privacy [38].

Gaussian Mechanism Another popular mechanism to answer numeric queries while preserving differential privacy is the Gaussian mechanism [38]. The idea is similar to the Laplace mechanism, but instead of adding Laplacian noise, the curator adds noise drawn from a Gaussian distribution. In the Gaussian mechanism, the amount of required noise for an arbitrary function $f : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^d$ is proportional to its ℓ_2 sensitivity:

$$\Delta_2 f = max_{x,y \in \mathcal{D} \| x-y \|_1 = 1} \| f(x) - f(y) \|_2$$

Definition 2.3 (Gaussian Mechanism) *Given any arbitrary function $f : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^d$ with an ℓ_2 sensitivity of $\Delta_2 f$, the Gaussian mechanism is defined as:*

$$\mathcal{M}_G(x, f(\cdot), \epsilon) = f(x) + (Y_1, \cdots, Y_d)$$

where Y_i are i.i.d random variables drawn from the Gaussian distribution $\mathcal{N}(0, \sigma^2)$.

The Gaussian distribution $\mathcal{N}(0, \sigma^2)$ (centered at 0) is defined by the probability density function:

$$p(x|\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(\frac{x^2}{2\sigma^2}\right)$$

One can show that the Gaussian mechanism is (ϵ, δ)-differential private for parameters $\sigma \geq c\Delta_2 f/\epsilon$ and $c^2 \geq 2\ln(1.25/\delta)$ [38].

Continual Observation The initial literature on differential privacy focuses on achieving privacy for queries running on a static database. However, this scenario is not directly applicable to achieving differential privacy on growing databases (i.e., streams of data). There are different notions of differential privacy on growing databases, and the definition of a mechanism changes accordingly [59].

Definition 2.4 *Let $\mathcal{M}$ be a mechanism that takes as input a stream prefix of arbitrary size. Let $\mathcal{O}$ be the set of all possible outputs of $\mathcal{M}$. Then, $\mathcal{M}$ is event-level (user-level) (ϵ, δ)-differentially private if for all sets $\mathcal{S} \subseteq \mathcal{O}$, all event-level (resp. user-level) adjacent stream prefixes x_t, y_t, and all t, it holds that:*

$$Pr[\mathcal{M}(x_t) \in \mathcal{S}] \leq e^{\epsilon} \cdot Pr[\mathcal{M}(y_t) \in \mathcal{S}] + \delta$$

For continual observations, there are also other notions of differential privacy, such as w-event-level differential privacy, which is a compromise between event-level and user-level differential privacy [59].

Without trusted Curator The standard model in differential privacy assumes a trusted curator who holds all the data in a database. The curator is responsible for performing the query and ensuring that the released result is differentially private (i.e., running the mechanism). However, such a trusted curator might often not be available or desirable in the distributed setting of the thesis. There are two approaches to remove the need for a trusted curator, which are demonstrated based on an example.

Consider the example of calculating a differentially private sum. The naive version to achieve local differential privacy without a trusted curator is that all clients add sufficient noise to their contributions before sending them to the aggregator. However, when summing over all N contributions, the amount of noise is N times as much compared to the standard-setting with a trusted curator. Consequently, this leads to a low utility of the query.

A better alternative for such a situation is to replace the trusted curator with a secure multiparty computation (MPC) protocol. The idea is that each of the N participants adds only $1/N$ of the required noise and then performs a secure aggregation protocol among all participants, which reveals only the sum of all contributions. Since the sum contains $N \cdot 1/N$ the required amount of noise, the sum is differentially private [12].

Note that this protocol relies on trust in other participants that they also add their part of the noise. In a distributed setting, this full trust is usually not the case. Instead, there is often an assumption on the fraction of trusted participants. This assumption allows calibrating the noise to account for some colluding participants.

Further, the protocol assumes that the required noise originates from a divisible distribution such that each participant can generate $1/N$ of the total noise. It turns out that both the Laplace and Gaussian distribution are divisible.

Theorem 2.5 (Divisibility of Laplace Distribution [60]) *Let $\mathcal{L}(b)$ denote a random variable drawn from $Lap(x|b)$. Then the distribution of $\mathcal{L}(b)$ is infinitely divisible. Furthermore, for every integer $n \geq 1$, $\mathcal{L}(b) = \sum_{i=1}^{n}[\mathcal{G}_1(n,b) - \mathcal{G}_2(n,b)]$, where $\mathcal{G}_1(n,b)$ and $\mathcal{G}_2(n,b)$ are i.i.d random variables having gamma distribution with PDF $g(x|n,b) = \frac{(1/b)^{1/n}}{\Gamma(1/n)} x^{\frac{1}{n}-1} e^{-x/b}$ where $b \geq 0$.*

Theorem 2.6 (Divisibility of Gaussian Distribution [65]) *Let Y denote a random variable drawn from $\mathcal{N}(0, \sigma^2)$. Then the distribution of Y is infinitely divisible. Furthermore, for every integer $n \geq 1$, $Y = \sum_{i=1}^{n} Y_i$, where Y_i is a i.i.d random variable $Y_i \sim \mathcal{N}(0, \frac{1}{n}\sigma^2)$.*

2.2 Secure Computation

The goal of secure computation is to compute on data without revealing private information. PolicyCrypt leverages these cryptographic techniques to ensure that a service cannot violate user-defined privacy policies.

This section starts by covering cryptographic primitives that enable secure computation on top of encrypted data. The introduction of cryptographic primitives should further help to understand the cryptographic components of PolicyCrypt.

Subsequently, the section discusses two forms of secure computation. First, *homomorphic encryption* which targets secure computation on an untrusted server and second, *multiparty computation* which targets secure computation among untrusted parties.

2.2.1 Cryptographic Primitives

The following paragraphs present a brief introduction of three fundamental building blocks of modern cryptography. The first two paragraphs cover *block ciphers* and *pseudo-random functions*, which are concepts from symmetric-key cryptography, the type of encryption that uses a single key to protect data confidentiality. The last paragraph introduces the *Diffie-Hellman key exchange*, which is a form of public-key cryptography that allows generating shared keys over an untrusted network.

Block Ciphers Block ciphers are one of the core components of symmetric cryptography. A block cipher is a deterministic algorithm to encrypt a block of data using a symmetric key. More formally, a block cipher is a deterministic cipher $\mathcal{E} = (E, D)$. The encryption algorithm E takes a message m and a key k as input and produces a ciphertext c as output. The decryption algorithm D reverses the encryption. It takes a ciphertext c and a key k as input and produces a message m as output. For block ciphers, the message space and the ciphertext space (i.e., where messages and ciphertext respectively lie) is the same (finite) set $\mathcal{X}$. To be more precise, for a fixed key k, the function $E(k, \cdot)$ is a permutation on $\mathcal{X}$ and the function $D(k, \cdot)$ is the inverse of the same permutation. Intuitively, the security of a block cipher relies on the property that for a randomly chosen key k, the permutation $E(k,\cdot)$ should be indistinguishable from a random permutation [21].

The most widely used block cipher is the Advanced Encryption Standard or short AES. It operates on 128-bit blocks and supports 128, 192, and 256-bit strings as keys. The design of AES is already quite efficient in software. Nevertheless, all major processor vendors provide a hardware implementation to speed-up and simplify the use of AES considerably. For example, Intel extended their instruction set with "AES new instructions" (AES-NI) [21].

Pseudo-random Functions A pseudo-random function (or PRF) is a closely related concept to block ciphers. PRFs are conceptually simpler objects than block ciphers, and it even turns out that secure block ciphers are useable as a stand-in for secure pseudo-random functions (under certain assumptions) [21]. This property is useful because it implies that efficient AES can function as a PRF. More formally a pseudo-random function F is a deterministic algorithm that has two inputs: a key $k \in \mathcal{K}$ and an input data block $x \in \mathcal{X}$; its output $y = F(k,x)$ with $y \in \mathcal{Y}$ is called output data block. Intuitively, for a randomly chosen key k, the function $F(k,\cdot)$ should look like a random function from $\mathcal{X}$ to $\mathcal{Y}$ [21].

Diffie-Hellman Key Exchange The Diffie-Hellman key exchange is a form of public-key cryptography that allows two parties to establish a secret over an insecure channel [34]. Consider an example of two parties called Alice and Bob that want to exchange a secret message. The area of symmetric cryptography provides algorithms such as block ciphers that allow Alice and Bob to achieve their goal as long as they already have a shared secret key. However, Alice and Bob can only communicate over an insecure channel. This limitation raises the question of how they can establish such a shared key? In 1976, Diffie and Hellman answered this question by presenting a protocol which allows Alice and Bob to exchange cryptographic keys over an untrusted network. The remaining part of this paragraph describes their protocol more formally and introduces the most suitable form for applications.

Let $\mathbb{G}$ be a finite cyclic group of order n and a generator $g \in \mathbb{G}$, which are both public knowledge.

1. Alice chooses a random value a (i.e. Alice's private-key), where $1 \leq a \leq n$, and sends g^a (i.e. Alice's public-key) to Bob.

2. Bob chooses a random value b (i.e. Bob's private-key), where $1 \leq b \leq n$, and sends g^b (i.e. Bob's public-key) to Alice.

3. Alice computes the shared secret key $g^{ab} = (g^b)^a$ using Bob's public-key and her private-key.

4. Bob computes the shared secret key $g^{ab} = (g^a)^b$ using Alice's public-key and his private-key.

The security of the Diffie-Hellman key exchange relies on the assumption that the discrete logarithm is hard to compute in the group $\mathbb{G}$. In other words, it must be hard to compute the private-key a from the public-key g^a. It turns out that the group of points of an elliptic curve over a finite field is the most suitable in practice. The main reason for this is that the best known discrete-log algorithm in an elliptic curve group of size n runs in time $O(\sqrt{n})$. As a consequence, to provide similar security as AES-128, a group size of $n \approx 2^{256}$ is sufficient [21].

An elliptic curve E defined over $\mathbb{F}_p$ for a prime $p \geq 3$ is given by the equation:

$$y^2 = x^3 + ax + b$$

Let $E/\mathbb{F}_p$ be an elliptic curve and let $E(\mathbb{F}_{p^e})$ be the group of points on this curve. Given a secret integer $\alpha \in \mathbb{Z}_q$ (i.e. private-key) and a public generator point P, the resulting point $Q = \alpha P$ is the public-key [21].

2.2.2 Homomorphic Encryption

After introducing the most important cryptographic primitives in the previous section, this section considers the first form of secure computation. In 1978 Rivest, Adleman, and Dertouzos proposed the concept of homomorphic encryption [82]. The idea of homomorphic encryption is to enable computation on ciphertexts (i.e., encrypted data). Since data remains always encrypted, the computation can run in an untrusted environment without compromising confidentiality.

A homomorphic encryption scheme allows performing arbitrary arithmetic operations on ciphertexts. Let $\mathcal{E} = (E, D)$ denote a probabilistic encryption scheme over $(\mathcal{M}, \mathcal{C}, \mathcal{K})$ such that the message space $\mathcal{M}$ and the ciphertext space $\mathcal{C}$ are groups under operations $\oplus$ and $\otimes$ respectively. In a $(\oplus, \otimes)$-homomorphic encryption scheme $\mathcal{E}$, it holds that for any two ciphertexts $c_1 = E(k_1, m_1)$ and $c_2 = E(k_2, m_2)$, there is a key $k \in \mathcal{K}$ in the key space such that:

$$c_1 \otimes c_2 = E(k,\, m_1 \oplus m_2)$$

In other words, there is a way to perform computation on ciphertexts before decrypting, which leads to the same result as performing the computation on plaintext [26].

The literature distinguishes between fully homomorphic encryption schemes (FHE) [22,50] which support arbitrary computation, and partially homomorphic encryption schemes (PHE) [40,51,77,83] which support only a subset of the possible computation. Even though there is significant progress in FHE, the cost is still prohibitive for the streaming scenario considered in the thesis [29,37]. As a consequence, PolicyCrypt resorts to an efficient partially homomorphic encryption scheme, based on simple modular addition, to cope with the considered workloads.

Modulo Addition-based Encryption Modulo addition-based encryption introduced by Castelluccia et al. is a simple additive homomorphic encryption scheme relying only on lightweight symmetric cryptography [26].

A message m is encrypted with a fresh key k and a predefined modulus M:

$$c = E(m,\, k, M) = m + k \mod M$$

Both the message m and the key k are integers in the range $[0, M-1]$.

To decrypt the ciphertext c subtract the key k:

$$D(c,\, k, M) = c - k \mod M$$

The scheme is semantically secure if a new, unique cryptographic key is used for each encryption [26]. Let $c_1 = E(m_1, k_1, M)$ and $c_2 = E(m_2, k_2, M)$, the encryption scheme is additively homomorphic because for $k = k_1 + k_2$ we have that:

$$D(c_1 + c_2,\, k, M) = m_1 + m_2 \mod M$$

2.2.3 Multiparty Computation

Another form of secure computation is multiparty computation (MPC). A secure MPC protocol enables a group of participants with secret data, to perform a computation on the data without disclosing their individual inputs [43].

More formally, a fixed set of participants $p_1, p_2, \cdots, p_n$ agree on a function f. Afterward, the participants perform an MPC protocol to compute the output of the function together without anyone revealing their secret inputs $m_1, m_2, \ldots, m_n$.

Secure Aggregation One possible function is to calculate the sum over private inputs $m_1, m_2, \ldots, m_n$:

$$f(m_1, m_2, \cdots m_n) := \sum_{i=1}^{n} m_i$$

This problem is also known as secure aggregation and has a wide range of applications. For example, in privacy-preserving machine learning [20] or in the differential privacy model without a trusted curator [38].

PolicyCrypt leverages secure aggregation based on symmetric additive homomorphic encryption to enforce aggregation-based privacy policies. In this form of secure aggregation, each contribution is masked with a nonce [12, 20].

The participants agree pairwise on a random nonce d_{uv}. If u adds this nonce to m_u and v subtracts it from m_v, then the mask d_{uv} cancels out when adding up the two inputs. As a result, the sum $m_u + m_v$ is available while the individual contributions m_u and m_v remain hidden. For a larger group of participants, each user u computes:

$$c_u = \; m_u \; + \sum_{v \in U:\; u > v} d_{uv} \; - \sum_{v \in U:\; u < v} d_{uv} \mod M$$

The aggregator then sums up the individual contributions c_u, which results in the pairwise nonces canceling out. As a result, only the sum of actual inputs remains:

$$\begin{aligned} z &= \sum_{u \in U} c_u \mod M \\ &= \sum_{u \in U} \left(m_u + \sum_{v \in U:\ u>v} d_{uv} - \sum_{v \in U:\ u<v} d_{uv} \right) \mod M \\ &= \sum_{u \in U} m_u \mod M \end{aligned}$$

The random masks can be defined as $d_{uv} := PRF(k_{uv}, r)$, where r is a changing public value, and k_{uv} is a shared key between u and v which can be established through a Diffie-Hellman key exchange protocol.

2.3 Data Stream Analytics

The privacy platform PolicyCrypt targets the scenario of data stream analytics, which is about gaining insight in real-time on data in motion. The first part of this section introduces the streaming data model. Then the chapter proceeds to discuss techniques and specialties when processing data within this model. Finally, the last part is dedicated to existing streaming frameworks.

2.3.1 Data Streams

The idea of data stream analytics is to treat data as an unbounded stream of events over time. An event (or record) usually consists of a data tuple and a timestamp. The data tuple can be of arbitrary form and contain one or more elements, while the timestamp typically defines an ordering of a stream.

In this model, data is usually generated by distributed data sources that can send a new event at any time. As a result, the data streams are often highly irregular, which means that a burst of events can follow a more extended period without an event. For example, think of mobile phones as data sources while passing through a tunnel without connection. After the connection is re-established, the mobile adds a burst of events to the data stream.

The streaming model is widely applicable because almost all created data is a continuous stream of events. For example, user click actions on websites, server logs, messages on a social network, or sensor measurements. In reality, it is hard to find use cases where datasets are generated all at once [56]. After introducing the data model, the next part looks at what makes processing a data stream unique.

2.3.2 Stream Processing

Stream processing is a programming paradigm for performing computation on data in motion. In the classical paradigm of data processing, data is stored in a database, and queries run against this fixed dataset. Stream processing turns this classical paradigm around. The queries are static (i.e., fixed), and data flows through them immediately when it arrives [91].

Most stream processing pipelines share a similar structure. They compose of three components. First, a set of distributed *data producers* that generate streams of data. Second, a *broker system* that offers a publish-subscribe messaging pattern to access the streams. Additionally, the broker system is also responsible for storing the records in a fault-tolerant and reliable way [1]. Then finally, one or multiple *stream processors* running the predefined queries. The output of the queries is once more a new data stream containing the results. Figure 2.1 shows such a typical streaming pipeline in combination with popular systems for the different components.

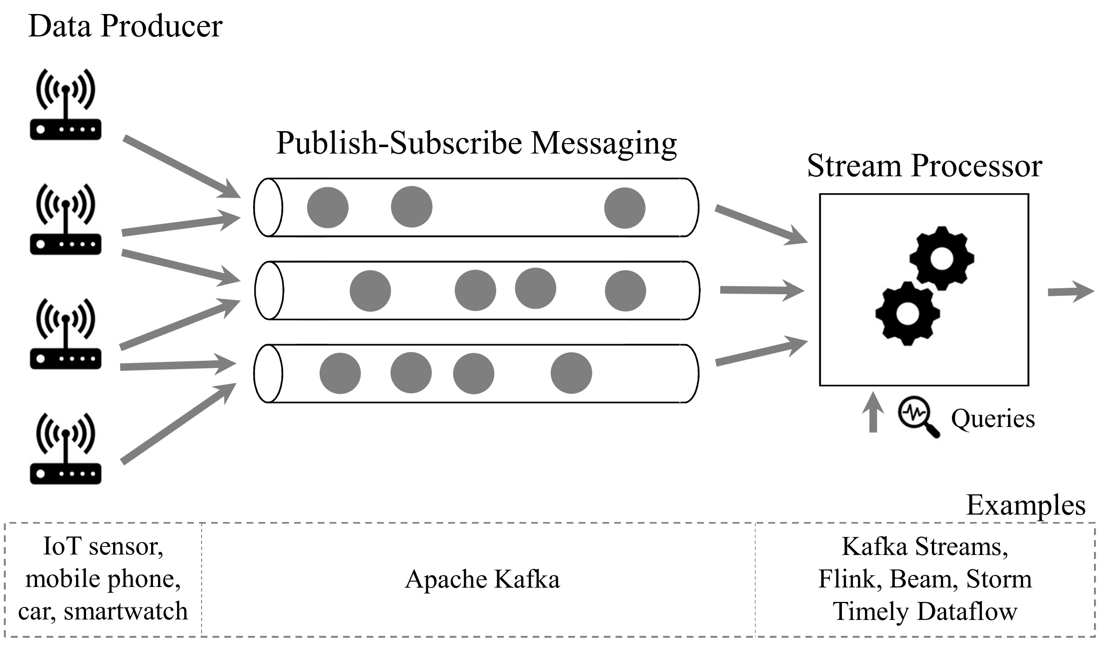

Data Producer
Publish-Subscribe Messaging
Stream Processor
Queries
Examples
IoT sensor, mobile phone, car, smartwatch
Apache Kafka
Kafka Streams, Flink, Beam, Storm Timely Dataflow

Figure 2.1: Typical streaming pipeline with data producers, an intermediate broker system which offers publish- and subscribe access to event streams, and a stream processor running the static queries.

The following paragraphs discuss the concepts of time and windows that have a central role in streaming processing.

Notion of Time As mentioned above, stream processing abstracts data as an unbounded stream of timestamped data tuples (i.e., records). There are two major categories for the meaning of such a timestamp. A timestamp can refer to when the event occurred (i.e., event-time) or when the record was processed (i.e., processing-time).

Using event-time requires that the data source attaches a timestamp to each record. Apart from the marginally increased event size, using event-time brings the additional challenge of dealing with out-of-order events. Assigning a timestamp at the source can lead to out-of-order events at the stream processor because the clocks of the data sources are not synchronized, and events incur different transportation delays from the source [1].

Window Operators Window operations are among the most common operations in stream processing because they enable transformations on bounded intervals of an unbounded stream. Window operations continuously create finite sets of events called buckets from an unbounded event stream and then allow to perform computations on these finite sets [56]. Figure 2.2 shows two of the most common window types, tumbling- and sliding windows (or also called hopping-windows). For both of them, the stream processor assigns events to buckets of fixed size based on time.

Tumbling windows assign events into non-overlapping buckets while sliding windows assign events into overlapping buckets of fixed size. Consequently, an event might belong to more than one bucket in a sliding window [56]. Both types of time windows usually align with the Unix epoch. As a result, a length parameter uniquely defines a series of tumbling windows. Sliding windows require both a length and a slide parameter. A minor detail is that the window start timestamp is inclusive and the end timestamp (i.e., which, in turn, is the start timestamp of the next window in a tumbling window) is exclusive.

In order to handle out-of-order events in time windows, stream processors use concepts such as watermarks or a grace period [1, 56]. In essence, all these concepts have an increasing stream time based on the observed timestamps. Late arriving events are accepted if they arrive within a predefined grace period or as long as they arrive before a watermark, which indicates that no earlier events can occur.

