Diss. ETH No. 25075

Retaining Data Ownership in the Internet of Things

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

Hossein Shafagh
M.Sc. in Computer Science, RWTH Aachen
born on 21 August 1985
citizen of Germany

accepted on the recommendation of
Prof. Dr. Friedemann Mattern, examiner
Prof. Dr. Srdjan Capkun, co-examiner
Prof. Dr. Philip Levis, co-examiner

2018

Abstract

As Internet of Things (IoT) systems further emerge, we face unprecedented security and privacy challenges, especially with regards to the collected data. This data typically consists of sensor readings, tagged with metadata. For scalability, ubiquitous access, and sharing possibilities, the data is most often stored in the cloud. Securing date while in transit and in particular when being stored in the cloud is of utmost importance, as the data can be used to infer privacy-sensitive information. Moreover, transparent and secure data sharing (e.g., sharing with friends or domain experts) is considered a key requirement for the practicality and success of typical IoT systems.

In today's cloud-centric designs, users have no choice but to trust centralized parties. The increased number of security and privacy incidents, such as system compromises or unauthorized trade with users data, show that this trust is not always justified. Despite varying levels of privacy-awareness among users of different age and geopolitical groups, and even societal shifts towards privacy pragmatism and indifference, the security and privacy threats do usually have far-reaching implications, demanding adequate mechanisms and measures to address them.

In this dissertation, we investigate building secure IoT systems that protect data confidentiality and retain data ownership. We build secure systems that allow reducing the trust end-users are required to put into third parties within the IoT ecosystem, specifically towards the cloud storage and service providers. More importantly, we take a new approach on empowering the user with ownership and fine-grained access control for IoT data without sacrificing performance or security. In particular, we present three approaches to enabling a secure IoT ecosystem:

(i) Talos: Talos is a system that stores IoT data securely in a cloud database while still allowing query processing over the encrypted data. Talos protects data even if the server is compromised. We enable this by encrypting IoT data with a set of cryptographic schemes such as order-preserving and partially homomorphic encryption. We tailor Talos to accommodate for the resource asymmetry of the IoT, particularly towards constrained IoT devices. We assess the feasibility of Talos on low-power devices with and without cryptographic hardware accelerators and quantify its overhead concerning energy consumption, computation time, and latency. With a thorough evaluation of our prototype implementation,

we show that Talos is a practical system that can provide a high level of security with reasonable overhead.

(ii) Pilatus: Storage of data on cloud services naturally facilitates data sharing with third-party services and other users, but bears privacy risks. We present Pilatus, a data protection platform that extends Talos where the cloud stores only encrypted data, yet is still able to process a defined set of database queries (e.g., range or sum). Pilatus features a novel encrypted data sharing scheme based on re-encryption, with revocation capabilities and in situ key-update. Our solution includes a suite of novel techniques that enable efficient partially homomorphic encryption, decryption, and sharing. We present performance optimizations that render these cryptographic tools practical for mobile platforms. We implement a prototype of Pilatus and evaluate it thoroughly. Our optimizations achieve a performance gain within one order of magnitude compared to state-of-the-art realizations.

(iii) Droplet: Droplet is a secure data management system that we designed from the ground up to accommodate for the distributed nature of the IoT and revive the IoT from the current vertical design paradigm. The consequent myriad of isolated data silos of classical vertical architectures is hard to manage and prevent heterogeneous applications from interacting with our IoT data. To address this challenge, we leverage the blockchain technology to bootstrap trust for a distributed, secure, and resilient access control and data management scheme. Droplet handles time series data, enables reliable sharing among heterogeneous applications without intermediate trust entities, and features a cryptographically-protected fine-grained and scalable access control mechanism to data streams. We leverage a hash-chain-based key management mechanism to enable interval sharing and compact key distribution. The built-in cryptocurrency feature of blockchains allows the integration of economic incentives into our system. These properties enable a variety of applications that are presently not easily realizable using existing systems.

The systems proposed and discussed in this dissertation demonstrate that end-to-end encryption with secure sharing can be achieved in IoT ecosystems with a modest overhead, while maintaining a consistent user-experience.

Zusammenfassung

Die zunehmende Verbreitung des Internet der Dinge (Internet of Things, IoT) konfrontiert uns mit neuen Sicherheits- und Schutzproblemen, insbesondere in Bezug auf die anfallenden Daten. Diese Daten bestehen in der Regel aus Sensormesswerten, die mit Metadaten versehen sind. Um darauf einen ubiquitären Zugriff sowie eine gemeinsame Nutzung zu ermöglichen, aber auch um die Skalierbarkeit des zugrundeliegenden Systems zu gewährleisten, werden die bei IoT-Systemen anfallenden Daten typischerweise in der Cloud gespeichert. Dabei ist der Schutz der Daten während der Übertragung und insbesondere bei der Speicherung in der Cloud von grösster Bedeutung, da aus diesen Daten oft sensitive Informationen hinsichtlich schützenswerter Bereiche (wie z.B. der Privatsphäre von Personen) abgeleitet werden können. Darüber hinaus wird auch eine transparente und sichere gemeinsame Nutzung von Daten (z.B. das Teilen mit Verwandten und Bekannten oder mit Fachexperten) als eine wesentliche Voraussetzung für die Praktikabilität und den Erfolg typischer IoT-Systeme angesehen.

Bei den gegenwärtig vorherrschenden cloudzentrierten Architekturen bleibt den Nutzern keine andere Wahl, als den zentralen Elementen und Entitäten zu vertrauen. Andererseits zeigt die zunehmende Zahl von Sicherheits- und Datenschutzvorfällen (wie kompromittierte Systeme oder unerlaubter Handel mit Nutzerdaten), dass dieses Vertrauen nicht immer gerechtfertigt ist. Auch wenn bei den Nutzern je nach Alter und soziokulturellem Hintergrund das Bewusstsein für Datenschutz unterschiedlich stark ausgeprägt ist und aufgrund gesellschaftlicher Veränderungen bis hin zu Datenschutzpragmatismus und Gleichgültigkeit reichen kann, haben Bedrohung der Sicherheit und Gefährdung des Datenschutzes in der Regel weitreichende Folgen und erfordern dementsprechend angemessene Mechanismen und Massnahmen, um diesen Bedrohungen zu begegnen.

Die vorliegende Dissertation hat den Bau sicherer IoT-Systeme zum Thema, welche von vornherein die Vertraulichkeit der Daten gewährleisten und den Nutzern eine weitreichende Kontrolle über diese geben. Konkret konzipieren wir sichere Systeme, bei denen das erforderliche Vertrauen substantiell reduziert werden kann, welches Nutzer den beteiligten Drittparteien, speziell den Cloud- und Service-Providern, entgegenbringen müssen. Dazu verfolgen wir einen neuen

Ansatz, der es Nutzern ermöglicht, die Hoheit über ihre Daten