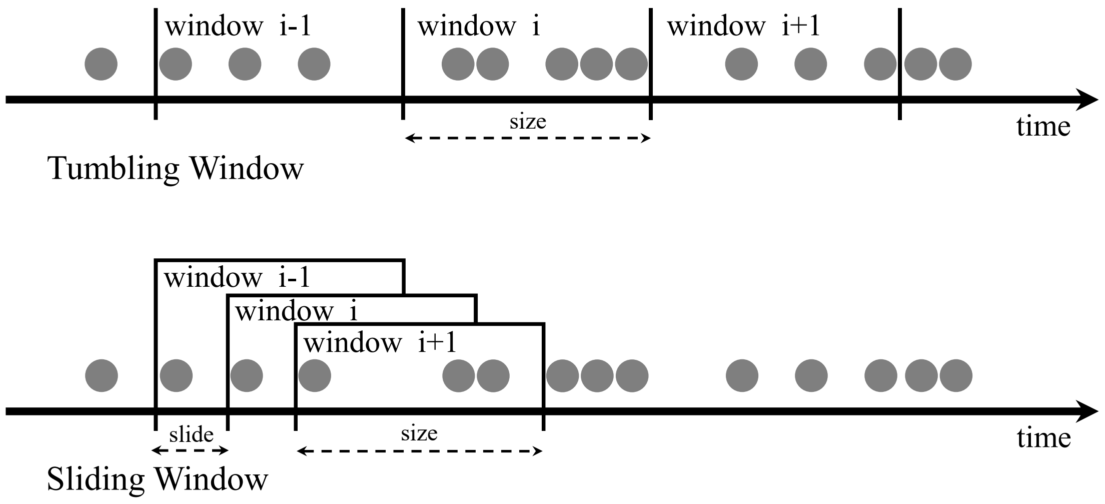


Figure 2.2: Schematic of tumbling- and sliding windows.

2.3.3 Streaming Framework

The central component in many stream processing pipelines is Kafka [2] and there is a range of different stream processor frameworks such as Flink [45], Beam [87], Storm [47], Kafka Streams [46], and Timely Dataflow [69]. However, most of them directly integrate with Kafka and use it as the source of data streams.

The prototype of the privacy platform PolicyCrypt also directly integrates and operates with Kafka, which makes it compatible with many of the existing stream processing pipelines. The next paragraph introduces the main concepts of the Kafka ecosystem for later reference.

Kafka Kafka enables a publish and subscribe messaging pattern while storing data streams in a fault-tolerant commit log [1]. Usually, Kafka operates on a cluster with multiple servers (called brokers) that run in different data centers.

The core abstraction for streams in Kafka is topics, and a cluster can contain an arbitrary number of topics. Each topic consists of one or more partitions distributed among the available brokers. These partitions are the critical factor that makes Kafka suitable for large volume streams generated at a high velocity. In Kafka, each record consists of a key, a value, and a timestamp. The record key determines the partition within a topic, and the value is an arbitrary data tuple. However, often the value references a data schema that allows a stream processor to deserialize a record into a more suitable form.

The Kafka ecosystem contains four core interfaces to Kafka: the Producer API which enables clients to write (publish) records into topics, the Consumer API which enables clients to subscribe to topics, the Streams API which is a stream processor and the Connector API which connects Kafka to external systems such as databases [1].

Kafka Streams Since the prototype of PolicyCrypt uses Kafka Streams for performing privacy transformations, this paragraph provides a brief introduction to this data stream processing library.

Kafka Streams directly integrates with the Kafka core and reuses many concepts. A stream processing query is defined via the domain-specific language (DSL) or the low-level processor API of Kafka Streams. Furthermore, the two APIs can also be used together to define a stream processor topology. The workload of a single query divides into tasks, similar to the concept of topic partitions. As a consequence, a Kafka Streams application consists of a set of streaming tasks that process different partitions of the same data stream. These streaming tasks are distributed over multi-threaded instances of the same application that can run on different machines. To enable stateful stream processing, Kafka Streams provides state stores that allow applications to store and query data.

On top of Kafka Streams, the ecosystem also contains KSQL, a streaming engine that enables a wide range of applications based on a simple query language similar to SQL [57].

3 Related Work

This work presents a new system design for a privacy platform that enables the safe sharing of data while complying with users' privacy preferences in an end-to-end encryption setting. To our knowledge, this is the first system design that explores the enforcement of privacy policies with cryptography. Before describing the system design in the next chapter, this chapter discusses related work that targets aspects close to this work.

3.1 Privacy Policy Enforcement

Most work related to enforcement of privacy policies is manual and focuses on compliance. Few recent works in academia and start-ups aim to automate enforcement. These systems, however, either rely on a trusted service or trusted hardware [4, 17, 67, 93]. Hence, these approaches do not comply with end-to-end encryption. This section introduces three related systems in more detail and compares them to PolicyCrypt.

- *Privitar:* The start-up Privitar provides a privacy management platform to de-identify data and achieve regulatory compliance while permitting advanced analytics to extract maximum value from the data [4]. However, in contrast to this work, the platform applies privacy transformations with access to raw data. While this approach helps with regulatory compliance, from the user viewpoint, the enforcement of privacy policies still relies on trust rather than cryptographic guarantees.

- *Riverbed:* The framework Riverbed permits building privacy-respecting web services [93]. Users can define policies that specify whether aggregation and persistent storage are possible, and a user can define a network whitelist for communication. A client runs a transparent Riverbed proxy between the user front-end and the server. The server has a trusted platform module and runs software in a Riverbed compatible runtime (e.g., adjusted JVM, which enforces the policies). The server attests to the proxy that the code respects the policies by showing a proof that the code runs in such a Riverbed compatible runtime using the Cobweb attestation system [92]. In other words, trusted hardware automatically enforces privacy policies.

  The approach of Riverbed requires that all applications run in a Riverbed compatible runtime. This approach works well for languages that are compiled to bytecode and then interpreted by a runtime (e.g. Java, Python). However, the approach does not directly translate to languages that compile to machine code (e.g. C, Rust). As a consequence, the approach does not integrate well into the existing system landscape.

- *Towards Multiverse Databases:* This work explores how to apply access policies in a web application backend database [67]. A single database called base universe stores all the data. Each logged-in user receives a universe on the server, derived from the base universe using a stateful dataflow computation. The nodes in this dataflow computation apply the defined policies such that each user's universe is a policy-compatible materialized view of the database. The server runs all queries from users against their universe. This method ensures that a user only sees data that complies with the policy.

  Multiverse databases allow a service to define policies that are later automatically enforced. This form of enforcement improves the status quo of manual enforcement and simplifies compliance. The automatic enforcement is only possible because of a policy language that allows expressing privacy transformations in a structured form and due to the central storage of all data. However, in comparison to this work, Multiverse databases assume a fully trusted service that automatically enforces the defined policies.

3.2 Cryptographic Access-Control

Storing data in the cloud using end-to-end encryption gives the best level of protection because data remains always encrypted. As a consequence, only authorized entities with encryption keys have access. However, defining fine-grained access to data poses a challenge in the end-to-end encryption scenario.

Among other techniques, research proposed identity-based encryption (IBE), attribute-based encryption (ABE), predicate encryption, or functional encryption to address these challenges. The main idea in all these schemes is that a third party can only decrypt a subset of the data. However, all these schemes operate in a tradeoff between expressiveness and performance. Recent systems implement different flavours of encrypted access-control [49, 52, 61, 85, 94, 95]. Among those systems, three examples are presented in more detail below.

- *Jedi:* This is a system that defines an encryption scheme that allows many-to-many end-to-end encryption on resource hierarchies [61]. The focus lies on a system that works on minimal hardware with severe battery constraints. The system decouples senders from receivers by supporting a publish-subscribe messaging pattern. Jedi supports the expiration of access by treating time as a hierarchy and implements a form of revocation. The cryptography bases on WKD-IBE, which is a form of attribute-based encryption.

- *Sieve:* This work presents a system for the central storage of encrypted user data in an untrusted cloud [94]. The user can define cryptographic access policies for web services that want to use the data. A policy consists of metadata attributes and a symmetric key, both encrypted with attribute-based encryption (ABE). The metadata attributes define who has access to the data, and the symmetric key is required to decrypt the object itself. The system implements access revocation via a key homomorphism but requires an honest server at the time of rekeying.

- *Bolt:* This work introduces a time series database that stores consecutive records in encrypted chunks on an untrusted server [54]. Each record has a timestamp and one or more tag-value pairs attached, which allows querying the data. The time series data is encrypted using a hash-based key regression, which, given a key, permits deriving all previous keys. As a result, the only supported access policies define that services can access all data up to a certain point in time. Another drawback of Bolt is that it relies on a trusted key management server.

The area of cryptographic access-control is relevant for this work because it considers methods to express policies and deal with encrypted data in untrusted cloud environments. However, in cryptographic access-control, the untrusted cloud can only be used for storage. This work requires that the untrusted server can also perform computation in an end-to-end encryption scenario.

3.3 Encrypted Data Processing Systems

The intention behind encrypted data processing is to perform queries on data that remains encrypted. Such an encrypted database could run in an untrusted cloud environment while preserving both confidentiality and functionality. Over the last decade, many designs of encrypted data-processing systems emerged. They adopt a similar approach but target different scenarios: time series data [23], batch analytics [79], graph databases [71], key-value stores [24], and relational databases [80, 81, 89]. The remainder of this section discusses three of those systems and highlights why encrypted data-processing is not sufficient for this work.

- *CryptDB:* This work is an SQL database which runs queries on encrypted data [81]. CryptDB encrypts columns with an onion encryption scheme. Initially, all columns are encrypted with random encryption. However, depending on the queries running within the database, the system strips some layers. For example, to run a join query on a column, the system needs to strip all layers down to a deterministic encryption layer. However, recent work highlighted possible leakage attacks on the deterministic and order-preserving encryption layers, which show that the system is not secure [53].

- *Seabed:* The system Seabed uses symmetric additive homomorphic encryption to enable efficient analytics over encrypted large-scale datasets [79]. Clients write encrypted data to the server using different encryption schemes based on the supported queries. A user supplies a query to execute on top of encrypted data. After the computation, the user receives the result, which is still encrypted. To decrypt the result, a user requires access to the corresponding keys. The system leverages a key canceling technique to support efficient in-range queries on large datasets. Additionally, the authors propose Splashe, which is an encryption scheme that protects against frequency-based attacks and enables the system to support group by sum and count queries.

- *TimeCrypt:* This work targets encrypted data-processing of time series data. A data producer writes encrypted digests of time series data to an untrusted server. The additively homomorphic encryption scheme, in combination with an aggregation-based encoding, allows the server to compute statistical queries while data remains encrypted. Due to a key canceling technique, an authorized data consumer can decrypt the result of in-range aggregates efficiently. Further, the work also provides cryptographic access-control that permits sharing arbitrary intervals of the time series data or restrict access to a lower resolution. Among all encrypted data-processing systems, TimeCrypt is the most similar system to this work because of a related scenario and encryption scheme. However, TimeCrypt does not have support for irregularly spaced data streams and lacks methods to compute aggregate functions across data streams.

For this work, the research on encrypted data-processing systems is relevant because both tackle the problem of performing the computation in an untrusted cloud environment. However, these systems do not support the required functionality of a privacy platform. There is no support to perform privacy policy compliant computation over a set of users with different privacy policies. Furthermore, many of the systems target the more traditional database paradigm instead of streaming data.

3.4 Private Statistics

Research addressed the challenge of collecting and handling private data in three different flavors. Some systems collect private data only in aggregated form using techniques from secure multiparty computation. Other systems only collect differentially private data. Finally, some systems combine the two ideas and collect differentially private statistics of aggregated data.

The area of private statistic collection is relevant for this work because it introduces techniques to facilitate privacy-preserving transformations on data. However, no existing system fulfills the requirement to support aggregation across data streams with heterogeneous user-defined privacy policies.

3.4.1 Multiparty Computation

- *Prio:* This work provides a scalable system for collecting aggregate statistics using multiple servers [32]. Prio leverages secure multiparty computation techniques to preserve the client's privacy as long as there is at least one honest server. Clients split their contribution into shares, which they submit to different servers. The servers aggregate shares from different clients before combining their aggregate with the result from the other servers. Additionally, Prio also includes protection against malicious clients via secret-shared non-interactive proofs (SNIPs). There are many more systems that target the space of private aggregate statistics [26,33,36,39,58,63,70].

- *Practical Secure Aggregation for Privacy-Preserving Machine Learning:* This is a paper that introduces a secure aggregation scheme for scenarios with large vectors of values and frequently dropping clients [20]. The scheme's primary application area is in federated machine learning, where the gradient updates can contain millions of parameters. Users share their secret across all other users using Shamir's Secret Sharing. Consequently, the protocol is robust as long as t out of n users do not drop out. However, this comes at a high communication cost to distribute the secret shares. The encryption is similar to Dream [12]; each user adds $n-1$ shared masks to the message that cancel out. An additional nonce protects the contribution of the user in case of a dropout. In further rounds, the remaining users both lift the additional nonces and the shares of dropped users. In the end, the server learns nothing except the sum of the contributions of the remaining users. For the scenario of streaming data with comparably small records, the overhead of continuously distributing the secret shares is too large compared to the benefits.

3.4.2 Differential Privacy

- *Sage:* The differentially private machine learning platform [64] proposes a new accounting method to keep track of the (ϵ, δ) privacy budget while training models under continual observations. The platform, run by a trusted curator, retires training examples with an exhausted privacy budget. Additionally, Sage introduces an idea that addresses the privacy-utility tradeoff (i.e., how to detect that a model trained under differential privacy well enough).

- *PeGaSus:* This is a system that provides differential privacy for streams of count data [27]. A trusted curator operates three main components. A *perturber* adds Laplacian noise to raw counts, a *grouper* partitions the stream into continuous, uniform segments using the sparse vector technique [38] and a *smoother* which combines the outputs from the grouper and perturber to improve utility on the results. The drawback of the system is that it only works on simple and stable counts and relies on a trusted curator.

- *Rappor:* This work provides differential privacy in the local model over multiple reports [42]. It builds on ideas from Randomized Response [38]. However, Rappor memorizes the random answers which it has already given to guarantee resilience against an attacker which can see multiple reports. The server uses lasso regression to learn aggregated statistics on the server.

3.4.3 Differential Privacy with Secure Multiparty Computation

- *Dream:* The system Dream collects differentially private data from smart meters [12]. From all methods to collect private statistics, Dream is the closest to this work because of a similar secure aggregation scheme. However, Dream does not allow users to express preferences in the form of privacy policies. Further, Dream does not optimize for repeated transformations on data streams, and limiting access to a lower resolution is also not supported.
- *Privacy-Preserving Aggregation of Time-Series Data:* This work provides distributed differential privacy (i.e., without a trusted curator) for time series data [88]. It leverages an additively homomorphic, Diffie-Hellman-based encryption scheme which relies on solving the discrete logarithm problem. This form of decryption is computationally expensive and only feasible for small plaintext space. Furthermore, the authors propose to use additive noise from a Geometric distribution, which has the advantage that it provides discrete noise.

4 Overview and Objectives

This chapter starts by giving an overview of the PolicyCrypt platform. Then, a section describes the capabilities of an attacker in the considered scenario with a threat model, before the next section presents meaningful privacy policies for the scenario. The chapter concludes with challenges for the design of the privacy platform PolicyCrypt.

4.1 Overview

PolicyCrypt is a platform that facilitates privacy-preserving transformations between user data streams and an untrusted service provider. Figure 4.1 shows the high-level system components of PolicyCrypt. A user is in control of one or more trusted *data producers* and a trusted *privacy controller*. The untrusted service provider is in control of a *policy manager* and a *data transformer*.

The central idea of PolicyCrypt is to put users in charge of defining privacy policies and enforce their privacy rules and preferences via encryption. The system design decouples the data plane from the privacy plane, to support sophisticated and heterogeneous privacy policies. The privacy plane is responsible for organizing and orchestrating privacy-preserving transformations while the data plane focuses only on end-to-end encrypted data streams. Instead of encrypting data towards a privacy policy (i.e., encrypt data such that a specific privacy-preserving transformation can be applied to the data). PolicyCrypt encrypts data such that various privacy-preserving transformations can be applied to the encrypted data by combining encrypted data with cryptographic transformation tokens. This way, data sources do not need to manipulate keys, cryptographic primitives, or use an additional encryption scheme other than the one used to ensure data confidentiality. As a result, encrypted data can flow as before to be stored in encrypted cloud storage.

Data Plane PolicyCrypt continuously processes data that originates from different data producers. As mentioned above, users are in control of data producers, whereby a user can manage multiple data producers at the same time. Data producers interact with a client-side proxy that encrypts the data stream before pushing it to a remote streaming platform for storage. Note that this design follows the end-to-end encryption paradigm because the remote streaming platform only observes encrypted data streams. At a later stage, a user or another authorized client can consume the encrypted data stream and decrypt it locally at the end-user application. The design corresponds roughly to the design of TimeCrypt and hence supports the

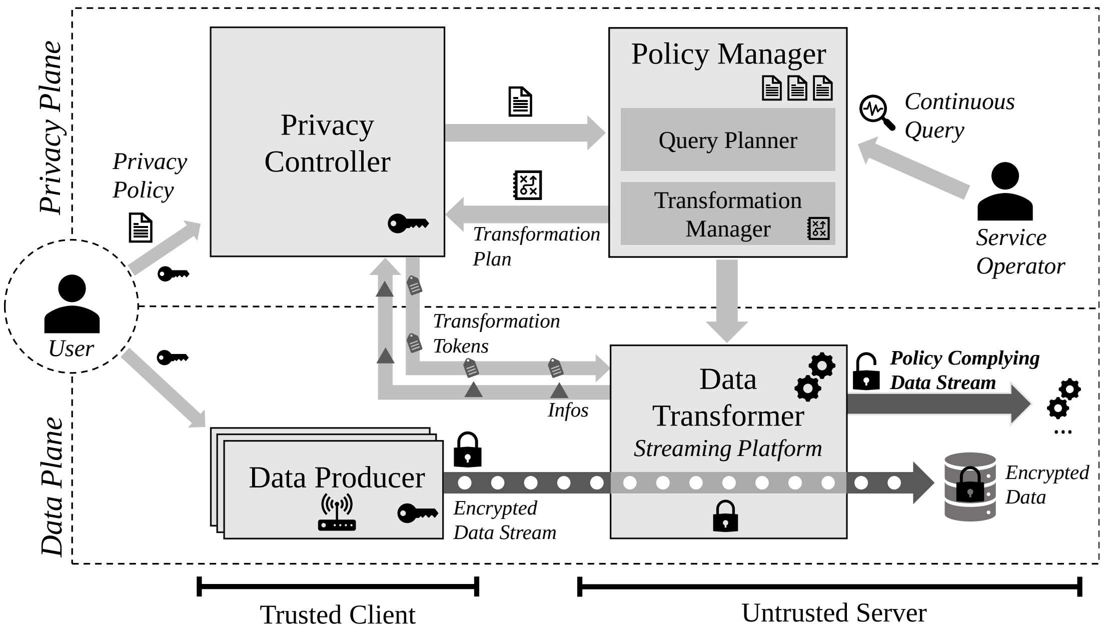


Figure 4.1: PolicyCrypt system architecture overview.

same applications [23]. However, to enable an even broader set of applications, PolicyCrypt introduces a privacy plane that allows the platform to perform privacy-preserving transformations on top of the encrypted data streams. The idea behind privacy-preserving transformations is that a service provider can utilize the private user data streams as long as the privacy preferences of the corresponding users are respected.

Privacy Plane In PolicyCrypt, the privacy plane consists of a privacy controller that represents the privacy interests of the user and a policy manager that administrates the streaming queries of the service provider. Users define privacy policies through a simple, understandable policy language that describes acceptable data transformations. For example, “this stream must be aggregated with streams from other users”, or “this stream can only be accessed at a 12-hour resolution”. The policy manager of the service provider collects and manages these privacy policies.

A service provider specifies a streaming query in the policy manager that requests a particular transformation of data streams based on their metadata (e.g. type of data, description of the data source). The query planner of the policy manager thereupon creates a transformation plan by matching the streaming query to available streams that have compatible privacy policies.

A transformation plan defines a privacy transformation over a universe of streams with compatible privacy policies. In PolicyCrypt, a privacy transformation is a recurring computation performed on the same universe of data streams (i.e., a stream processing computation). Figure 4.2 sketches the relation between transformation plan, privacy transformation, and universe in PolicyCrypt. Existing transformation plans are executed by the data transformer that transforms private data streams in real-time without access to the underlying raw data items. PolicyCrypt relies on cryptographic transformation tokens that allow the data transformer to operate on the private data streams without decrypting them. These tokens inherently enforce compliance with per-stream privacy policies.

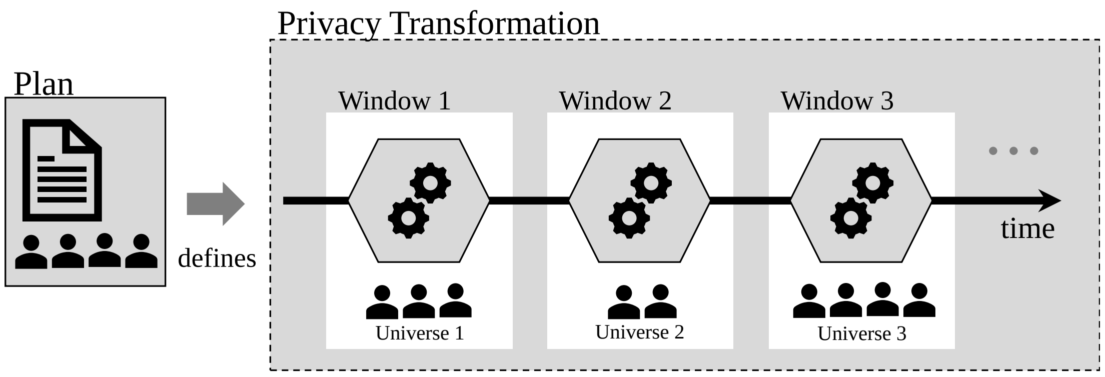


Figure 4.2: A transformation plan defines a privacy transformation that performs a recurring computation over an adapting universe of data streams.

Each user of PolicyCrypt operates a privacy controller, which continuously produces these transformation tokens based on the user-defined privacy policy and the transformation plan from the policy manager. The privacy controller is an essential component of PolicyCrypt, as it enforces cryptographically that the remote service must comply with the user-defined privacy policies.

The separation between data and privacy planes implicates that the privacy controller does not have access to data for generating these transformation tokens. Instead, the privacy controller and the data producer share only a secret master key. The data producer and the privacy controller use this master key to derive keys for encryption and to create corresponding transformation tokens, respectively. For transformations that span across multiple streams from different users, the responsible privacy controllers participate in a lightweight MPC protocol to create the respective transformation tokens. In PolicyCrypt, a privacy controller can be located in the data producer, in an IoT gateway, in a local server, or even in a trusted cloud service.

Applications The data transformer transforms the private streams in real-time according to the transformation plan with the help of the incoming streams of transformation tokens. The resulting transformed streams comply with all user-defined policies and hence can be used without restrictions in further and more complex stream processing pipelines to enable various applications. For example, a health application might provide a comparison of the individual heart rate of a user during a work-out session to the average heart rate of a group of users that share a similar training goal. While all users can access the public information of the average heart rate of the group, only users with the required keys can access their heart rate.

4.2 Threat Model

The data producer and privacy controllers are controlled by a fully trusted user in PolicyCrypt, while the policy manager and the data transformer are part of an untrusted server.

The server follows the *Honest-but-Curious* security model. Hence, the server must follow the protocol but can analyze all observed data to learn something which violates the user-defined privacy policy.

This work assumes a public-key infrastructure (PKI) for authenticating users participating in a privacy transformation. The PKI associates a public-key with each identity for establishing pairwise shared secrets using the Diffie-Hellman key exchange. Essentially, this means that an attacker can neither impersonate nor simulate arbitrary parties.

Note that even though a user is trusted in PolicyCrypt, this does not mean that users can violate the privacy policies of other users. More specifically, this work assumes that the number of colluding parties is bounded by a constant. This assumption allows for expressing guarantees about the number of non-colluding users involved in an aggregation.

This work does not hide the message timestamps of the stream. As a result of this leakage, an attacker might be able to infer information about an individual data stream. However, data producers can hide the time-pattern of their data by randomly submitting neutral messages, which provides a tradeoff between mitigating this leakage at the cost of more bandwidth. Moreover, PolicyCrypt currently does not defend against side-channel attacks such as volume and pattern leakage. However, in theory, PolicyCrypt could be complemented with known techniques to mitigate this form of leakage.

4.3 Privacy Policies

Sections 2.1 and 3.1 discussed the status quo of privacy policies and existing work on automatically enforcing compliance. This section outlines a list of ideas for meaningful privacy policies in the considered scenario and describes what capabilities privacy policies should support.

4.3.1 Types

The most straightforward privacy policy defines data as either public (i.e., usable without any restrictions) or private (i.e., not usable at all). While these two policies should ideally always be supported, there are many possibilities in between which preserve the user's subjective expectation of privacy while providing utility for a service at the same time.

The provided list below groups presented privacy policies into three categories. First, *aggregation policies* define that a service provider can only use data in an aggregated form across multiple sources or time. Second, *Differential Privacy policies* gives the user the ability to express that they want to disclose only differentially private data, which is required to achieve formal privacy guarantees. Finally, *constrained policies* reveal data only if the data meets specific criteria.

Aggregation Policies

- *Resolution policies* limit the access to a lower resolution of the data. For example, this could be a daily resolution of the number of steps or a monthly resolution of the amount of consumed electricity in a household for billing. Another form of thinking about data resolution is to divide the time axis into fixed-sized, non-overlapping windows, and data is disclosed only aggregated across such a window.

- *Multi-source policies* reveal data only in a combined form with data from other sources. For example, this could be that exam grades are only published once aggregated across the whole class. Even though these aggregated results do not provide a formal privacy guarantee, they give the user increasing practical privacy in the number of involved data sources. This form of privacy is similar to voting, where a single voter at least has plausible deniability.

- *Multi-source resolution policies* combine the ideas from *resolution-* and *multi-source policies.* Such a policy discloses data only in an aggregated form across multiple-sources at a certain maximum resolution. An example is that a user might be willing to share the number of steps at an hourly resolution aggregated across 100 people.

Differential Privacy Policies

To achieve formal privacy guarantees, a user requires privacy policies based on differential privacy. As discussed in Section 2.1, for continual observations there are different notions of differential privacy. As a result, a differential privacy policy defines both the notion of differential privacy and the (ϵ, δ)-parameters.

- *Event-level Differential Privacy* assumes independence between events from a single source. This form of differential privacy is the simplest but also with the weakest guarantees because of the strong and often unrealistic independence assumption.

- *User-level Differential Privacy* assumes that events from a single source are dependent. This notion is the strongest form of differential privacy under continual observations. However, over a long time, it requires to continually increase the noise to preserve privacy at the cost of utility.

- *W-Event-level Differential Privacy* assumes that events from a single source are dependent for a certain period of time. For $W = 1$ and $W = \infty$ it is equivalent to *Event-level* and User-level Differential Privacy respectively.

Constrained Policies

- *Value range policies* specify that the precise value can only be accessed if it is within a predefined range. Consider a geofence example for a company with a large warehouse. Workers might be willing to share their live location as long as they are within the warehouse. However, outside of the warehouse, the location should remain private.
- *Granularity policies* restrict access to raw data and instead only provide a class or a category. An example of such a policy could be that users are willing to share that they are within a specific city or in a particular street but no further details.
- *Equality policies* permit to identify if two values from two different sources are identical but not the actual value. For example, notify friends that they are currently at the same location (e.g. sitting in the same train)
- *Single-source group by policies* are a mixture between *constrained* and *aggregation policies*. The idea is to support aggregation groups depending on other fields of the event. An example would be that a user is willing to share the average heart rate while running at different speeds.
- *Multi-source group by policies* are similar to the single-source version. However, with the additional restriction to release only aggregated data among a certain number of other users. For example, a user is willing to share the heart rate while running at different speeds in aggregated form over multiple users.

4.3.2 Capabilities

A privacy policy's capability specifies a feature that a privacy policy should support in order to be useful in the considered scenario. They are an orthogonal concept to types of privacy policies.

- *Reusability* of privacy policies. The policy language should support policies defined on a logical level. This capability enables the user to reuse the same policy for multiple fields or sources of data.
- *Validity* of privacy policies expresses when the privacy policy is in effect and when it expires. This capability allows users to define planned policy updates in the future.

- *Revocation* of privacy policies means that a user can always revoke a previously granted privacy policy. Compared to the expiration of a policy defined by the *Validity* capability, revocation is an unplanned policy update.

- *Entity* of a privacy policy defines the service for which the privacy policy holds.

4.4 Challenges

The objective of this thesis is to design a user-centric privacy platform for the streaming scenario that supports and enforces a useful subset of the privacy policies presented in Section 4.3 with encryption. Building such a platform for the streaming scenario is linked with challenges regarding *privacy aspects* and *systems aspects*. The *privacy aspects* consider challenges related to a user-centric end-to-end approach to data privacy. The *system aspects* examine the challenges that arise for the integration and adoption in existing data processing pipelines.

Privacy Challenges

- *Data protection and enforcement:* PolicyCrypt must provide cryptographic guarantees that access to data for all external parties is limited to a form that complies with privacy policies. Otherwise, data always remains vulnerable to data breaches. Moreover, the data protection must remain intact even in the face of a subgroup of colluding users.

- *Privacy policy language:* A user-centric approach puts the burden of expressing privacy on the user. Hence, it is essential to equip users with a straightforward yet expressive privacy policy language as assistance.

- *Heterogeneous privacy policies:* In a user-centric approach to privacy, a privacy platform needs to cope with heterogeneous privacy policies because the privacy preferences might differ between two users.

System Challenges

- *Resource-constrained data sources:* In the considered streaming scenario, the data sources are often resource-constrained edge devices. As a consequence, they do not have the capabilities to deal with privacy policies and corresponding transformations. Ideally, a data producer should be unaware of any transformations on the server, and hence a change of privacy policy should require no adjustments at the data source. Furthermore, data producers are often unreliable, and thus privacy transformations should be robust against dropouts.

- *Scalability in the face of recurring streaming workloads:* Due to the increasing amount of generated data, the system needs to be able to scale to a large number of data streams. Additionally, the platform should support high-velocity streams while limiting the overhead to support privacy transformations in real-time. The recurring nature of streaming workloads poses a unique challenge. While finishing a privacy transformation for some time window, data for the next window already arrives and must be processed.

- *Compatibility with existing streaming frameworks:* Most services already have an existing data processing pipeline. As a result, a compelling privacy platform must be able to integrate within many of these data processing pipelines seamlessly. Towards this challenge, a privacy platform should equip service operators with a concise query language to express their requirements.

5 Cryptographic Design

This chapter discusses the different cryptographic building blocks of PolicyCrypt. The first section presents the design of the encryption scheme that data producers use to encrypt data streams. Subsequently, the next section examines the available privacy transformations on streams and explains how privacy controllers can construct the corresponding transformation tokens to enable these transformations. Finally, the last section introduces a new optimization of an existing secure aggregation protocol that is tailored to facilitate recurring privacy transformations spanning over multiple parties efficiently.

5.1 Stream Encryption

In PolicyCrypt, data producers encrypt data streams with a symmetric additive homomorphic encryption scheme to protect data confidentiality. The encryption scheme relies only on efficient symmetric cryptography. This property enables the encryption of high-velocity streams even on resource-constrained edge devices. Since the encryption scheme is additively homomorphic, an untrusted service provider can perform addition on top of the encrypted data stream. More specifically, PolicyCrypt applies a form of the modulo addition-based encryption scheme introduced in Section 2.2 which encrypts each message m with a new key k and a predefined modulus M:

$$Enc(m, k, M) = m + k \mod M \tag{5.1}$$

In this scheme, encryption and decryption only consists of a single addition and subtraction respectively. To decrypt the aggregation of multiple ciphertexts obtained by leveraging the additively homomorphic property of the scheme, a client has to subtract the corresponding keys:

$$\sum_i Enc(m_i, k_i, M) = \sum_i m_i + \sum_i k_i \mod M \tag{5.2}$$

However, this form of decryption can become expensive because decryption requires the same number of operations (i.e., sum up keys) as aggregating ciphertexts. Hence, PolicyCrypt modifies the above encryption scheme to enable more efficient decryptions of in-range aggregated ciphertexts.

The system supports two flavors of the optimized encryption scheme. One of them is the encryption scheme introduced in TimeCrypt [23], which works for equally spaced time series data (i.e., the time between two consecutive events is constant). The other one is a new form that also supports irregularly spaced data streams.

5.1.1 Equally spaced Data Streams

As mentioned above, PolicyCrypt has support for the encryption scheme for equally spaced data streams from TimeCrypt.

Definition 5.1 (TimeCrypt Stream Encryption [23]) *The message m_i occurring at the i-th time slot in the data stream is encrypted with:*

$$Enc(m_i, k'_i, M) = m_i + k'_i \mod M \qquad \textit{with } k'_i = k_i - k_{i+1}$$

where k_i and k_{i+1} are fresh keys from a pseudo-random key stream $\{k_0, k_1, k_2, \dots\}$.

In this encryption scheme, only the two boundary keys are required for the decryption of an in-range aggregated ciphertext (Eq. 5.2) because the inner keys cancel out:

$$\sum_{i=0}^{n} k'_i = (k_0 - \cancel{k_1}) + (\cancel{k_1} - \cancel{k_2}) \ \dots \ (\cancel{k_n} - k_{n+1})$$

Thus, decryption time is independent of the number of in-range aggregated ciphertexts, which allows for efficient decryption of large sums of ciphertexts:

$$\sum_{i=0}^{n} m_i = Dec\left(\sum_{i=0}^{n} c_i,\ k_0, k_{n+1},\ M\right) = \sum_{i=0}^{n} c_i - k_0 + k_{n+1} \mod M$$

The above encryption scheme requires that there is a message at every time slot (i.e., an equally spaced data stream). Otherwise, the decryption algorithm cannot use the property that keys cancel out, and decryption becomes expensive.

5.1.2 Irregularly spaced Data Streams

Unfortunately, more often than not, the events of a data stream do not occur in equally spaced intervals. A strawman solution for the problem could be to introduce neutral messages (i.e., with a zero value) for every timestamp without an event and then use an encryption scheme for equally spaced data streams. However, depending on the load, this introduces a significant overhead both in bandwidth and processing. As a result, PolicyCrypt requires a new modification of the symmetric additive homomorphic encryption scheme to support irregularly spaced data streams natively.

The modified encryption scheme assumes a data stream $\{e_0, e_1, \dots, e_{i-1}, e_i, \dots\}$ where an event $e_i := (m_i, t_i)$ is a pair consisting of a message m_i, which is an integer in the range $[0, M-1]$, and a timestamp t_i. The events in a stream are ordered by their timestamps (i.e., a data producer creates these events in-order), and timestamps are unique identifiers for an event. For encryption and decryption the data producer has access to a key stream $\{k_0, k_1, k_2, \dots\}$ derived from an initial shared master secret k using a pseudo-random function F_k (see Section 2.2). The PRF F_k maps the timestamp t_i of an event to a pseudo-random encryption key k_i.

Definition 5.2 (Stream Encryption) *Assume a data stream* $\{e_0, e_1, \ldots, e_{i-1}, e_i, \ldots\}$ *with* $e_i := (m_i, t_i)$*. Given the timestamp of the previous message* t_{i-1}*, the data producer encrypts a message* m_i *at time* t_i *with:*

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

where $k_{i-1} = F_k(t_{i-1})$ *and* $k_i = F_k(t_i)$*.*

Due to the optimized key construction and the additive homomorphic property of the encryption scheme, the decryption of an in-range aggregated ciphertext

$$(c_i,\ (t_{i-1}, t_i)) \oplus (c_{i+1},\ (t_i, t_{i+1})) \ = \ (c_i + c_{i+1} \bmod M,\ (t_{i-1}, t_{i+1}))$$

requires only the two outer keys $k_{i-1} = F_k(t_{i-1})$ and $k_{i+1} = F_k(t_{i+1})$ because the inner key k_i cancels out:

$$c_i + c_{i+1} \mod M \quad = \quad m_i + m_2 + (-k_{i-1} + \cancel{k_i}) + (-\cancel{k_i} + k_{i+1}) \mod M$$

As a result, even for larger in-range aggregations over more than two ciphertexts, decryption remains efficient. More specifically, for decrypting an aggregated ciphertext c_{ij}, which contains the sum of all messages from event e_i to event e_j with $i < j$, the data producer only needs to subtract the two outer keys $-F_k(t_{i-1}) + F_k(t_j)$:

$$SDec(k, (c_{ij}, (t_{i-1}, t_j))) = c_{ij} + F_k(t_{i-1}) - F_k(t_j) \mod M$$

The scheme requires keeping track of the required keys for decryption; hence both the start- and end-timestamp are encoded in the ciphertexts. This encoding permits to always cancel keys while aggregating ciphertexts. Note that the key derivation and canceling technique does not impact the semantic security of the encryption scheme [23, 30].

5.2 Privacy Transformations

The main idea of PolicyCrypt is that privacy controllers submit transformation tokens, which enable a server to perform a privacy transformation on encrypted data streams. PolicyCrypt decouples the data plane from the privacy plane. As a consequence, the privacy controllers, which are part of the privacy plane, are unaware of the data that is generated by the data producers in the data plane.

A privacy transformation Γ is a function that takes as input a set of transformation tokens $\{\tau_1, \tau_2, \ldots\}$, and a set of ciphertexts $\{c_1, c_2, \ldots\}$ from the participating streams. The transformation tokens only allow an untrusted server to perform computations on top of the encrypted streams that do not violate any of the involved privacy policies. In theory, a transformation token τ could be an arbitrary object which facilitates such a privacy transformation. However, in PolicyCrypt, all transformation tokens are an element of the additive homomorphic group of the stream encryption *Enc* (i.e., an integer in the range $[0, M-1]$). Currently, PolicyCrypt supports additive transformations over time windows in a data stream and aggregation of these windows across multiple streams.

For example, a privacy transformation for streams of heart rate data can be that a service computes the average heart rate over one day in California.

These privacy transformations are simple yet powerful because PolicyCrypt can support a wide range of typical streaming queries by leveraging aggregation-based encodings. The idea behind aggregation-based encodings is that for many queries, there exists an encoding of the input data that allows obtaining the query result only using addition.

5.2.1 Aggregation-based Encodings

The goal of aggregation-based encodings is to evaluate a function $f(x_1, x_2, \dots, x_n)$ by transforming large parts of the computation into an aggregation (Fig. 5.1). An encoder encodes a value x as a vector $\vec{x}$. Afterward, an aggregator collects an arbitrary number of encoded vectors $\vec{x_i}$ and sums them up element-wise. Finally, a decoder takes the aggregated vector and converts the vector into the desired result. Both the encoder and the decoder should be efficient.

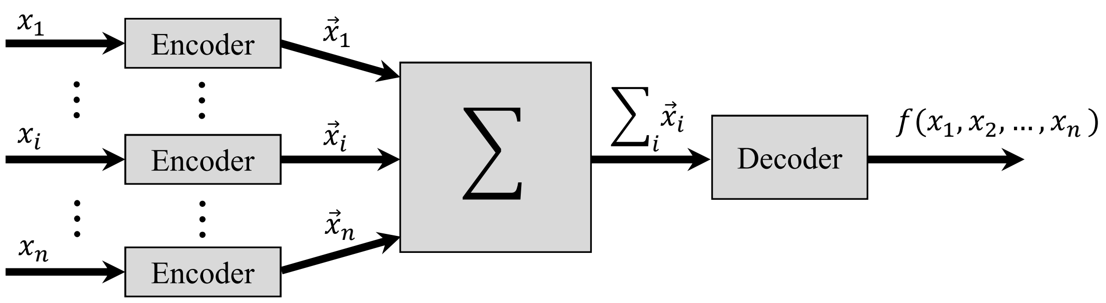


Figure 5.1: Leverage aggregation-based encodings to compute a function $f(x_1, x_2, \dots, x_n)$ on multiple inputs x_i.

The following list demonstrates existing techniques [32] of how to transform statistical queries into aggregation-based queries:

- `Sum`: the encoder maps $x_i \to [x_i]$ and the decoder maps $[\sum x_i] \to \sum x_i$.
- `Count`: the encoder maps $x_i \to [1]$ and the decoder maps $[\sum 1] \to n$.
- `Average`: the encoder creates vector $x_i \to [x_i, 1]$ and the decoder computes $[\sum x_i, \sum 1] \to \sum x_i / n$.
- `Variance` / `Standard Deviation`: the encoder creates vector $x_i \to [x_i, x_i^2, 1]$ and the decoder leverages the variance formula $\mathbb{E}[X^2] - \mathbb{E}[X]^2$ to compute $[\sum x_i, \sum x_i^2, \sum 1] \to (\sum x_i^2/n) - (\sum x_i/n)^2$. For the standard deviation, the decoder additionally takes the square root of the result.

- `Histogram`: the encoder divides the range of values into a series of k intervals $\{b_0, b_1, \ldots, b_{k-1}\}$ called bins. A value x_i which falls into the j-th bin is encoded as a vector of size k where all elements are zero except the j-th element is one (i.e., $x_i \rightarrow [0_0, \ldots, 0_{j-1}, 1_j, 0_{j+1}, \ldots, 0_{k-1}]$). The decoder is the identity function because the aggregated vectors directly correspond to the histogram's bin counts.

- `Max` / `Min`: a bucketed version of minimum and maximum uses the same encoding as the histogram. The decoder returns the maximum or minimum bucket with a count > 0, respectively.

- `Regression`: Given a training set $\{(x_i, y_i) | \, i = 1, \ldots, n\}$ of data points, aggregation-based encodings allows to train an ordinary least square regression model $h(x) = a_1 x + a_0$. The encoder encodes a training pair into a vector $(x_i, y_i) \rightarrow [x_i,\, x_i^2,\, y_i,\, x_i y_i,\, 1]$. The corresponding decoding algorithm needs to solve a small system of linear equations to obtain the model coefficients a_0 and a_1 for a training set of n observations:

$$\begin{pmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \end{pmatrix}$$

5.2.2 Transformation Tokens

In PolicyCrypt, the privacy controllers create transformation tokens with access to the respective stream encryption keys but without direct access to raw data. Similar to the data producer, the privacy controller can derive the necessary keys from the corresponding master secret. For transformations involving multiple parties with different privacy controllers, the involved privacy controllers perform a secure multiparty computation protocol to construct the transformation tokens. Using a secure MPC protocol guarantees that the individual transformation tokens do not reveal any keys which allow the server to decrypt messages from an individual data stream. Currently, the privacy controller in PolicyCrypt can create three types of tokens that build on each other.

Tokens for Window Transformation

A window aggregation for the i-th tumbling window consists of summing over all messages with timestamps in the range $[t_i,\, t_i + w)$ where w is the window size.

As long as data producers use the encryption scheme for irregularly spaced data streams from Section 5.1 and additionally submit a message on each window border, the in-range aggregated ciphertext of the window has the form:

$$\sum_{s \in [t_i, t_i + w)} Enc(m_s, k'_s, M) = \sum_{s \in [t_i, t_i + w)} m_s - F_k(t_i - 1) + F_k(t_i + w - 1) \mod M$$

which is, due to the fact that the end of the i-th window is the start of the $(i+1)$-th window (i.e., $t_i + w = t_{i+1}$), equivalent to:

$$\sum_{s \in [t_i, t_{i+1})} Enc(m_s, k'_s, M) = \sum_{s \in [t_i, t_{i+1})} m_s - F_k(t_i - 1) + F_k(t_{i+1} - 1) \mod M$$

As a result, the required token for the i-th window transformation has the form: $\tau = F_k(t_i - 1) - F_k(t_{i+1} - 1)$. The semantic security of the encryption scheme ensures that the token τ enables the service to only decrypt the window aggregate (i.e., perform the defined transformation) but nothing else.

Note that in case a data producer does not have a message at the window border, it must add an empty message. Figure 5.2 shows this procedure. The window borders are defined as the timestamps $t_i - 1$ and $t_i + w - 1$. However, since a window transformation recurs, the data producer has to add only up to a single record per window because, as mentioned above, for consecutive tumbling windows, the end-border of window i is the start-border of window $i+1$. These border messages ensure that the privacy controller does not need access to raw data to determine the required decryption key.

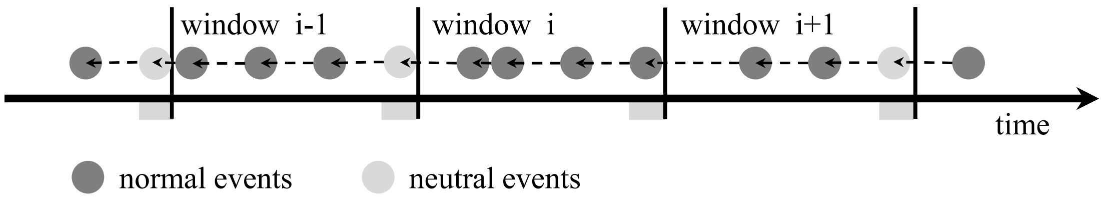


Figure 5.2: Irregularly spaced data stream encryption with neutral window border messages.

Tokens for Multi-Stream Transformation

A multi-stream aggregation is an aggregation of messages within a tumbling time window $[t_i,\, t_i + w)$ involving multiple data streams (i.e., from different data producers).

Let the window aggregate $m_{aggr_i} = \sum_{s \in [t_i, t_{i+1})} m_s$ be defined as the sum of all messages in the i-th tumbling window (i.e., with a timestamp in the range $[t_i,\, t_i + w)$) and let $k_{aggr_i} = -F_k(t_i - 1) + F_k(t_{i+1} - 1)$ be defined as the corresponding window key. The aggregation of all ciphertexts from all streams in S results in the sum of all window aggregates and the sum of all window keys for the i-th window:

$$\sum_{j \in S} c^{(j)}_{aggr_i} = \sum_{j \in S} m^{(j)}_{aggr_i} + \sum_{j \in S} k^{(j)}_{aggr_i} \mod M$$

Hence, the server must obtain a transformation token of the form $\tau = -\sum_{j \in S} k^{(j)}_{aggr_i}$ to complete the privacy transformation and decrypt the window aggregate. However, the server should not learn any individual window keys $k_{aggr_i}^{(j)}$ from the transformation token as this would allow him to decrypt the window aggregate of an individual stream which is equivalent to a simple window transformation. As shown in Section 2.2, the problem of computing a multiparty sum without any participant revealing its contribution can be solved with secure aggregation, a specialized secure MPC protocol.

In PolicyCrypt, the privacy controllers and the server participate in a secure aggregation protocol to privately compute the sum over the set of transformation tokens $\{\tau_1, \tau_2, \ldots, \tau_{|S|}\}$ from the privacy controllers:

$$\tau = \sum_{j \in S} \tau_j = -\sum_{j \in S} k_{aggr_i}^{(j)}$$

The secure aggregation protocol proceeds in three phases:

1. In an initial setup phase (performed only once), the privacy controllers, exchange pairwise secrets.

2. For every transformation, each privacy controller creates a transformation token τ_j that hides the individual window key using the pairwise secrets.

3. The server collects all transformation tokens and computes the sum over all τ_j to facilitate the privacy transformation.

Noisy Tokens

A noisy transformation can build both on single- and multi-stream window transformations. The idea is that privacy controllers add carefully calibrated noise to the keys (i.e., submit noisy keys):

$$\tilde{\tau_j} = \tau_j + \eta_j \tag{5.3}$$

where η_j is noise from some arbitrary noise distribution. The goal of noisy tokens is to support privacy transformations that produce differentially private results. In previous work, the noise was usually added to ciphertexts rather than to decryption keys [12, 88]. The advantage of adding noise to keys rather than adding noise to the ciphertexts is that encrypted data remains unchanged. As a result, the same data is reusable for encrypted storage and to facilitate one or multiple differentially private privacy transformations.

PolicyCrypt supports an arbitrary additive noise mechanism from the differential privacy literature, which draws noise from a divisible distribution. Another constraint is that the mechanism cannot access data when adding the noise. Note that this constraint prohibits, for example, the use of the sparse vector technique. However, with the Laplacian and the Gaussian mechanism, introduced in Section 2.1, two of the most popular differentially private mechanisms are supported.

5.3 Secure Aggregation

This section aims to describe the employed secure aggregation protocol of PolicyCrypt to build transformation tokens over multiple parties.

The requirement for the protocol is that, *(i)* it remains lightweight in terms of computation for privacy controllers even for transformations involving many parties, *(ii)* it can be efficiently repeated with similar parties. Furthermore, *(iii)*, it must be robust to dropped parties. Based on these requirements, PolicyCrypt builds on the symmetric secure aggregation protocol from Ács et al. [12]. Their protocol integrates well with the design of the transformation tokens as it relies on the same additive homomorphic encryption scheme from Section 5.1. Even though Ács et al. provide an optimization compared to a Strawman solution, their protocol does not optimize towards recurring execution. As a result, PolicyCrypt proposes a new optimized version tailored to the specific requirements. In a setting with 10k parties and 2k repetitions, PolicyCrypt results in a 55x and 96x speedup compared to Dream from Ács et al. and the Strawman.

The remainder of this section outlines the PolicyCrypt optimization of the secure aggregation protocol. In the first part, the protocol introduction from Section 2.2 is supplemented with a reformulation in terms of PolicyCrypt. Afterward, a paragraph reframes the complexity of the protocol as a graph problem, which helps to reason about possible optimizations. Finally, the last part of the section builds up the PolicyCrypt optimization starting from a Strawman and the original formulation by Ács et al..

Privacy Transformation via Secure Aggregation Let $\mathcal{P}$ be a set of N privacy controllers where each of them has a token τ_p. Remember that the goal is that the server computes $\sum_{p\in\mathcal{P}} \tau_p$ without learning the individual inputs τ_p.

The protocol from Ács et al. realizes this by hiding the individual inputs with a special nonce k_p which has the property that the sum of all nonces is $\sum_{p\in\mathcal{P}} k_p = 0$. As a result, the sum of all encrypted inputs is equal to the sum of inputs:

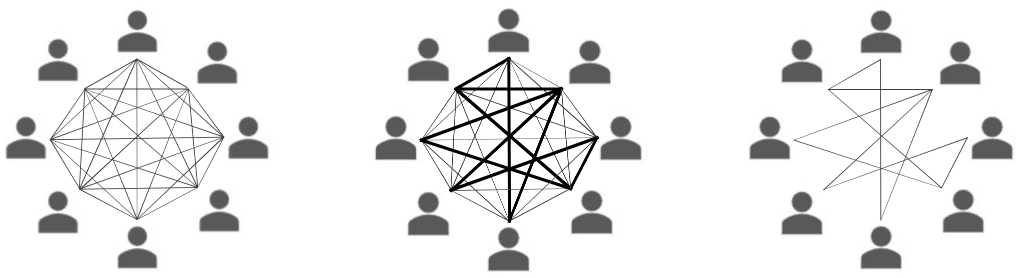
$$\sum_{p\in\mathcal{P}} \tau_p + \underbrace{\sum_{p\in\mathcal{P}} k_p}_{=0} \mod M \quad = \quad \sum_{p\in\mathcal{P}} \tau_p \mod M$$

Privacy controllers construct the nonces from pairwise shared dummy keys $d_{p,q}$, which they share with the other controllers. More specifically, privacy controller p adds $d_{p,q}$ if $p > q$ and otherwise subtracts $d_{p,q}$ from his input:

$$k_p = \sum_{q\in\mathcal{P}:\, p>q} d_{p,q} - \sum_{q\in\mathcal{P}:\, p<q} d_{p,q} \mod M$$

Secure Aggregation Graph This paragraph introduces a graph formulation that models the complexity of the protocol. Let the secure aggregation graph $G := (V, E)$

model the symmetric protocol with N parties. The set of vertices V represents the involved parties ($|V| = N$), and the set of edges E denotes the pairwise canceling masks.



(a) Strawman (b) Dream (c) PolicyCrypt

Figure 5.3: Graph representation of different secure aggregation approaches.

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

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

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

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

5.3.1 Strawman

The strawman solution for performing secure aggregation among N nodes based on pairwise canceling masks involves sharing a dummy key with all $N - 1$ other nodes. As a result, the secure aggregation graph $G := (V, E)$ forms a clique, which leads to an overall complexity of $\mathcal{O}(N^2)$. This N^2 is an upper bound on the number of required edges because, as previously shown, as long as all non-colluding clients form a single connected component, the aggregation is secure.

From the perspective of each participant, the time complexity is $\mathcal{O}(N)$ because all participants have to evaluate a PRF $N-1$ times and add together $N-1$ dummy keys. In terms of space complexity, each participant needs to store all $N-1$ shared keys.

5.3.2 Dream

With the goal of reducing the $\mathcal{O}(N^2)$ complexity, Ács et al. propose a distributed protocol for randomly selecting a subset of edges $E_c \subseteq E$ such that if node v_i selects v_j then node v_j also selects node v_i [12].

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

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

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

However, this leads to a problem that almost nullifies the benefits of having a lower degree in the graph. To select the neighbors, each node has to evaluate the PRF $N-1$ times. Let us assume that $\ell << N$ nodes were selected. Consequently, the node has to re-evaluate the PRF among all ℓ selected nodes with a different public changing value r_2 to generate the dummy key. In total, this leads to $N - 1 + \ell$ PRF evaluations for every round, which is even more than in the Strawman version. The only benefit is that now only ℓ instead of $N-1$ dummy keys need to be added up. As a result, from the perspective of each participant, the time complexity remains $\mathcal{O}(N)$. Additionally, the space complexity remains $\mathcal{O}(N)$ since each participant needs to store all $N-1$ shared keys.

5.3.3 PolicyCrypt Optimization

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

Before exploring how the PolicyCrypt optimization works, this part starts by previewing a result from the evaluation in Section 8.4 to motivate the optimization. The effect of the PolicyCrypt optimization is that for W rounds, with N participants and expected number of selected neighbors $\ell << N$, the optimization must evaluate the PRF only $N - 1 + W \cdot \ell$ times compared to $W \cdot (N-1)$ and $W \cdot (N-1) + W \cdot \ell$ in the Strawman and Dream respectively. Note that the large N is only an additive factor in PolicyCrypt while it is to a multiplicative factor in both the Strawman and Dream.

Sharp Connectivity Threshold Recall that the requirement for a secure aggregation is that the non-colluding nodes form a single connected component, which means that no part of the graph is isolated. In the Erdős–Rényi model, there are a fixed number of vertices n, and each edge is in the graph with probability p independently. In 1960, Erdős and Rényi studied the probability that the graph $G(n,p)$ is connected as a function of p [41]. Intuitively, for very small p, $G(n,p)$ consists of mostly isolated vertices, and for large p, $G(n,p)$ is connected with high probability. It turns out that the change from disconnected to connected with high probability is quite sudden at the critical threshold p_c:

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

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

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

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

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

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

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

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

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

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

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

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

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

subject to the constraint

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

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

In order to get an idea of how many rounds (i.e., random graphs) are possible from performing the procedure a single time, Table 5.1 shows a few examples of different numbers of participants to demonstrate the effect of the optimization. The number of graphs W and the expected degree of each vertex $\mathbb{E}[\text{degree}]$ holds under the assumption that up to half the nodes are colluding $\alpha := 0.5$ and a probability of success higher than $1 - \delta$ with $\delta := 1 \times 10^{-7}$.

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

Table 5.1: Expected degree of a node in a secure aggregation graph for different number of participants N and for different number of rounds W.

Mapping between Privacy Transformation and Secure Aggregation Graph During a privacy transformation, a participating privacy controller must always know which graph to use. This fact requires a mapping between the window of a privacy transformation and a secure aggregation graph. In PolicyCrypt, this is achieved by dividing a privacy transformation into epochs consisting of W rounds. In combination with an enumeration of the windows in a privacy transformation, this results in a direct mapping. To clarify, consider an example. Assume that an epoch consists of $W = 200$ rounds, then the 205-th repetition of the privacy transformation uses the 5-th graph of the second epoch.

6 System Design

This chapter presents the system design of the privacy platform PolicyCrypt introduced in Chapter 4. The chapter starts by describing how data producers can write their data streams into the privacy platform. This description is followed by a section that outlines a simple privacy policy language that allows users to express their privacy preferences, and a query language for service providers to define streaming queries. The next two sections outline the policy manager and the privacy controller, which, based on these two languages, provides an interface for service providers and users, respectively, to interact with PolicyCrypt. Further, the privacy controller is responsible for enforcing user-defined privacy policies with encryption while the policy manager is responsible for forming privacy transformations based on streaming queries. The final section describes the data transformer, which is essentially a job running in a streaming platform that performs the privacy transformation on top of encrypted data streams.

6.1 Writing Data

The data sources in PolicyCrypt are the data producers. Data producers write data streams through a client proxy module to an untrusted cloud infrastructure. However, before sending the records, the proxy module encrypts the records with a partially homomorphic encryption scheme that allows protecting confidentiality while preserving utility by enabling an untrusted server to perform certain computations on the encrypted data.

Remember that PolicyCrypt has a strict separation between the data- and the privacy plane. The data producer is part of the data plane and thus is entirely unaware of the privacy layer and is not involved in any transformation. This design has the advantage that the proxy module remains lightweight and does not require any interactions with the server other than sending the data. This approach is similar to existing stream processing pipelines. The only additional step performed on the data producer is to encrypt the data. However, this is also not a problem because the encryption scheme only relies on lightweight symmetric cryptographic operations. As a result, the client proxy module also runs on resource-constrained edge devices. Overall, the proxy architecture of PolicyCrypt is similar to the architecture used in TimeCrpyt [23].

This section begins by describing how to register a new data stream on the PolicyCrypt platform. The remaining part of the section explains how the data producer encodes and encrypts data before writing it into the platform.

Registration For setting up a new data producer, the user has to register the new data stream on the Privacy Controller API. With the stream registration, the user generates a master secret and assigns a new data producer to the privacy controller. After the registration process, the proxy receives the master key and additional parameters for encryption and data serialization from the generated setup file.

Data Record Serialization The server in PolicyCrypt supports encrypted aggregates based on addition but no multiplication. Despite this limitation, PolicyCrypt supports a wide range of statistical queries by leveraging the aggregation-based encodings introduced in Section 5.2.

The proxy module serializes each attribute of the data record into a vector of statistical values based on the configuration file obtained during the setup. In the default configuration, an attribute v is encoded as a vector $(v, v^2, 1)$ to support sum, average, and standard deviation.

Encryption The proxy module encrypts the data records with the symmetric homomorphic encryption scheme from Section 5.1 using the master secret from the setup phase. As a result of the separation between the data plane and privacy plane, the privacy controller requires a contract with the data producer that enables the privacy controller to derive the correct window keys without communicating with the producer.

Towards this goal, the proxy module receives a base transformation window size Δ (e.g. a minute) for the stream during the setup phase. On each border timestamp of this base window size (i.e., the last timestamp within the window), the proxy module must submit a ciphertext to the service. If there is no data produced at this specific point in time, the data producer encrypts a neutral value that does not affect the statistics. As a result, the privacy controller can derive transformation tokens for all window sizes that are a multiple of the base window size Δ without access to the data stream because the contract guarantees that the ciphertexts are present. Moreover, the data transformer interprets the border events as commit messages from the data producer, which indicates that the data of this window is complete. These commit messages are a mechanism that allows the data transformer to detect a producer that is currently unavailable.

Besides, PolicyCrypt supports submitting neutral ciphertexts at random points in time. Since the server can observe the timing of the events, these neutral ciphertexts are useful for hiding the exact data generation pattern. Furthermore, in a multi-stream windowed-aggregation where not all data producers have a value, the neutral ciphertexts can obfuscate the origin of the data.

6.2 Defining Privacy Policies and Queries

PolicyCrypt enables users to define privacy policies for their data streams through the Privacy Controller API. It is in the interest of users to share data only on their terms, which might differ depending on the type of data. As a result, each data stream in PolicyCrypt has an attached privacy policy that defines what transformation preserve privacy in the viewpoint of the user.

On the other side is the service provider with interest in executing queries on top of the data streams. PolicyCrypt provides a framework to reconcile the two interests. Towards this goal, the framework offers a language to express privacy policies and a language to express simple queries (i.e., privacy transformations). Further, PolicyCrypt provides a method to match queries to streams with compatible privacy policies. Finding a matching is only possible due to a data stream schema, which is the foundation for both defining privacy policies and queries.

First, this section introduces the data stream schema, followed by the privacy policy language and the streaming query language of PolicyCrypt. At the end of the section, a small case study illustrates the interplay between the three components.

6.2.1 Data Stream Schema

The data stream schema describes the characteristics of a data stream and provides sensible options for privacy policies. In PolicyCrypt, the service provider is responsible for defining such a data stream schema. However, also an independent governing body or a standardization committee could define the schema.

As Listing 6.1 shows, each schema has a unique *name* and consists of three major parts: *metadata attributes*, *stream attributes*, and possible *policy options*. PolicyCrypt defines schemas in the Yaml format [76] and uses the Avro schema language to define types of attributes [44].

Listing 6.1: Data Stream Schema Structure

```
name: <schema id>
metadataAttributes:
- name: <attribute id>
  type: <attribute type>
streamAttributes:
- name: <attribute id>
  type: <attribute type>
  aggregations: <attribute encodings>
streamPolicyOptions:
- option: <option type>
  <option params>
```

Metadata Attributes The metadata attributes describe static fields that remain constant for a more extended period (i.e., they are not regularly changed). The service uses metadata attributes to group the available streams into reasonable populations. For example, a service might want to know the average number of steps per day for people living in rural areas. Hence, the service filters streams based on the geographic area meta-attribute. In some cases, these metadata attributes potentially reveal sensitive information. To mitigate this problem, in many cases, it is sufficient to get a class rather than a precise value. For example, instead of revealing the age, users might be willing to disclose that their age is within the age range of 60-70 years. However, if this is not sufficient, the user always has the option to omit sensitive attributes. The data schema supports enums and arbitrary primitive types from the Avro schema language for metadata attributes.

Stream Attributes The stream attributes are the schema of the actual data stream. Remember that a data stream is an unbounded stream of events where each event consists of a timestamp and a data tuple. The stream attributes define the contents of such a tuple. Each element or field of the tuple has a name and a primitive type from the Avro schema language. Due to the homomorphic encryption scheme in PolicyCrypt, most fields are integers. However, other types are available, but with no support for private transformations on encrypted data. Instead, elements without integer types can either be public or private. In addition to a type, a stream attribute also defines a set of possible aggregations. They determine the aggregation-based encodings used during the stream encryption. By default, the encoding consists of sum, count, and the sum of squares, which enables common statistical queries, as shown in Section 5.2.

Privacy Policy Options A schema defines reasonable privacy policy options for stream attributes that correspond to the privacy transformations on encrypted data streams introduced in Section 5.2. PolicyCrypt supports the following privacy policy options ordered from the least to the most restrictive:

- *Public* is the least restrictive option and is available for all stream attributes, including non-integer types. The semantics of the *public* option is that for the user, the data is not sensitive, and thus all transformations are possible.

  ```
  - option: public
  ```

- *Window* corresponds to the privacy transformation that limits the resolution of the data. The option has a single parameter window size, which is a list of choices for the minimal allowed window size.

  ```
  - option: window
    window: <window size options>
  ```

- *Aggregate* is the counterpart of the privacy transformation that aggregates data from multiple streams in windows. The parameters of the aggregate option are a minimum window size and a minimum aggregation size. The window size is a list of choices that define the highest permitted resolution. For the aggregation size, it is possible to limit the choices to a discrete list of aggregation sizes (e.g. 100, 200, 500, 1000) or predefined levels (e.g. low, medium, large) with implicit aggregation sizes.

```
- option: aggregate
  client: <client aggregation sizes>
  window: <window size options>
```

- *Noise aggregate* corresponds to the privacy transformation that uses noisy tokens to produce differentially private results. The option takes as parameters both the notion of differential privacy and the (ϵ, δ)-budget.

```
- option:  dp
  notion:  <dp notions>
  epsilon: <dp epsilons>
  delta:   <dp deltas>
```

- *Private* is the default option for all stream attributes including non-integer types. The semantics of the *private* option is that this data cannot be used for any transformation. This option is the most restrictive, and stream attributes with this privacy policy are only used for encrypted storage.

```
- option: private
```

6.2.2 Privacy Policy Language

Apart from header information that defines origin, destination, and when the privacy policy is valid, a privacy policy in PolicyCrypt consists of two parts that build on the data stream schema. (i) A privacy policy contains values for the metadata attributes defined in the data stream schema, and (ii) a privacy policy selects a privacy configuration based on the available options from the data stream schema.

Header The header of a privacy policy contains a *userID*, which identifies the owner of the stream. In PolicyCrypt, the hash of the user's public-key represents an identity. However, any public-key infrastructure (PKI) compatible label is usable as long as the identity always maps to a public-key. Moreover, each stream has a unique *streamID* within the namespace of the user. The *serviceID* identifies the service owner for whom this policy applies. Finally, the header ends with a *validity* tag that defines the time frame for when the policy is in effect. In case the policy is valid until further notice, the *to* tag can also be left empty.

Listing 6.2: Privacy Policy Structure

```
userID:   <user id>
streamID:  <stream id>
serviceID: <service id>
validity:
  from: <valid from timestamp>
  to:   <valid to timestamp>
stream:
  schema: <schema id>
  metadataAttributes:
  - <attribute id>: <value>
  privacyConfiguration:
  - <option type>:
    <option param>: <value>
    attributes: <stream attributes>
```

Stream Configuration Under the *stream* tag, the policy defines which data *schema* the policy belongs to. The *metadataAttributes* tag contains a list of key-value pairs that define metadata of the stream. Only the metadata attributes defined in the schema are supported. The policy ends with the actual privacy configuration for the stream attributes. The *privacyConfiguration* tag contains a list of the selected privacy options from the schema, including the chosen parameters. The *attributes* tag under every selected option defines the stream attributes to which this policy applies. Note that this arrangement allows both defining multiple privacy policies for the same stream attribute and assigning the same privacy policy to multiple stream attributes.

6.2.3 Stream Query Language

The query language of PolicyCrypt builds on KSQL [57], an SQL-like query language for expressing continuous queries on data streams. A prototypical query in PolicyCrypt has the following structure:

```
CREATE STREAM <stream name> (<stream attributes>) AS
  SELECT <UDFs with stream attributes>
  WINDOW TUMBLING (SIZE <time>, GRACE PERIOD <time>)
  FROM <stream schema> BETWEEN <range>
  WHERE <metadata attribute predicate>
```

This query creates a new privacy policy compliant stream which has a name and a set of stream attributes. The *select* clause applies user-defined functions (e.g. sum, avg, count) to stream attributes from the stream schema. The *window* clause defines the window of the privacy transformation. The *from* clause defines the schema type over which this query runs, and additionally, it provides the possibility to set a minimum and a maximum on the number of required streams. The *where* clause allows defining predicates on the metadata attributes to filter out streams.

The PolicyCrypt query language is designed to offer a simple set of transformations to create privacy-preserving data streams that are guaranteed to comply with the user-defined policies. The reason for this design choice is that PolicyCrypt intends to serve as the first step in a typical streaming data processing pipeline. Further and potentially more complex queries can then run on top of the privacy-preserving data streams using KSQL or an arbitrary other stream processing engine.

User-Defined Functions The KSQL query language provides a concept called user-defined functions (UDF) to perform more complex and custom operations on data streams [57]. They are available within the *select* clause of a query. PolicyCrypt leverages UDFs to build a decoder for the aggregation-based encodings, which improves the compatibility with existing stream processing pipelines. Table 6.1 lists the user-defined functions currently integrated in PolicyCrypt.

Function	Description	Default
`sum(x)`	windowed sum of the stream attribute x	✓
`avg(x)`	windowed average of the stream attribute x	✓
`count(x)`	number of events within a window	✓
`stddev(x)`	standard deviation of attribute x within a window	✓
`var(x)`	variance of attribute x within a window	✓
`hist(x)`	histogram of counts in a window for stream attribute x	
`reg(x,y)`	regression coefficients trained on data in this window	
`max(x)`	max bucket of a stream attribute x within window	
`min(x)`	min bucket of a stream attribute x within window	
`sumdp(x)`	differentially private sum of the stream attribute x	

Table 6.1: User-defined functions supported by PolicyCrypt. The last column shows which UDFs are available by default (i.e., without a schema annotation).

Time Window The query language of PolicyCrypt has support for tumbling- and hopping-windows, which were both introduced in Section 2.3. A *tumbling window* requires a size parameter that determines the length of the window and a grace parameter that specifies how long to wait for out-of-order records.

```
WINDOW TUMBLING(SIZE <time>,
                GRACE PERIOD <time>)
```

A *hopping window* also requires a size parameter and a grace parameter. However, additionally, it takes a slide parameter that determines how much the window advances in every step.

```
WINDOW HOPPING(SIZE <time>,
               ADVANCE BY <time>,
               GRACE PERIOD <time>)
```

Tumbling windows directly correspond to the window concept in privacy policies, which limits the data resolution. Such a direct counterpart does not exist for hopping windows. However, it turns out that the query language can support hopping windows by the insight that for addition-based aggregation, the output of a hopping window can be emulated by combining the results from a set of tumbling windows as long as the size of the hopping window is a multiple of the slide.

Filtering The details on how PolicyCrypt matches the available streams to the queries are described in Section 6.3. However, the PolicyCrypt query language has three mechanisms to filter and constrain the choices of streams:

- In the *from* clause a query can define a range on the number of streams.

  ```
  FROM <stream schema> BETWEEN <range>
  ```

  For example, a range `BETWEEN 100 AND 200` specifies that the query requires between 100 and 200 streams to be reasonable.

- A *where* clause filters the available streams based on the metadata attributes of the streams.

  ```
  WHERE <metadata attribute predicate>
  ```

  For example, `WHERE age='old' AND region='California'` specifies that the query only wants to include streams from old people in California.

- A *group by* clause categorizes the available streams based on a metadata attribute. Defining a *group by* clause in a query results in a privacy transformation per class in the group by attribute.

  ```
  GROUP BY <metadata attributes>
  ```

  For example, assuming the metadata attribute age has three classes (young, middle-aged, old), `GROUP BY age` specifies that the query should combine streams into the three available groups. As a result, three separate privacy transformations are formed.

6.2.4 Case Study

The following case study illustrates the interplay of *stream schema*, *privacy policy*, and *streaming query* based on a medical sensor application. The medical sensor stream has age and region as metadata attributes and heart rate and heart rate variability (hrv) as stream attributes. The stream schema defines that either a stream attribute is private or that it is meaningful to define an aggregation among a medium or large group of streams over a window of either one or four hours.

The displayed privacy policy selects the aggregation privacy option for the heart rate and defines that the aggregation must be performed over at least a one hour window and with a medium-sized group of streams. The hrv attribute remains private. The service provider defines an aggregation query to find the average heart rate over one hour involving up to 1000 streams from the group of older people in California.

Listing 6.3: Stream Schema

```
name: MedicalSensor
metadataAttributes:
- name: age
  type: [enum, optional]
  symbols: [young, middle, old]
- name: region
  type: string
streamAttributes:
- name: heartrate
  type: integer
  aggregations: [var]
- name: hrv
  type: integer
streamPolicyOptions:
- option: aggregate
  clients: [medium, large]
  window:  [1hr, 4hr]
- option: private
```

Listing 6.4: Privacy Policy

```
userID:   2474b75564b
streamID:  235632224234
serviceID: healthsensorapp.com
validity:
  from:2020-04-20T17:42:02+02:00
  to:  2021-04-20T17:42:02+02:00
stream:
  schema: MedicalSensor
  metadataAttributes:
  - age: old
  - region: California
  privacyConfiguration:
  - aggregate:
    clients: medium
    window: 1hr
    attributes: [heartrate]
  - private:
    attributes: [hrv]
```

Listing 6.5: Streaming Query

```
CREATE STREAM HeartRateCaliforniaOld (heartrate) AS
  SELECT AVG(heartrate)
  WINDOW TUMBLING (SIZE 1 HOUR, GRACE PERIOD 5 SECONDS)
  FROM MedicalSensor BETWEEN 1 AND 1000
  WHERE region = 'California' AND age = 'old'
```

6.3 Policy Manager

Section 6.2 introduced the privacy policy language, the query language, and the data stream schema. This section describes how the policy manager, operated by the service provider, uses the three components to organize privacy policy conform transformations on data streams. On a high level, the policy manager is responsible for matching the privacy interests of users with interest in specific queries from the service provider.

The policy manager in PolicyCrypt consists of two parts, a *query planner*, which is responsible for collecting privacy policies and providing a query interface for service administrators, and a *transformation manager* which is responsible for orchestrating the data transformers that execute the queries. All communication between the policy manager and the user goes over the privacy controller.

6.3.1 Query Planner

The job of the query planner is to parse the queries and find a match between the queries and the available streams without violating any privacy policy. Both privacy policies and queries refer to one of the data stream schemas maintained by the query planner. Note, the policy manager is not responsible for enforcing privacy policies. Nevertheless, the service provider's interest is that the query planner finds a matching that does not violate any privacy policy. Otherwise, the privacy controllers responsible for enforcing the violated policies, do not participate in the query. As a result, the query does not execute successfully.

The query planner processes the queries in two phases. In the *first phase*, the query planner filters the available streams based on the schema type in the *from* clause, and the metadata attribute predicate in the *where* clause of the query. The resulting filtered set of streams contains all available candidates for the query.

In the *second phase*, the query planner checks the privacy policy of each stream for compliance. At this moment, the query planner drops streams that do not offer a privacy policy that is compatible with the query. How to check privacy compliance optimally is non-trivial, as finding an optimal matching between a set of queries and a set of streams is a complex combinatorial optimization problem with several possible metrics to optimize. For example, both minimizing or maximizing the number of streams in a single query might be a reasonable goal for a service provider. More streams per query lead to more meaningful results. However, at the same time, the service might need to follow a very restrictive policy to form such a large group. Instead, the service provider could get more fine-grained data access by excluding some streams with strict policies. However, this is only one example; further metrics to optimize could be the number of possible queries or prioritize compliance with more complex queries.

PolicyCrypt currently applies a simple greedy matching strategy but can be extended with more advanced approaches. The simple matching strategy considers only one query at the time, checks that the window size defined in the privacy policy is compatible, that the group size exceeds the requested size, and in case of a differential privacy query that the privacy budget is sufficient.

A data stream can only be matched to one query and is removed from the set of queryable streams if the stream is part of a transformation. This restriction ensures that an attacker cannot combine outputs from different transformations to violate privacy policies. Consider two separate aggregation transformations over two almost identical sets except that Alice is only in one of the sets. By subtracting one result from the other, an attacker can obtain the individual input of Alice.

The constraint of a single transformation per stream limits the number of transformations running at the same time. However, the service administrator can define a range on the number of streams required for a meaningful query in the *from* clause. If the number of compatible streams exceeds this limit, the query planner removes the streams with the least restrictive privacy policies. With these limits, the service has more control over the number of streams that remain available for later queries.

In PolicyCrypt, the matching between queries and streams is called a transformation plan. The transformation plan consists of the query itself, a minimum number of users required for the transformation and the list of *userIDs* that participate in the transformation. The most restrictive privacy policy defines the minimum number of users. Usually, the list of *userIDs* is longer than the minimum number of users to tolerate some dropouts. Note that, in some cases, the list might also be empty, and the query cannot run on the available streams. In this case, the service provider has to update the query. The query planner gives the transformation plan to the transformation manager.

When a user updates or revokes a privacy policy, the query planner checks the compatibility of the new privacy policy with the transformation plan and potentially removes the stream from the transformation. However, as long as there are more than the minimum number of users remaining, the transformation can continue and simply treat the stream as dropped.

6.3.2 Transformation Manager

The transformation manager receives as input a set of transformation plans. Consequently, the transformation manager's job is to ensure that the existing transformation plans execute, which consists of two parts. First, the transformation manager needs to distribute the transformation plan to all privacy controllers involved in the transformation. Second, the transformation manager is responsible for starting a job in the stream processing system that executes the privacy transformation. Depending on the size and complexity of the transformation, this can also involve multiple jobs. In PolicyCrypt, the component that executes the privacy transformation is called the data transformer.

6.4 Privacy Controller

In PolicyCrypt, the privacy controller is the component which is responsible for expressing and enforcing privacy policies. The first part of this section describes the interface that the privacy controller offers to users to register data streams and to express privacy policies. The following parts outline the *setup phase* to bootstrap a new privacy transformation and the *privacy transformation phase* that helps to facilitate transformations while enforcing the privacy policy.

6.4.1 User Interface

The privacy controller provides an interface for users to interact with PolicyCrypt. The API consists of two parts, a Registration API used for setting up new data producers and a Privacy Policy API to define and change privacy policies for streams.

Registration API The Registration API allows a user to register new data streams with the privacy controller. During this process, the privacy controller fetches data stream schemas from the policy manager, generates a master secret, and creates a configuration file for the data producer based on the data stream schema.

Privacy Policy API The Privacy Policy API allows a user to create, update, or revoke privacy policies based on the available data stream schemas. After submitting a new or updated version of a privacy policy to the policy manager, the privacy controller needs to be ready in case the query planner includes the stream in a privacy transformation. A new privacy transformation always starts with a setup phase based on the transformation plan.

6.4.2 Setup Phase

The setup phase starts, once the privacy controller receives a transformation plan from the policy manager. The privacy controller verifies the compliance of the transformation plan against the user-defined privacy policy. In PolicyCrypt, this involves verifying that for the included attributes, the window size, aggregation size, and noise configuration do not violate the privacy policy. In the case of transformation involving streams from multiple users, the privacy controller fetches certificates from the PKI and verifies the involved identities. As a next step, the privacy controller establishes a pairwise shared secret with the other privacy controllers using a Diffie-Hellman key exchange protocol. The pairwise shared secrets build, combined with the hash of the transformation plan, the master secrets for the MPC protocol in the privacy transformation phase.

Support for larger Universes Establishing a large number of shared secrets using the Diffie-Hellman key exchange is expensive. As a result, in many settings, this prohibits transformation plans involving a large number of identities. The evaluation in Section 8.1 shows the costs associated with different transformation plans. This section outlines a basic idea on how to extend the capabilities of PolicyCrypt for massive transformations.

One of the key ideas of the PolicyCrypt secure aggregation protocol is that no one knows the complete structure of the secure aggregation graph because every participant is only aware of their neighbors. In the current setting, every node can potentially have an edge with every other node. As a result, every node needs to establish a pairwise secret with every other node to determine whether the edge exists. This property can be relaxed to support large universes efficiently. An option is to determine the possible neighbors only from a smaller set because this would mean that each node would need to perform fewer Diffie-Hellman key exchanges. However, doing this naively could result in a disconnected secure aggregation graph.

The idea for optimizing the setup phase starts by using an efficient cryptographic hash function to partition all N participants into a fixed number of buckets $\{b_0, b_1, \dots b_{p-1}\}$ based on the user id, such that in expectation each bucket contains N/k participants. Afterward, a participant of the i-th bucket, forms a bucket group $\mathcal{B}_i$ with participants from "neighboring" buckets.

$$\mathcal{B}_i = b_{(i-k \bmod p)} \cup \dots \cup b_i \cup \dots \cup b_{(i+k \bmod p)}$$

The Diffie-Hellman key exchange is now only performed with participants in the same bucket group instead of all N. These bucket groups fulfill two properties:

- Participant u is in the bucket group of participant v if and only if participant v is in the bucket group of participant u.
- The bucket groups are overlapping (controlled by the parameter k).

Informally, a consequence of these two properties is that when all participants perform the secure aggregation protocol from Section 5.3 on their bucket group $\mathcal{B}_i$, the secure aggregation graph including all N participants forms a connected component as long as the individual bucket groups form a connected component. In this optimization, the bucket groups are public knowledge. However, due to the overlapping bucket groups that form a ring, an attacker would require $2k$ buckets full of colluding participants to disconnect the secure aggregation graph.

6.4.3 Privacy Transformation Phase

During the privacy transformation phase, the privacy controller's task is to continuously check that the desired transformation does not violate the privacy policy. This check involves verifying that the aggregation includes enough participants or that the (ϵ, δ)-budget for differential privacy is not depleted.

For privacy transformations over a single stream (e.g. in window transformations), a privacy controller can calculate the transformation token for any window, because the token does not have to be adapted to the situation that an unknown subset of other streams is not available during the respective window. As a result, the privacy controller can pre-calculate the transformation tokens and send them to the server. Under the assumption that no data producer and no privacy controller drops out, the same holds for transformations over multiple streams. However, this is an unrealistic assumption in particular for data producers, which usually run on cheap, low-power hardware.

Fault Tolerance To be robust in case of failures, the privacy controller needs to interact with the server before building and submitting a transformation token.

This interaction is required because the secure aggregation protocol, to construct transformation tokens for multi-stream aggregations, relies on the participants' information (i.e., the universe of the transformation). To provide this information in PolicyCrypt, the privacy controller maintains a local representation of the universe. Initially, the privacy controller bootstraps the local representation with the list of *userIDs* from the transformation plan. At the end of every window, the data transformer requests a heartbeat from all privacy controllers. Afterward, the data transformer computes the intersection of available data producers and privacy controllers and broadcasts a membership delta compared to the previous window to all privacy controllers. After updating the local representation of the universe with this delta information, the privacy controller builds the transformation token and sends it to the server. Figure 6.1 illustrates the communication of a single window.

6.5 Data Transformer

The data transformer is a job running on the streaming platform that is responsible for performing the privacy transformation on top of encrypted data. The job is started by the transformation manager based on a transformation plan, which contains a compatible matching between streaming query and available privacy policies.

As part of this responsibility, the data transformer aggregates the arriving encrypted data from all data producers of the transformation. In parallel, the data transformer collects the required transformation tokens and finally performs the actual privacy transformation. During this process, the data transformer is also responsible for detecting and handling both data producer and privacy controller dropouts. Figure 6.1 shows the interaction for a single window. As mentioned in Section 6.4, after detecting a dropout of a producer or controller, the data transformer broadcasts a membership delta to the previous window to all privacy controllers.

The following two paragraphs describe how the transformer detects dropouts of data producers and privacy controllers.

Data Producer Dropout A dropout of a data producer is detected by checking the observed timestamps of an individual data stream. As mentioned in Section 6.1, the data transformer interprets the window border events as commits. When a window boundary timestamp is missing after the grace period is over, a data producer is considered to be dropped for this window.

Privacy Controller Dropout An unavailability of a privacy controller is detected via a heartbeat mechanism. The privacy controller sends a commit to the data transformer to indicate the preparedness for sending a transformation token at the end of a window. The data transformer defines a timeout for collecting commits. When the timeout is reached, all privacy controllers that failed to commit are marked as dropped for this window. PolicyCrypt focuses on minimizing the time

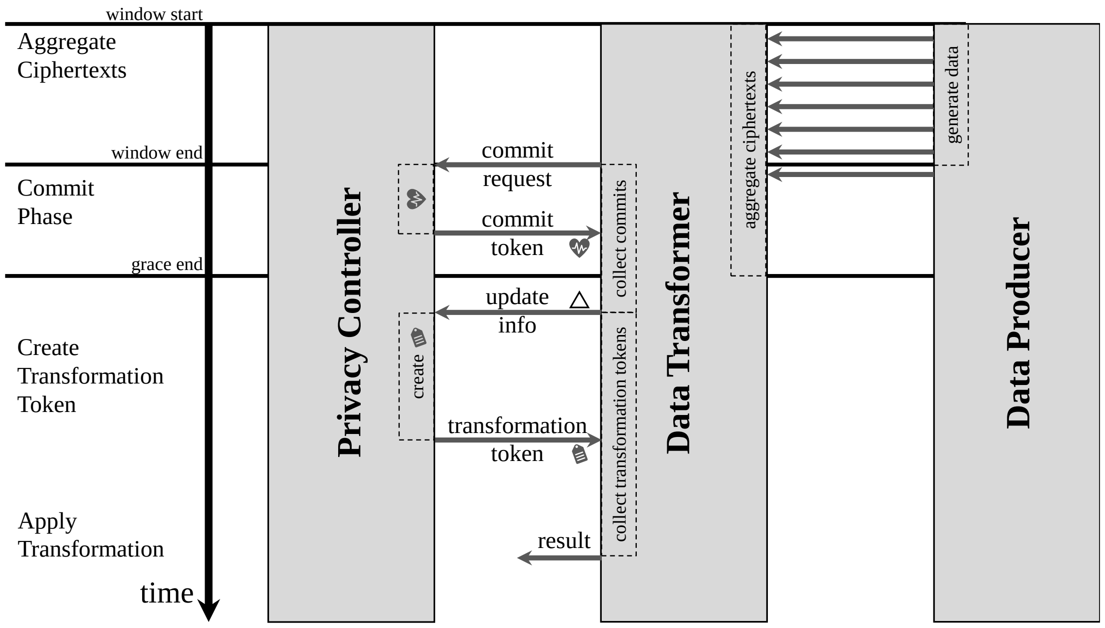


Figure 6.1: Interaction for a single window of a privacy transformation between data producer, data transformer, and privacy controller.

between committing and sending a transformation token. However, in the rare case that a privacy controller fails in between the two events, a transformation cannot be completed until the privacy controller recovers and provides the missing transformation token.

Note that sending a commit (heartbeat) followed by the transformation token is functionally equivalent to the protocol of Ács et al. [12]. Instead of sending a commit, they send a masked value. After the client receives updates from the server about the dropped clients, the client adjusts its contribution and lifts the masked value. As long as no privacy controller drops out in between sending the commit and the token or in between the first masked value and the second, the protocol is robust to clients dropping out. However, the protocol from PolicyCrypt has the advantage that a commit is much smaller than a masked value.

6.5.1 Post-Processing

After performing the privacy transformation on encrypted data, results are available in plaintext in a privacy policy compliant form. The resulting stream enables the data transformer in PolicyCrypt to apply a simple set of post-processing requested by the query. For example, this involves decoding the aggregation-based encoding or implementing the hopping window. If this is not sufficient, a service provider can always use the existing stream processing pipeline to implement more complex post-processing on top of the privacy policy compliant result streams.

7 Implementation

This section provides an overview of the implementation of PolicyCrypt that is used in the evaluation. The PolicyCrypt platform is implemented on top of Apache Kafka [46]. Figure 7.1 shows a sketch of PolicyCrypt's implementation and how the different components communicate and interact.

The implementation consists of a Java library for the privacy controllers and the data producers and a Kafka Streams application to execute the privacy transformation. The following sections describe each component in detail.

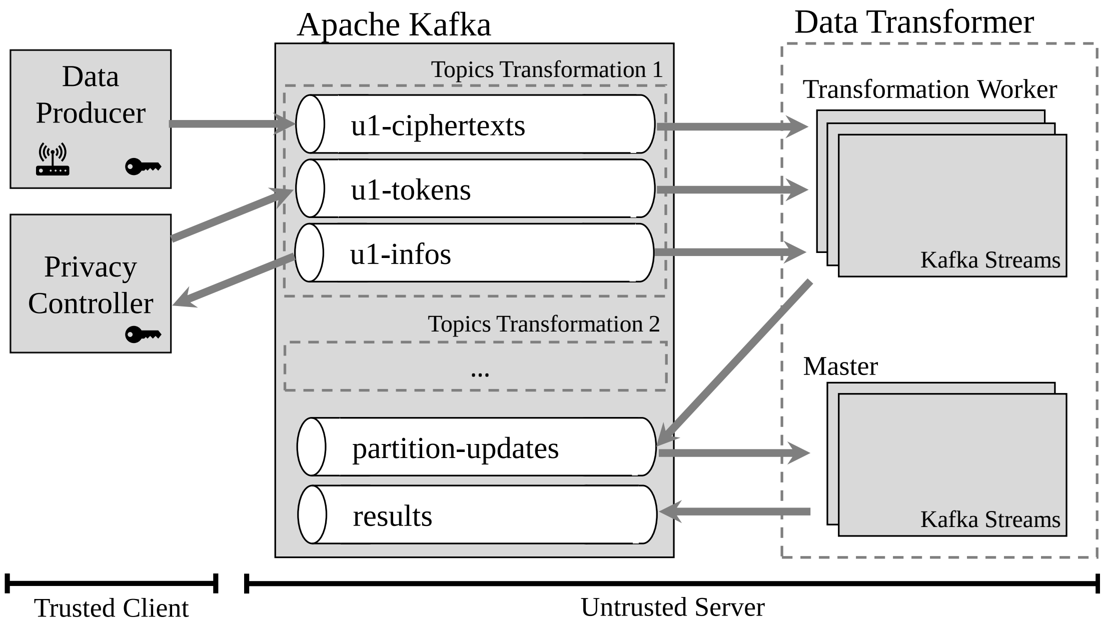


Figure 7.1: Implementation Overview

7.1 Data Producer

The data producer is a lightweight Java library that handles stream preprocessing, serialization, and encryption. It leverages the Kafka Producer API to send the encrypted digests to the Kafka cluster. In the current implementation of the data producer, all attributes of an event are 64-bit integers, and Avro [44] is used for data serialization.

The PRF of the encryption scheme is instantiated with an AES block cipher using 256-bit keys. The data producer relies on the AES implementation of the default

Java security provider. The encryption scheme adjusts the 128-bit PRF output to the 64-bit attribute size using a technique called length matching hash function [25].

The data producer assigns two timestamps to every event; first, the actual timestamp of the event, and second the timestamp of the previous event used in the encryption scheme. The prototype uses Unix timestamps, describing the number of milliseconds elapsed since the Unix epoch, as these timestamps. In PolicyCrypt, the event timestamps must be unique per data producer because otherwise, the same nonce is used more than once in the encryption. As a result, the theoretical throughput that a single producer can achieve has a limit of 1000 records per second. However, in cases where this is not sufficient, an implementation could always use timestamps in micro- or nanoseconds. The library provides two simple methods, listed in Table 7.1, for developers to integrate PolicyCrypt with existing data stream producers.

API	Description
`submitRecord`	Encrypts, serializes, and sends the record to PolicyCrypt.
`submitHeartbeat`	Triggers an update in case no other records are available.

Table 7.1: API to interact with the PolicyCrypt data producer.

7.2 Privacy Controller

The privacy controller is implemented as a standalone Java app and uses the Kafka Consumer and Producer API for communication with the server.

A privacy controller can be responsible for handling the privacy transformations of multiple data producers in different universes. For each universe, the privacy controller subscribes to the info topic of the respective universe to get updates on the progress of the transformations. Using updates from this connection, the privacy controller knows when to send commits, how the members of the universe changed over time, and when to send the transformation tokens. The privacy controller writes both the commit messages and the transformation tokens to the token topic belonging to the universe.

The setup phase of a new privacy transformation involves an ECDH key exchange protocol to establish pairwise secrets. In the prototype of PolicyCrypt, the privacy controller relies on the Bouncycastle [74] implementation of the ECDH key exchange using the secp256r1 curve to derive the 256-bit shared keys.

Remember, that a goal of the system is to minimize the time in between sending a commit and sending the transformation token due to robustness concerns. To achieve this and to reduce the latency, the privacy controller has an optimized implementation of the secure aggregation from Section 5.3.

The MPC protocol among the privacy controller is the most expensive part during this time, in particular for large universes. Therefore, the secure aggregation protocol described in Section 5.3 is implemented in Rust and is invoked from the privacy controller app via the Java Native Interface (JNI). Recall, the secure aggregation protocol requires many evaluations of a secure PRF. In PolicyCrypt, the PRF is instantiated with an AES block cipher using 256-bit keys. As a further optimization, the secure aggregation protocol uses, if available, the AES-NI instruction set extension.

7.3 Data Transformer

The prototype data transformer consists of two components, which are both built on top of Kafka Streams. The first component is the *transformation worker*, which is responsible for aggregating both the ciphertexts and the transformation tokens. Every privacy transformation has a separate set of transformation workers that perform the transformation. The number of workers per transformation depends on the complexity and size of the privacy transformation.

The second component is the *master*, which is shared across all transformations. The master component is responsible for the synchronization between different workers. The workload on the master for every transformation is small. However, for cases where the master becomes a bottleneck, it is also possible to introduce a new master and share the workload. Both Kafka Streams applications use in-memory data stores.

Every window of the privacy transformation has a status. Different windows within the same privacy transformation can have a different status. For example, while the data transformer aggregates ciphertexts in window i, window $i-1$ might be waiting for transformation tokens, and window $i-2$ might already be complete. Table 7.2 gives an overview of the possible window status in PolicyCrypt and their respective meaning.

Status	Description
`Open`	The observed stream time passed the start of the window.
`Staged`	The observed stream time passed the window end but the grace period might still be open. This status is the signal to collect commits from the privacy controller.
`Committed`	The data transformer does not accept further commits.
`Merged`	The universe members are fixed (i.e., the delta to the previous window is available).
`Closed`	The data transformer collected all transformation tokens and performed the privacy transformation.

Table 7.2: The different status of a window transformation.

In PolicyCrypt, the mechanism to scale a privacy transformation for large universes with high-velocity data is based on the concept of Kafka Streams tasks. PolicyCrypt supports multiple workers for the same universe, and a single one of these workers consists of several tasks. As a result, every privacy transformation has a set of tasks available for processing, which are possibly running on different worker nodes.

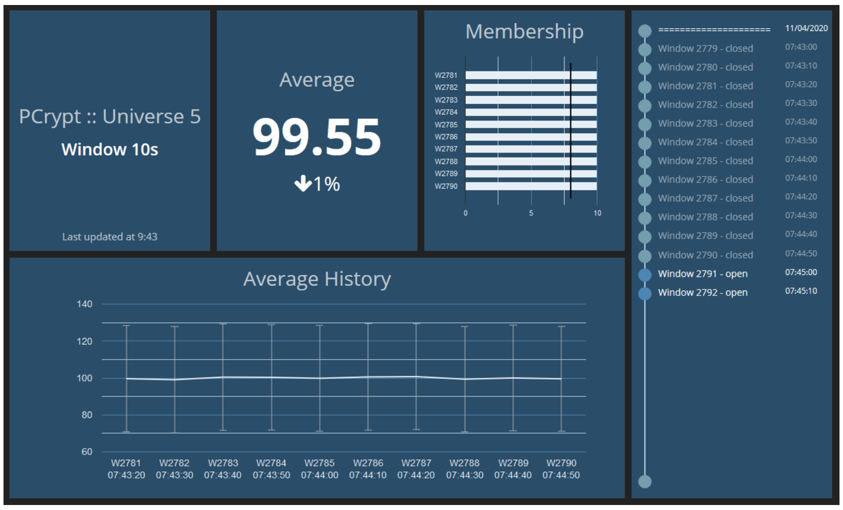
The idea is that tasks are working on different partitions of the same privacy transformation at the same time. A single task only observes and processes a fraction of the ciphertexts, commits, and transformation tokens. Based on this observed subset of data, a task assigns a status to each window. Due to this partitioning, the same window might not necessarily have the same status in all tasks. However, when a task updates the status of a window, the task informs the master node. The master node collects the status updates from the individual tasks and defines the global window status. Whenever the global status of a window changes, the master writes an update to the info topic of the privacy transformation.

To reduce complexity, the privacy controllers are unaware of the concept of tasks and their status. They only observe and react based on the global window status.

7.4 Policy Manager

In the prototype implementation of PolicyCrypt, a configurable Ansible [55] playbook imitates the role of the policy manager. Given the parameters from the streaming query, the playbook coordinates setting up the environment and running privacy transformations on a remote cloud infrastructure.

As part of the policy manager, PolicyCrypt provides a dashboard based on Smashing [5], which visualizes the current state of the transformation. Apart from meta-information about the transformation, the dashboard displays the universe membership over time. Further, the dashboard shows the universe's minimum size, required to satisfy all involved privacy policies. Finally, it displays the status of each window and the public result of the transformation.



```
======================        11/04/2020
Window 2779 - closed          07:43:00
Window 2780 - closed          07:43:10
Window 2781 - closed          07:43:20
Window 2782 - closed          07:43:30
Window 2783 - closed          07:43:40
Window 2784 - closed          07:43:50
Window 2785 - closed          07:44:00
Window 2786 - closed          07:44:10
Window 2787 - closed          07:44:20
Window 2788 - closed          07:44:30
Window 2789 - closed          07:44:40
Window 2790 - closed          07:44:50
Window 2791 - open            07:45:00
Window 2792 - open            07:45:10
```

Figure 7.2: PolicyCrypt Dashboard to visualize the state of a privacy transformation.

8 Evaluation

This chapter presents the evaluation of the PolicyCrypt platform. The objective of the experimental evaluation is to answer the following three questions: *(i)* How expensive is PolicyCrypt in terms of computation, bandwidth, and storage for the different components? *(ii)* How does PolicyCrypt scale? And *(iii)* Can the platform support practical applications?

8.1 Experimental Setup

The experimental evaluation consists of two parts. First, a set of microbenchmarks quantifies the cost of the different components. For the privacy controller, this includes an assessment of the performance benefits of using the new secure aggregation optimization compared to the Strawman and Dream (Section 5.3). Second, the evaluation quantifies the performance overhead of PolicyCrypt compared to a system with no encryption in an end-to-end benchmark to demonstrate the practicality of the privacy platform. For this end-to-end benchmark, each data producer has a separate privacy controller (i.e., a privacy controller does not represent multiple data producers), which represents the worst-case because this means that the number of privacy controller involved in the MPC protocol is equal to the number of data streams.

8.1.1 Methodology

The evaluation measures and quantifies the overhead of PolicyCrypt concerning the following metrics:

- *Computation time:* The average time required to perform a computation on a single thread of a CPU.
- *Throughput:* The number of records processed by a system component per unit of time.
- *Latency:* The time it takes a system to complete a request or query. For time-windowed streaming queries, which include a grace period to handle late-arriving records, the latency represents the time between the end of the grace period until the result of the query is ready.
- *Bandwidth:* The amount of data transmitted over the network.

- *Storage:* The amount of data kept on a machine.

The results from each experiment consist of an average over multiple iterations accompanied by the standard deviation.

8.1.2 Setup

The data producer, privacy controller, and data transformer of PolicyCrypt are Java applications that run on the JVM with Java 11.

Microbenchmark All microbenchmarks run on a single Amazon EC2 machine (m5.xlarge, 4 vCPU, 16 GiB, Ubuntu Server 18.04 LTS). The data producer microbenchmarks also run on a Raspberry Pi 3 (Model B, 1 GiB, Raspbian Buster 10) to analyze the performance on resource-constrained edge devices.

End-to-End benchmark All components of the end-to-end benchmark run on the AWS cloud. The central component of the end-to-end benchmark is Amazon MSK [15], which provides a Kafka cluster as a fully managed service. The Kafka cluster contains two broker nodes spread over two availability zones. The encryption in transit provided by Kafka is turned off because data is already encrypted by PolicyCrypt.

For data producer and privacy controller, the end-to-end benchmark uses a set of Amazon EC2 machines (m5.xlarge, 4 vCPU, 16 GiB, Ubuntu Server 18.04 LTS) and the data transformer runs on two Amazon EC2 machines (m5.2xlarge, 8 vCPU, 32 GiB, Ubuntu Server 18.04 LTS). The end-to-end benchmark groups the data producers and privacy controllers into partitions of up to 200 producers and controllers, respectively. A single producer- or controller-partition runs on one of the EC2 machines. The data transformer application spreads over the two EC2 machines using Kafka Streams.

Configuration Unless explicitly stated, PolicyCrypt uses an event with a single stream attribute x in the standard aggregation-based encoding $\vec{x} = [x, x^2, 1]$. Throughout the evaluation, secure aggregation via the PolicyCrypt optimization assumes up to half the participants are colluding (i.e., $\alpha = 0.5$), and the failure probability is below $\delta = 1 \times 10^{-7}$.

8.1.3 Application

The end-to-end benchmark provides two options to simulate different application use cases. The first option is to provide a separate file for each data producer that contains the application-specific data stream.

The second option to benchmark PolicyCrypt is a data stream generator that simulates a random data stream using a *Poisson process*. Data streams for different applications have distinct characteristics. For example, they differ in the interarrival time, which describes the time between two events.

A *Poisson process* is a stochastic process, where the sequence of interarrival times is a sequence of i.i.d random variables sampled from an exponential distribution with density $f_X(x) = \lambda exp(-\lambda x)$ for $x \geq 0$ [48]. The parameter λ controls the arrival rate of the data stream. Moreover, the mean of an exponentially distributed random variable is $1/\lambda$, which is the average time (e.g. number of seconds) between two events. For any time interval t, λt is the expected number of events within that interval. In the end-to-end benchmark, the data stream of an individual data producer is controlled by the mean $1/\lambda$.

8.2 Cost - Data Producer

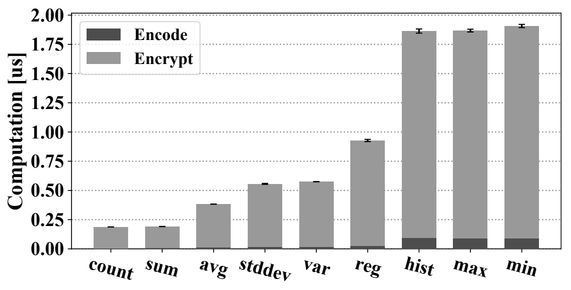
First, the evaluation quantifies the costs of running a data producer with PolicyCrypt. Recall that the data producer is unaware of the privacy layer and, hence, only has to encode and encrypt the data stream before forwarding it to the service. To demonstrate that the data producer is lightweight enough to run on resource-constrained edge devices, the section also reports the performance on a Raspberry Pi.

Computation Figure 8.1 shows the computation time of the data producer for different encodings. Remember that to support a broader range of statistical queries, PolicyCrypt encodes values as a vector (Section 5.2), which introduces additional overhead in encoding and encryption. For example, to calculate the variance, the encoding vector $\vec{x} = [x, x^2, 1]$ consists of three elements that have to be separately encrypted. The size of the encoding for the max, min, and histogram query depends on the number of buckets, which is by default set to 10.

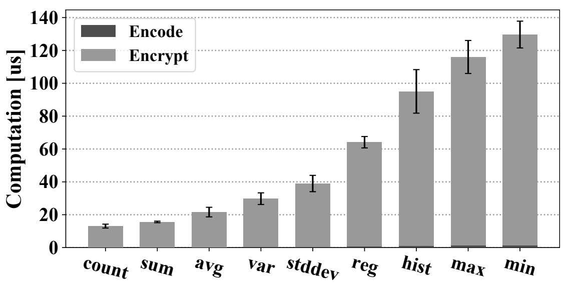
The cost of the different queries for encoding and encryption ranges from 0.19μs to 1.91μs on EC2, which corresponds to throughput in the range of 524k to 5.3m records per second. On the Raspberry Pi, the cost of encoding/encrypting the same queries is between 13.1μs and 129.7μs, which corresponds to a throughput in between 7.7k and 76.6k records per second.

Table 8.1 shows that the performance of the more complex bucket queries depends on the number of buckets. When varying between 10 and 1000 buckets, the cost for encoding/encryption takes between 1.9μs and 189.6μs on EC2. On the Raspberry Pi, this takes between 95μs and 11.8ms. Overall queries, the encryption amounts to 95% of the computation time on EC2 and more than 98% on the Raspberry Pi. This result means that in case the data producer's performance is not sufficient for encodings with a large number of buckets, a data producer can resort to hardware-accelerated AES to improve the throughput.

The data producer incurs a further small overhead to accommodate for the separation of the data plane and privacy plane. On every window border, the data producer must additionally submit a ciphertext, even if no data is present. This



(a) EC2 instance



(b) Raspberry Pi

Figure 8.1: The computation time of stream encoding and encryption in PolicyCrypt. The encodings for max, min, and hist use ten buckets.

	Time [μs]			Throughput [rec/s]		
Buckets	10	100	1000	10	100	1000
EC2	1.9	17.2	189.6	537k	58k	5.3k
Raspberry Pi	95	1.1k	11.8k	10.5k	916	84

Table 8.1: Comparison of the computation time and throughput of encoding and encryption for encodings with a varying number of buckets.

procedure increases the computational cost at a fixed rate e.g. for 10s windows; the data producer has to encode and encrypt one additional ciphertext, which consumes 0.1 of the throughput.

Overall, these results demonstrate that the cost in terms of computation is acceptable, and even constrained devices can support high throughput workloads as long as the number of buckets in the encodings remains in a manageable region.

Bandwidth Compared to plaintext, the aggregation-based encodings and the additional timestamp, required for decryption, introduce a ciphertext expansion which manifests in the required bandwidth of a data producer. Table 8.2 summarizes this ciphertext expansion for different encodings assuming both 64-bit values and timestamps. The expansion is in the range of 1.5x to 3.5x for encodings, not relying on buckets, which is reasonable.

For the other encodings (histogram, min, max), the expansion depends on the number of buckets. Using ten buckets leads to en expansion factor of 6x, which is still acceptable. However, for high-velocity data streams with many buckets, the bandwidth can become the bottleneck.

Storage Apart from the configuration that contains the information regarding where to forward the data stream and what the base window size is, a data producer has to store only the master secret (i.e., 16 bytes) for the stream encryption. Thus, overall the storage overhead on the data producer is negligible.

	sum	count	avg	var	hist	max	min	reg
$\|\vec{x}\|$	1	1	2	3	10	10	10	5
record size	24b (1.5x)	24b (1.5x)	32b (2x)	40b (2.5x)	96b (6x)	96b (6x)	96b (6x)	56b (3.5x)

Table 8.2: The PolicyCrypt record size expansion in comparison to plaintext. For the hist, min, and max function, the table shows the encoding size when using ten buckets.

8.3 Cost - Privacy Controller Single-Stream

PolicyCrypt introduces the privacy controller as a new component that has to be operated by the user or a trusted service. The cost of the privacy controller depends on the executed transformations on the service side. This section evaluates the costs of the efficient single-stream window transformations that do not require an MPC protocol. When a transformation involves only a single data producer (e.g. window transformation), the privacy controller can create the necessary tokens efficiently offline. The reason for this is that there are no adaptions necessary to react to different subsets of producers dropping out (Section 6.4).

Computation Due to the key canceling of the encryption scheme, the necessary time to create a token is independent of the number of in-range aggregated ciphertexts and, hence, also independent of the window size of the transformation (Section 5.1). On average, it takes a privacy controller 0.55μs to create a transformation token. As a consequence, the privacy controller can derive, on average, 1.8 million transformation tokens per second for arbitrary window sizes. This result shows that for a small window size of 20 seconds, a privacy controller can generate the necessary transformation tokens for more than a year within one second.

Bandwidth Each transformation token is an element of the additive homomorphic group of the stream encryption (Section 5.2). As a result, Table 8.2, which contains ciphertext sizes for different encodings, also shows the size of a respective transformation token. A privacy controller must submit one transformation token for every window of a privacy transformation. Considering the example from above with a window size of 20 seconds and under the standard encoding $\vec{x} = [x, x^2, 1]$, the tokens to facilitate the privacy transformation for a year require approximately 63 MB of bandwidth, which should pose no problem.

Storage Apart from the negligible storage requirements for the privacy policy and the transformation plan, the privacy controller only needs to store the master key (i.e., 8 bytes) that the associated data producer uses for encryption. Thus, the storage footprint of the privacy controller for single-stream transformations is small.

8.4 Cost - Privacy Controller Multi-Stream

After showing the small overhead of a privacy controller in the single-stream case, this section analyses the performance of the privacy controller in the multi-stream case. Recall that for transformations over multiple streams, the privacy controller has to be online and participate in the optimized secure aggregation protocol with the other privacy controllers (Section 5.3).

The evaluation measures the cost of running the PolicyCrypt secure aggregation protocol for a different number of privacy controllers and over multiple windows and compares it against the Strawman and Dream [12] protocols (Section 5.3).

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, in which the privacy controllers collaborate to create the required transformation tokens at the end of each window.

8.4.1 Setup Phase

The setup phase of a privacy transformation consists of two steps. First, fetching all public-keys from a central PKI, and second performing the ECDH key exchanges to establish the shared secrets. Table 8.3 shows the computation and bandwidth costs for running the setup phase.

Considering all privacy controllers, the setup phase overhead increases quadratically in the number of privacy controllers (i.e., total $\mathcal{O}(N^2)$) However, from the perspective of a single privacy controller, the overhead increases only linearly.

Privacy Controllers	2	10	100	1k	10k	100k
Bandwidth	91 B	819 B	9.0 KB	91 KB	910 KB	9.1 MB
Bandwidth Total	182 B	8.2 KB	901 KB	91 MB	9.1 GB	910 GB
Shared Keys	32 B	288 B	3.2 KB	32 KB	320 KB	3.2 MB
ECDH	0.25 ms	2.2 ms	25 ms	249 ms	2.5 sec	25 sec
ECDH Total	0.5 ms	22 ms	2.5 sec	4 min	7 h	693 h

Table 8.3: The cost of the setup phase for different number of privacy controllers. The total bandwidth and ECDH results sum up the cost involved for all involved privacy controllers of the universe. The other results show the costs of participation for a single privacy controller.

Bandwidth PolicyCrypt uses the elliptic-curve Diffie–Hellman (ECDH) key exchange protocol on the secp256r1 curve to derive 256-bit shared keys (Section 7.2). In a production system, the PKI would hand out signed certificates containing the public-key. However, the size of a public-key certificate varies depending on the data fields.

For simplicity, the bandwidth estimates in this analysis only consider the 91 bytes required for a public-key from the Java Bouncycastle security implementation [74]. Nevertheless, these bandwidth estimates provide a rough estimate for the requirements of a privacy controller in a production system.

The bandwidth requirements range from less than 1KB for ten privacy controllers up to 9.1MB for 100k privacy controllers. Considering that the setup phase has to be performed only once in the lifetime of a privacy transformation, it is reasonable even for 100k participants.

Computation Given the public-key of another identity, establishing a single shared secret using the ECDH key exchange protocol takes on average 0.25ms. This measurement allows extrapolating the expected time it takes to set up larger universes. For example, a privacy controller needs approximately 2.5s to establish all shared secrets in a universe with 10k privacy controllers and 25s with 100k privacy controllers. The overhead is modest, taking into account the infrequent execution of the setup phase. However, if necessary, the computation time can be further reduced by leveraging more than a single core because the different key exchanges are independent and thus highly parallelize.

Storage In addition to their private-key (i.e., 150 bytes), privacy controllers only need to store the established shared secrets of the current privacy transformation because after verifying the certificates and performing the ECDH key exchange, the privacy controller can discard the certificate including the public-key. Each shared key requires 32 bytes, which results in 320 KB and 3.2MB storage, respectively, for 10k and 100k shared keys. However, given the tradeoff between bandwidth, computation, and storage, it can make sense to preserve the certificates and shared secrets from previous setup phases on disk. This caching allows reusing work, in the case that at a later stage, the two privacy controllers end up again in the same universe. Overall, the storage requirements of the setup phase are reasonable.

8.4.2 Privacy Transformation Phase

After quantifying the cost of the setup phase of a new privacy transformation, this part focuses on the complexity of the recurring creation of transformation tokens during the privacy transformation phase. The target is to evaluate and analyze the multi-round computational advantage of the optimized secure aggregation protocol in comparison to the alternatives Strawman (96x) and Dream (55x).

For being robust in the face of dropouts, all secure aggregation protocols require interaction between privacy controller and service (Section 6.4). When the service initiates a two-stage interaction (i.e., at the end of a window), the available privacy controllers commit to sending a transformation token. Subsequently, the service determines the current universe based on the collected commits and the state of the data streams and informs the privacy controller about the changes in respect to the previous window.

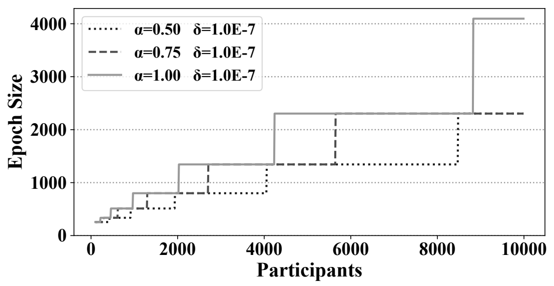
To obtain a detailed understanding of the performance improvements of the secure aggregation protocol optimization, the following paragraphs compare the computation for a single round, provide an analytical comparison, consider the performance over time, and analyze the performance in the face of dropouts.

Computation: Round While in the Strawman and Dream, all rounds are identical (i.e., every round performs the same computation), the PolicyCrypt optimization distinguishes between two types of windows. Recall that the optimization groups multiple rounds into epochs. The first round of every epoch is different from the other rounds because the protocol performs additional work to improve the performance of the remaining rounds. The high-level idea of the optimization is that after a few rounds, the additional work performed at the beginning of an epoch is amortized, and hence the overall cost of the computation reduces significantly in the long run.

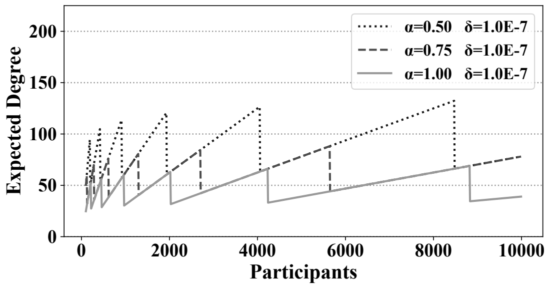
Three factors determine the effectiveness of such an optimization: *(i)* What is the cost of the first round of an epoch? *(ii)* What is the cost of a later round in the epoch? Furthermore, *(iii)* How frequently does the epoch change?

The evaluation starts with the third question. Figure 8.2a shows the number of rounds per epoch (i.e., epoch size) for different parameter pairs (α, δ) as a function of the number of participants. The parameter α controls the percentage of non-colluding clients, while the parameter δ is an upper bound on the failure probability of the protocol (Section 5.3).

In the standard configuration (i.e., $\alpha = 0.5$, $\delta = 1 \times 10^{-7}$), one epoch has between 256 and 2304 rounds when varying the number of privacy controllers in between 100 and 10k. The epoch size increases in the number of participants in the form of a step function. The reason for this lies in the construction of the secure aggregation graphs because the Erdős–Rényi graph construction requires that the segment size must be a power of two. At the step, there are enough participants to choose the next higher power of two for the segment size while still satisfying the error bound (details in Section 5.3).



(a) Epoch Size



(b) Expected Degree

Figure 8.2: PolicyCrypt secure aggregation epoch size and expected degree for different number of participants.

The next part of the evaluation addresses questions *(i)* and *(ii)* by quantifying the average time of a single round of the secure aggregation protocol. The comparison of the average round times provides insight into the effectiveness of the optimization. The evaluation varies the number of privacy controller from 100 to 10k; Table 8.4 shows the results. For PolicyCrypt, the table distinguishes between the average time of the first round and later rounds in the epoch.

The average time for the non-setup rounds is consistently the lowest. All round-times increase with more participants (i.e., the universe size), except for these later rounds in an epoch that remain below 20μs. This is because the universe size has little influence on the operations required in an optimized round. This phenomenon is described in more detail at a later stage of this evaluation.

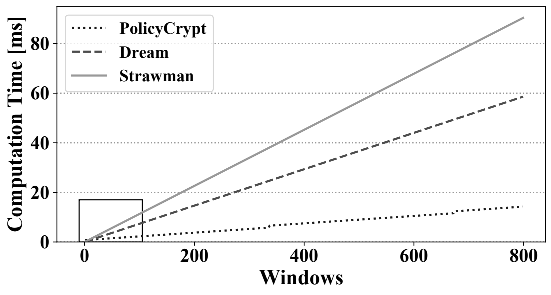
The round time of Dream is consistently lower than the round time of the Strawman. The difference is in-between 4μs for 100 participants and 978μs for 10k participants. This discrepancy is because, for the standard aggregation-based encoding, the Strawman requires both more PRF and ADD operations.

The cost to set up an epoch in PolicyCrypt is high in comparison to a round in Strawman and Dream. For example, for 500 participants, it takes 782μs to setup an epoch for PolicyCrypt compared to a regular round that takes 113μs and 73μs for the Strawman and Dream. However, privacy transformations are designed to run for multiple windows, and hence the cost for the first round of an epoch is not the crucial factor. Instead, for the considered workloads, it is important that over a long series of windows, the computation time remains low.

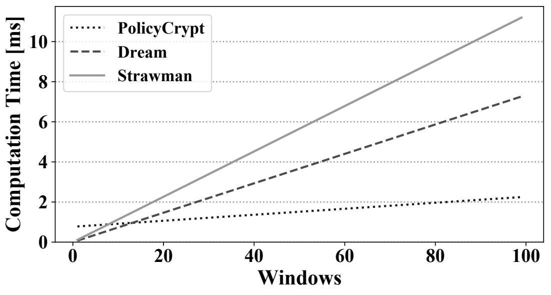
Round	Participants					
	100	200	500	1k	10k	
PolicyCrypt [1st]	477	482	782	1222	7740	μs
PolicyCrypt [2^{+}]	14	14	15	15	20	μs
Strawman	25	47	113	222	2234	μs
Dream	21	33	73	134	1256	μs

Table 8.4: Computation cost of a single round. For PolicyCrypt, the table distinguishes between the first round of an epoch [1st] and all the remaining rounds of the epoch [2^{+}].

Figure 8.3 shows the cost associated with running the three approaches with 500 participants over a series of 800 windows. The cumulative computation time of PolicyCrypt forms a piecewise linear function with steps at the beginning of each epoch due to the two different types of rounds. In the beginning, both the Strawman and Dream perform better than PolicyCrypt (Fig. 8.3b). This result is because of the higher costs incurred in the first round to set up the epoch. However, after repeating the protocol for 15 rounds, the cumulative latency shows that the PolicyCrypt optimization outperforms both the Strawman and Dream.



(a) Cumulative Latency



(b) Zoomed Cumulative Latency

Figure 8.3: Comparison of secure aggregation approaches over multiple windows for 500 participants.

Computation: Analytical The experiments underline what the analytical analysis of the three approaches in Table 8.5 suggests. The three considered secure aggregation protocols mainly consist of a sequence of PRF evaluations and additions, and the degree of a node in the secure aggregation graph determines the length of a sequence (Section 5.3). Hence, for an encoding vector with $|v|$ elements, over a period of W rounds, with N participants and expected node degree $\ell << N$ (Dream and PolicyCrypt), the optimized approach of PolicyCrypt must evaluate the PRF only $N - 1 + W \cdot \ell \cdot |v|$ times compared to $W \cdot (N-1) \cdot |v|$ and $W \cdot (N-1) + W \cdot \ell \cdot |v|$ in the Strawman and Dream respectively. Note that the large N is only an additive factor in PolicyCrypt while it is a multiplicative factor in both the Strawman and Dream. The only multiplicative factor in PolicyCrypt is the expected node degree ℓ.

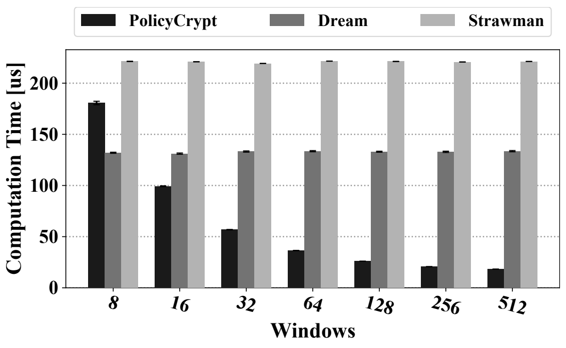
Approach	PRF ops	ADD ops
Strawman	$W \cdot (N-1) \cdot \lvert v \rvert$	$W \cdot (N-1) \cdot \lvert v \rvert$
Dream	$W \cdot (N-1) + W \cdot \ell \cdot \lvert v \rvert$	$W \cdot \ell \cdot \lvert v \rvert$
PolicyCrypt	$N - 1 + W \cdot \ell \cdot \lvert v \rvert$	$W \cdot \ell \cdot \lvert v \rvert$

Table 8.5: Comparison of the expected number of PRF and ADD operations in the three secure aggregation protocols over W rounds, for N participants, ℓ expected node degree, and an encoding vector with $|v|$ elements.

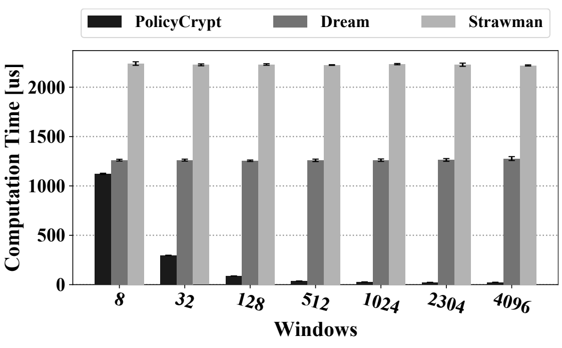
As a next step, the goal is to show that the expected node degree ℓ is, in fact, much smaller than the number of participants (i.e., $\ell << N$). Figure 8.2b provides an overview of the expected node degree ℓ in a single secure aggregation graph for different parameter pairs (α, δ) as a function of the number of participants. From a performance perspective, a lower node degree results in better performance because less hiding nonces need to be derived and added. However, a lower node degree increases the probability that the secure aggregation graph is disconnected. Hence, the error bound δ determines a lower bound for the expected node degree. In the standard configuration, the expected degree ranges from 50 up to 133, which is considerably lower than any N. Note that the zigzag artifacts in the figure stem from the graph construction process, similar to the steps in Figure 8.2a.

Computation: over Time The above analysis has demonstrated that when varying the number of participants in-between 100 and 10k, the three key metrics of the PolicyCrypt optimization behave as follows: *(i)* The cost to create a transformation token in the first round is in the range of 477μs and 7740μs which includes the time to set up the later rounds, *(ii)* The cost to create a transformation token during an epoch is between 14μs and 20μs, and finally *(iii)* The epoch size is in between 256 and 2304 rounds.

Now the goal is to quantify the overall performance gain of the secure aggregation optimization compared to the Strawman and Dream. Figure 8.4 compares the average computation cost of the three approaches over an increasing number of windows (i.e., rounds), for aggregation over 1k and 10k participants. Already for 8 and 16 windows for 10k and 1k participants, respectively, the PolicyCrypt optimization is more efficient on average. This result means that a privacy transformation does not have to run for a complete epoch to benefit from the optimization; a few windows are sufficient. While the average time remains constant in the Strawman and Dream (because all their rounds are identical), the average time per round decreases in PolicyCrypt as the cost of the setup round becomes less significant.



(a) 1k Participants



(b) 10k Participants

Figure 8.4: Comparison between secure aggregation methods for different number of windows.

Figure 8.5 compares the average computation cost of the three approaches for a different number of participants over an entire epoch. Note that using the epoch size in Figure 8.5 as the number of rounds is the best case for the PolicyCrypt optimization because all the initial setup work was "reclaimed". Figure 8.2a lists the associated epoch sizes for the different numbers of participants; however, this is not the essential point. The essential point is that both the Strawman and Dream accumulate considerable cost over time in particular if many participants are involved, while PolicyCrypt remains efficient.

For 10k participants, PolicyCrypt can maintain a throughput of 43k transformation tokens per second compared to 449 and 792 tokens per second in the Strawman and Dream. Recall the example from Section 8.3, where the evaluation showed that for a window size of 20s, a privacy controller for a single stream could create all the transformation tokens required for a year within one second. To create the same tokens for a multi-stream transformation involving 10k participants under the assumption of no dropouts, it takes 58 min and 33 min with the Strawman and Dream and only 36s with the PolicyCrypt optimization.

To conclude, the extensive analysis of the three approaches' computation time demonstrated that the privacy controller could create transformation tokens efficiently, even for transformations involving 10k participants.

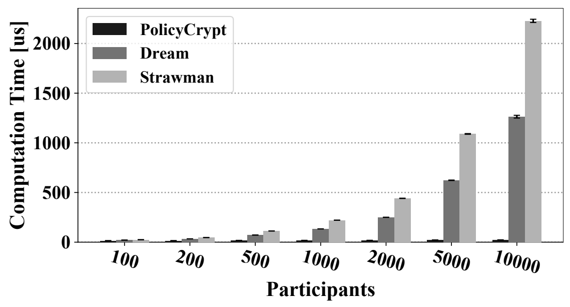


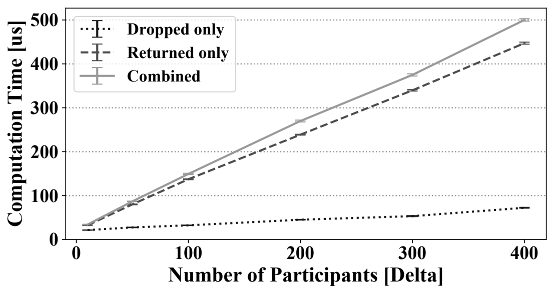
Figure 8.5: Average computation time over an entire epoch.

Computation: with Dropouts The transformation plan contains the initial list of participants. These members form a universe for a privacy transformation (Section 6.4). However, over time the universe can change due to members dropping out or previously dropped members returning. There are two possibilities regarding how to handle these dropouts: *(i)* The first round builds the graph for all participants from the transformation plan, including the dropped. This option has the advantage that in case a participant returns during an epoch, it is already known whether they share a mask. However, if a participant is permanently dropped, building these edges is an unnecessary overhead. Note that this possibility is equivalent to handling the event that participants drop during an epoch, and hence later rounds must ignore their edges in the graphs. *(ii)* The second possibility is to consider only the current participants in the first round of an epoch when building the graphs. In case a participant returns during an epoch, the existing graphs can then be extended.

Figure 8.6 shows the cost of different numbers of participants dropping and returning using possibility *(ii)*. For a dropped participant, the protocol needs to check if there is an edge in the respective round. If this is the case, the shared mask is not derived and added to the transformation token, which is the cause of the small difference (20.3μs to 27.5μs) to adapt for a dropped node with an edge (E) and without an edge (NE). For a window with 50 of the 1000 participants dropping out, the protocol requires 27.5μs compared to the required 15μs for no dropouts from Table 8.4. In comparison to the 222μs and 134μs round times of the Strawman and Dream, this is acceptable.

Furthermore, Figure 8.6 shows that the graph expansion required for returning participants comes at a cost that is linear in the number of returned members. Note that this is a computational cost that the privacy controller saved during the first round of the epoch. For example, when 300 of 1000 participants return in a round, it takes 340μs to extend the graphs. The cost of handling returned and dropped nodes (i.e., combined) is additive.

Overall, also the cost of handling a changing universe is modest on the privacy controller.



Δ	Returned		Dropped		
	E	NE	E	NE	
10	31.9	32.3	19.2	21.4	μs
50	83.8	80.0	20.3	27.5	μs

Figure 8.6: Time needed to adapt PolicyCrypt optimization to a changing universe of initial size 1000. Delta stands for the number of participants that dropped or returned in the round respectively. In the combined case, delta members dropped and delta other members returned. The table compares the impact of a delta depending on whether a neighbor (E), or a node without an edge (NE) is affected.

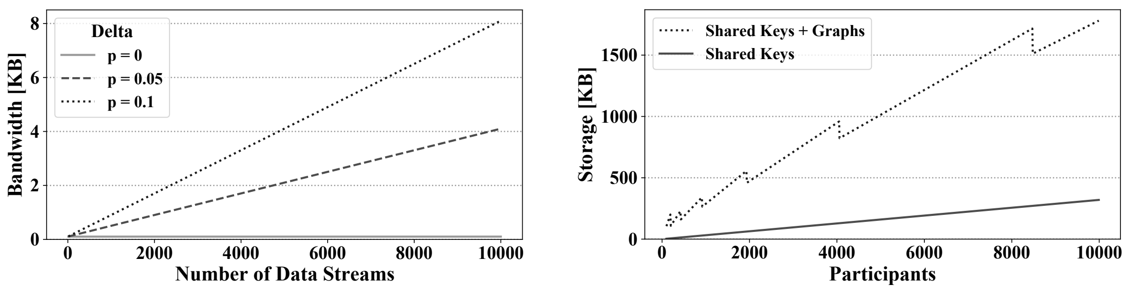
Bandwidth The required bandwidth for multi-stream transformation tokens is identical to the bandwidth costs for single-stream transformation tokens because the tokens are also an element of the additive homomorphic group of the stream encryption (Section 5.2). Table 8.2 shows the size of a transformation token for different encodings. A token in the standard encoding $\vec{x} = [x, x^2, 1]$ requires 40 bytes.

In addition to the bandwidth required for submitting a transformation token, a privacy controller involved in multi-stream transformation must also account for the bandwidth during the interaction.

Many MPC protocols suffer from severe bandwidth overheads [43]. However, this is not the case for the secure aggregation protocol of PolicyCrypt. The only variable-sized component of the bandwidth is the size of the delta set Δ (i.e., number of dropped and returning participants per window), which the evaluation models as a fixed percentage p_Δ of the total number of participants in the analysis. Figure 8.7a shows the bandwidth requirements for three different percentages p_Δ (0, 0.05, 0.1) as a function of the number of involved data streams.

The secure aggregation protocol for up to 1k participants requires less than 1KB bandwidth, even under the assumption of a 10% fluctuation p_Δ, which is extreme for the considered scenario. The reason for 10% being extreme is that PolicyCrypt only treats a data stream as dropped if either the data producer or the privacy controller is unavailable at the end of a window. Temporary offline phases during a window do not cause a dropout in the protocol.

The bandwidth also remains reasonable when scaling the protocol to even larger aggregation groups. For example, for 10k participants of which 10% drop or return, the required bandwidth is only 8.1KB.



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

(b) Storage for privacy controller during privacy transformation phase.

Figure 8.7

Storage For the privacy transformation phase, the privacy controller needs to store the shared keys (i.e., 32 bytes per key) and all the secure aggregation graphs of the epoch. Figure 8.7b shows the storage overhead introduced by the secure aggregation optimization in comparison to the Strawman and Dream that only need to store the shared keys. For 10k privacy controllers, an individual participant requires less than 2 MB to store the shared keys and the over 2k secure aggregation graphs. As a result, even though the overhead increases in the number of privacy controllers, the total storage remains acceptable.

Note that if storage becomes a problem despite the moderate expansion (e.g. because a single privacy controller is responsible for many data streams), a privacy controller can always resort to storing only a fraction of the secure aggregation graphs and recalculate the next batch of graphs at the required time. This procedure remains reasonable because the additional computational costs for creating the graphs are amortized after a few windows.

8.5 Cost - Data Transformer

Finally, the evaluation investigates the overhead of running a privacy-preserving data transformer. The data transformer runs streaming jobs on a cluster of nodes, which usually offers plenty of resources. In PolicyCrypt, the data transformer relies on the Apache Kafka architecture, which is designed to run at scale. Deployed Kafka clusters are capable of ingesting 800 billion messages per day, which amounts to over 175 terabytes of data while at the same time being able to supply 650 terabytes to consumers [78]. Nevertheless, the following paragraphs look at the overhead that PolicyCrypt introduces.

Computation The data transformer needs to maintain a structure to track dropped data producers and collect commits from privacy controllers. Additionally, ciphertexts within a privacy transformation must be aggregated separately per data producer to be able to react to dropouts in a later stage of the window. At the end of a window, the data transformer needs to sum the ciphertext aggregates together with the transformation tokens for decryption of the privacy policies' conform result. However, since all these overheads increase only in the number of privacy controllers or data streams rather than in the number of ciphertexts, the computation overhead is acceptable.

Bandwidth During the setup phase of a new privacy transformation, all involved privacy controllers need to download the certificates of all participating identities. Thus, the bandwidth overhead for a server increases quadratically in the number of privacy controllers. Section 8.4 of the evaluation already considered the bandwidth requirements for the setup phase from the viewpoint of a single privacy controller. Now the goal is to examine the overall requirements. Table 8.3 shows the total bandwidth for different numbers of privacy controllers. For 10k privacy controllers, the transformation manager or a PKI needs to distribute 9.1 GB of public-keys, and for 100k, this increases to 910 GB; for comparison, 910 GB would be about the same bandwidth Netflix requires for distributing one hour of an HD video stream to 300 users [72]. However, if this exceeds the available resources, Section 6.4 outlines an idea of how to reduce this overhead.

During the privacy transformation phase, the data transformer receives a commit and a transformation token from every involved privacy controller, and the data transformer needs to distribute the delta update of the universe (i.e., who dropped out and who returned compared to the previous window). Figure 8.7a shows the bandwidth requirements for a single privacy controller. By extrapolating these numbers for 10k privacy controllers and assuming a fluctuation of 10%, a data transformer requires 81 MB of bandwidth to perform the transformation on a single window. This bandwidth is negligible in comparison to the bandwidth that the transport of data requires.

As discussed in Section 8.2, the aggregation-based encodings introduce a ciphertext expansion of a factor 2.5x for the standard configuration. This expansion manifests in the bandwidth requirements of a data transformer. However, the overall bandwidth overhead of PolicyCrypt should remain manageable for a data transformer, particularly by leveraging Kafka's architecture, which allows partitioning the load over a cluster of machines.

Storage The data transformer has two overheads. First, it must aggregate the ciphertexts within a window per data producer. Second, the data transformer requires an additional structure to track active data producers and privacy controllers. However, in total, the overhead is small.

8.6 End-to-End Benchmark

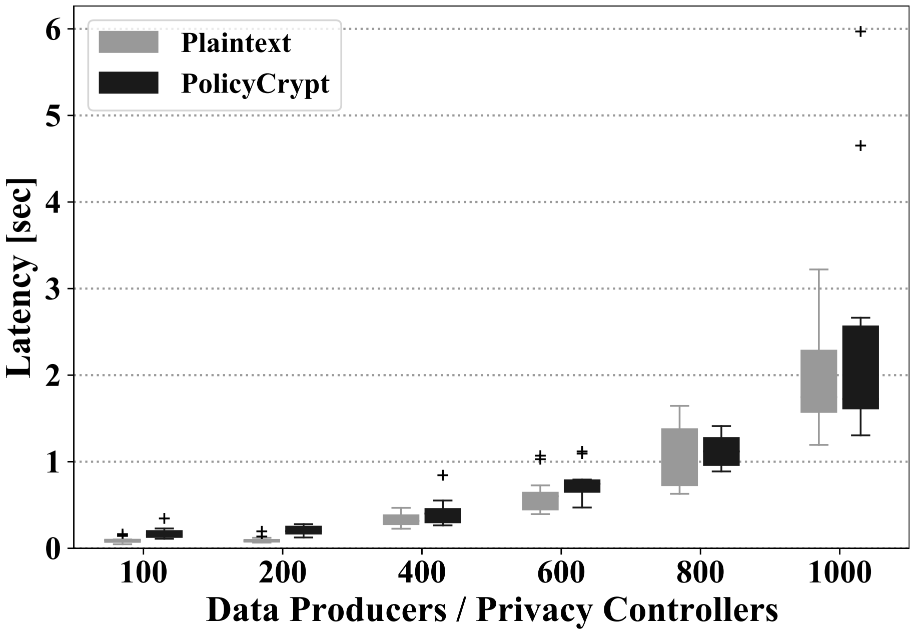


Figure 8.8: Boxplot of the latencies of the two systems Plaintext (no encryption) and PolicyCrypt for different number of participants. The latency measures the time after the grace period of a window is over until the result of the transformation is available.

The goal of this section is to demonstrate, based on an end-to-end benchmark, that PolicyCrypt enables privacy transformations in real-time on a realistic application scenario. As a baseline, the evaluation uses a plaintext version to run the same application with no encryption. The plaintext version sends unencrypted values to the data transformer (no aggregation-based encodings). The data transformer continuously forms windowed aggregates for each data stream before adding them across the streams. This structure allows the plaintext version to scale to large workloads. Note that there is no privacy controller involved in the plaintext version because there are no privacy policies to enforce.

The data producers create random data streams with the stream generator described in Section 8.1. Recall that the stream generator uses a Poisson process to emulate a data producer workload in an application. The underlying exponential distribution has a mean $1/\lambda$ of 0.1. This configuration results in an average of 10 records per second from each data producer. The data producers use the standard aggregation-based encoding $\vec{x} = [x, x^2, 1]$. All experiments aggregate data in 10-second windows and use a grace period of 5 seconds to account for late-arriving events. Considering the case study from Section 6.2, this corresponds to the following query:

```
CREATE STREAM HeartRate (heartrateAvg, heartrateStdDev) AS
  SELECT AVG(heartrate), STDDEV(heartrate)
  WINDOW TUMBLING (SIZE 10 SECONDS, GRACE PERIOD 5 SECONDS)
  FROM MedicalSensor BETWEEN 100 AND 1000
  WHERE region = 'California' AND age = 'old'
```

For PolicyCrypt, each data producer has a separate privacy controller. This scenario is the worst case because the MPC protocol involved in creating the transformation tokens involves the maximum number of privacy controllers. Figure 8.8 shows the results of the end-to-end benchmark for transformations in the range of 100 to 1000 participants. For privacy transformations in this range, the average latency varies between 174ms and 2.4 seconds in PolicyCrypt compared to 90ms and 1s in the plaintext baseline.

The increasing latencies in the results for 800 and 1k participants indicate that for both PolicyCrypt and plaintext, the stream processor that runs on two machines starts to saturate. For up to 800 participants, the difference in average latency between the two approaches is in-between 40ms and 143ms. In the experiment involving 1k participants, two outliers in PolicyCrypt cause the difference to increase to 479ms. Nevertheless, these results show that the overhead that PolicyCrypt introduces compared to plaintext is small.

9 Conclusion

The ever-growing demand for collecting and analyzing data is not expected to slow down anytime soon. Organizations across all sectors recognize that data is a valuable asset that has both enormous value to their businesses and a tremendous potential to solve some of the hardest challenges societies are facing today. Think, for example, of the COVID'19 pandemic, which started at the time of writing this thesis. Data is the most crucial asset to control and contain the pandemic. Governments are rushing to collect personal and sensitive data (e.g. location, people interactions) to understand its spread and health impact. However, as we accumulate more and more sensitive data, the issue of individual privacy is becoming more urgent. Adequately addressing privacy in the current complex computing landscape is vital if we aim to interconnect data and unlock its potential to the fullest while respecting the individual's privacy. The path for achieving this implies the need for developing privacy tools that are easily blended in existing data analytics systems. This thesis addresses the aforementioned pressing challenge with a new data privacy platform design.

This thesis presents PolicyCrypt, a new privacy platform design for distributed unbounded data that follows a user-centric model to privacy, allowing users to articulate their preferences in the form of a simple privacy policy. Based on these privacy policies, PolicyCrypt provides a framework to find a match between the privacy interests of users and the objectives of a service provider. Instead of relying on trust and manual compliance, PolicyCrypt proposes an end-to-end approach that leverages cryptographic techniques to automatically enforce privacy-preserving transformations on streaming data. Due to a clear separation of the data and privacy planes, these transformations entail no more than a minor outlay for data sources, which allows existing stream processing pipelines with resource-constrained data producers to integrate PolicyCrypt seamlessly.

The evaluation based on a prototype implementation indicates that the overhead of PolicyCrypt is modest (2.4 sec compared to 1.9 sec for 1k participants), which enables large-scale low-latency data stream analytics on top of the platform. Furthermore, the evaluation of the optimized secure aggregation protocol over multiple rounds demonstrates an increase in performance by a factor of 55 over existing protocols.

In addition to the research questions addressed in this thesis, the architecture of PolicyCrypt raises some new challenges that we believe are interesting to pursue in further research. The following section briefly highlights some of these challenges and discusses new ideas that originate from this work.

9.1 Future Work

- *Privacy Policy Extensions:* PolicyCrypt supports privacy policies based on aggregation and perturbation. Further research could investigate whether cryptographic techniques are suitable for enforcing more complex privacy policies beyond aggregation and perturbation. The constrained privacy policies from Section 4.3 can serve as a starting point therefor.
- *Privacy Policy Ramifications:* The user-centric model of PolicyCrypt gives users more control over their data by allowing them to express their preferences. However, the ramifications of differential privacy budgets (ϵ, δ) or different aggregation sizes within privacy policies often remain hazy, especially in conjunction with other data. These doubts pose the question of whether these privacy controls should be exposed to the user or be left to independent privacy entities.
- *Protection against Users:* The work in this thesis investigates how users can cooperate to protect each other's privacy against an untrusted server. Even though the privacy of an individual user remains protected, the consequences of the PolicyCrypt approach are that a single malicious user can interfere with privacy transformations and manipulate the result. Further work could investigate the integration of protection mechanisms similar to what is done in other systems [32]. As a result of this and in combination with the idea from Section 6.4, privacy transformation might be able to include data from even more streams.
- *Incentive for Users:* PolicyCrypt implicitly assumes that users have an incentive to share their data with service providers. This is motivated by the presumption that the service provider uses the data to offer a service that directly benefits the user. Further research could investigate if these incentives are sufficient, or if some form of data market is required to facilitate this exchange within the boundaries of privacy policies.

Bibliography

[1] Apache Kafka Intro. Online: https://kafka.apache.org/intro.

[2] Apache Kafka Powered By. Online: https://kafka.apache.org/powered-by.

[3] The california consumer privacy act of 2018. Online: https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375.

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

[5] Smashing dashboard. Online: https://smashing.github.io/.

[6] Learning Statistics with Privacy, aided by the Flip of a Coin . Google, Online: https://ai.googleblog.com/2014/10/learning-statistics-with-privacy-aided.html, 2014.

[7] Apple's Differential Privacy Is About Collecting Your Data, But Not Your Data. WIRED, Online: https://www.wired.com/2016/06/apples-differential-privacy-collecting-data/, 2016.

[8] Testing Privacy-Preserving Telemetry with Prio . Mozilla, Online: https://hacks.mozilla.org/2018/10/testing-privacy-preserving-telemetry-with-prio/, 2018.

[9] Flink Documentation: Security . Flink, Online: https://ci.apache.org/projects/flink/flink-docs-stable/ops/security-ssl.html, 2020.

[10] Kafka Documentation: Security . Kafka, Online: https://kafka.apache.org/documentation/#security_overview, 2020.

[11] Spark Documentation: Security . Spark, Online: https://spark.apache.org/docs/latest/security.html#authentication-and-authorization, 2020.

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

[13] K. Agarwal. The Rise Of Streaming Apps, And What Developers Need To Consider . Forbes, Online: https://www.forbes.com/sites/forbestechcouncil/2018/09/26/the-rise-of-streaming-apps-and-what-developers-need-to-consider/, 2018.

[14] Airbnb. Privacy policy airbnb. Online: `https://www.airbnb.com/terms/privacy_policy?locale=en`.

[15] Amazon. Managed streaming for apache kafka. Online: `https://aws.amazon.com/msk/`.

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

[17] E. Birrell and F. B. Schneider. Fine-grained user privacy from avenance tags. 2014.

[18] T. E. D. P. Board. Eu data protection rules. Online: `https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules/eu-data-protection-rules_en`.

[19] T. E. D. P. Board. Infographic: Gdpr in numbers. Online: `https://ec.europa.eu/commission/sites/beta-political/files/infographic-gdpr_in_numbers_1.pdf`.

[20] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *ACM CCS*, 2017.

[21] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. 2020.

[22] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on theory of computing*, STOC '13, pages 575–584, New York, NY, USA, 2013. ACM. event-place: Palo Alto, California, USA.

[23] L. Burkhalter, A. Hithnawi, A. Viand, H. Shafagh, and S. Ratnasamy. Timecrypt: Encrypted data stream processing at scale with cryptographic access control. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 835–850, Santa Clara, CA, Feb. 2020. USENIX Association.

[24] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: data structures and implementation. In *NDSS*, 2014.

[25] C. Castelluccia, A. C. Chan, E. Mykletun, and G. Tsudik. Efficient and Provably Secure Aggregation of Encrypted Data in Wireless Sensor Networks. *ACM TOSN*, 5(3):20, 2009.

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

[27] Y. Chen, A. Machanavajjhala, M. Hay, and G. Miklau. Pegasus: Data-adaptive differentially private stream processing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1375–1388, New York, NY, USA, 2017. ACM.

[28] L. Cheng, F. Liu, and D. D. Yao. Enterprise Data Breach: Causes, Challenges, Prevention, and future Directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(5), 2017.

[29] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. *SpringerLink*, pages 3–33, Dec. 2016. tex.publisher: Springer, Berlin, Heidelberg.

[30] A. Cohen, S. Goldwasser, and V. Vaikuntanathan. Aggregate Pseudorandom Functions and Connections to Learning. In *TCC*, 2015.

[31] Confluent. Streaming Data - A Complete Guide. Confluent, Online: `https://www.confluent.io/learn/data-streaming/`, 2020.

[32] H. Corrigan-Gibbs and D. Boneh. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *USENIX NSDI*, 2017.

[33] G. Danezis, C. Fournet, M. Kohlweiss, and S. Zanella-Béguelin. Smart meter aggregation via secret-sharing. In *Proceedings of the First ACM Workshop on Smart Energy Grid Security*, SEGS '13, page 75–80, New York, NY, USA, 2013. Association for Computing Machinery.

[34] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, Sept. 2006.

[35] M. Douriez, H. Doraiswamy, J. Freire, and C. T. Silva. Anonymizing nyc taxi data: Does it matter? In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 140–148, 2016.

[36] Y. Duan, N. Youdao, J. Canny, and J. Zhan. P4p: Practical large-scale privacy-preserving distributed computation robust against malicious users. pages 207–222, 09 2010.

[37] L. Ducas and D. Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 617–640, 2015. tex.organization: Springer.

[38] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, Aug. 2014.

[39] T. Elahi, I. Goldberg, and G. Danezis. Privex: Private collection of traffic statistics for anonymous communication networks. 11 2014.

[40] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

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

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

[43] D. Evans, V. Kolesnikov, M. Rosulek, et al. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.

[44] T. A. S. Foundation. Apache avro data serialization system. Online: `https://avro.apache.org/`.

[45] T. A. S. Foundation. Apache flink. Online: `https://flink.apache.org/`.

[46] T. A. S. Foundation. Apache kafka. Online: `https://kafka.apache.org/`.

[47] T. A. S. Foundation. Apache storm. Online: `https://storm.apache.org/`.

[48] R. Gallager. Discrete stochastic processes. Online: `https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-262-discrete-stochastic-processes-spring-2011/course-notes/MIT6_262S11_chap02.pdf`.

[49] W. C. Garrison, A. Shull, S. Myers, and A. J. Lee. On the Practicality of Cryptographically Enforcing Dynamic Access Control Policies in the Cloud. In *IEEE Security and Privacy*, 2016.

[50] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[51] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270 – 299, 1984.

[52] V. Goyal, A. Jain, O. Pandey, and A. Sahai. Bounded Ciphertext Policy Attribute Based Encryption. In *ICALP*, 2008.

[53] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. Cryptology ePrint Archive, Report 2019/011, 2019. `https://eprint.iacr.org/2019/011`.

[54] T. Gupta, R. P. Singh, A. Phanishayee, J. Jung, and R. Mahajan. Bolt: Data Management for Connected Homes. In *USENIX NSDI*, 2014.

[55] R. Hat. Ansible. Online: `https://www.ansible.com/`.

[56] F. Hueske and V. Kalavri. *Stream Processing with Apache Flink: Fundamentals, Implementation, and Operation of Streaming Applications*. O'Reilly Media, 2019.

[57] H. Jafarpour and R. Desai. Ksql: Streaming sql engine for apache kafka. In *EDBT*, pages 524–533, 2019.

[58] M. Jawurek and F. Kerschbaum. Fault-tolerant privacy-preserving statistics. In *Privacy Enhancing Technologies*, 2012.

[59] G. Kellaris, S. Papadopoulos, X. Xiao, and D. Papadias. Differentially private event sequences over infinite streams. *Proc. VLDB Endow.*, 7(12):1155–1166, Aug. 2014.

[60] S. Kotz, T. Kozubowski, and K. Podgorski. *The Laplace Distribution and Generalizations*. 01 2001.

[61] S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, and D. E. Culler. JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT. In *USENIX Security*, 2019.

[62] V. B. Kumar, R. Iyengar, N. Nisal, Y. Feng, H. Habib, P. Story, S. Cherivirala, M. Hagan, L. F. Cranor, S. Wilson, et al. Finding a choice in a haystack: Automatic extraction of opt-out statements from privacy policy text. In *The Web Conference (the Web Conf)*, 2020.

[63] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *Proceedings of the 11th International Conference on Privacy Enhancing Technologies*, PETS'11, page 175–191, Berlin, Heidelberg, 2011. Springer-Verlag.

[64] M. Lécuyer, R. Spahn, K. Vodrahalli, R. Geambasu, and D. Hsu. Privacy accounting and quality control in the sage differentially private ml platform. *Proceedings of the 27th ACM Symposium on Operating Systems Principles - SOSP '19*, 2019.

[65] D. S. Lemons. An introduction to stochastic processes in physics. *American Journal of Physics*, 71(2):191–191, 2003.

[66] K. Litman-Navarro. We read 150 privacy policies. they were an incomprehensible disaster.

[67] A. Marzoev, L. T. Araújo, M. Schwarzkopf, S. Yagati, E. Kohler, R. Morris, M. F. Kaashoek, and S. Madden. Towards multiverse databases. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '19, pages 88–95, New York, NY, USA, 2019. ACM.

[68] A. M. McDonald and L. F. Cranor. The cost of reading privacy policies. 2009.

[69] F. McSherry. Timely dataflow. Online: `https://timelydataflow.github.io/timely-dataflow/`.

[70] L. Melis, G. Danezis, and E. D. Cristofaro. Efficient private statistics with succinct sketches, 2015.

[71] X. Meng, S. Kamara, K. Nissim, and G. Kollios. Grecs: Graph encryption for approximate shortest distance queries. In *ACM CCS*, 2015.

[72] Netflix. How can i control how much data netflix uses? Online: `https://help.netflix.com/en/node/87`.

[73] L. H. 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.

[74] L. of the Bouncy Castle Inc. Bouncy castle crypto apis. Online: `https://www.bouncycastle.org/`.

[75] P. Ohm. Broken promises of privacy: Responding to the surprising failure of anonymization'57 ucla law review 1701 (2010) 1701-1711, 2010.

[76] B. I. Oren Ben-Kiki, Clark Evans. Yaml ain't markup language. Online: `https://yaml.org/`.

[77] P. Paillier. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, pages 223–238, 1999.

[78] T. Palino. Running kafka at scale. Online: `https://engineering.linkedin.com/kafka/running-kafka-scale`.

[79] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan. Big Data Analytics over Encrypted Datasets with Seabed. In *USENIX OSDI*, 2016.

[80] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind Seer: A Scalable Private DBMSs. In *IEEE Security and Privacy*, 2014.

[81] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *ACM SOSP*, 2011.

[82] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[83] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, Feb. 1978.

[84] M. A. Rothstein. Is deidentification sufficient to protect health privacy in research? *The American Journal of Bioethics*, 10(9):3–11, 2010.

[85] A. Sahai and B. Waters. Fuzzy Identity-Based Encryption. In *EUROCRYPT*, 2005.

[86] M. J. Schwartz. GDPR: Europe Counts 65,000 Data Breach Notifications So Far. Online: `https://www.bankinfosecurity.com/gdpr-europe-counts-65000-data-breach-notifications-so-far-a-12489`, May 2019.

[87] C. Shen, R. P. Singh, A. Phanishayee, A. Kansal, and R. Mahajan. Beam: Ending Monolithic Applications for Connected Devices. In *USENIX Annual Technical Conference (ATC)*, 2016.

[88] E. Shi, R. Chow, T.-H. H. Chan, D. Song, and E. Rieffel. Privacy-preserving Aggregation of Time-series Data. In *NDSS*, 2011.

[89] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing Analytical Queries Over Encrypted Data. In *VLDB*, pages 289–300, 2013.

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

[91] Ververica. What is stream processing? Online: `https://www.ververica.com/what-is-stream-processing`.

[92] F. Wang, Y. Joung, and J. Mickens. Cobweb: Practical remote attestation using contextual graphs. pages 1–7, 10 2017.

[93] F. Wang, R. Ko, and J. Mickens. Riverbed: Enforcing user-defined privacy constraints in distributed web services. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 615–630, Boston, MA, Feb. 2019. USENIX Association.

[94] F. Wang, J. Mickens, N. Zeldovich, and V. Vaikuntanathan. Sieve: Cryptographically Enforced Access Control for User Data in Untrusted Clouds. In *USENIX NSDI*, 2016.

[95] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. pages 1 – 9, 04 2010.

A Appendix

A.1 Random Graph - Connectivity

Theorem A.1 *Let $G(n,p)$ be a random graph generated via the Erdős–Rényi model, then it holds that:*

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

Proof 1 *The proof is from [90], but the appendix lists it for completeness. The first step is to count the number of j-subsets S of vertices, such that there are no edges between S and $\bar{S}$. There are $\binom{n}{j}$ such subsets, and each of the j vertices has $n-j$ possible neighbors outside of S. Let $\binom{V}{j}$ denote the set of all j-subsets of the vertices, then*

$$\begin{aligned}
\mathbb{E}\left[\left|\left\{S \subseteq \binom{V}{j} : \text{ no edges between } S \text{ and } \bar{S}\right\}\right|\right] &= \binom{n}{j}(1-p)^{j(n-j)} \\
&\leq \left(\frac{e \cdot n}{j}\right)^j (1-p)^{j(n-j)} \\
&= \left(\frac{e \cdot n}{j}(1-p)^{n-j}\right)^j
\end{aligned}$$

where the inequality is a standard bound on binomial coefficients. Summing over all j up to $n/2$, the expected number of subsets S of V of size at most $n/2$ such that there are no edges between S and $\bar{S}$ is

$$\mathbb{E}\left[\left|\left\{S \subseteq \bigcup_{j=1}^{n/2} \binom{V}{j} : \text{ no edges between } S \text{ and } \bar{S}\right\}\right|\right] \leq \sum_{j=1}^{n/2} \left(\frac{e \cdot n}{j}(1-p)^{n-j}\right)^j$$

If the graph is disconnected, then there is at least one subset S of the above type since there must be a subset that is disconnected from the rest of the graph, and either it or its complement has size at most $n/2$. Thus by Markov's inequality, the probability that the graph $G(n,p)$ is disconnected (i.e. the graph has ≥ 1 of these subsets) is bounded by:

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

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Enforcement of Privacy Policies via Encryption for Distributed Unbounded Data

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Küchler	Nicolas

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
13.05.2020	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.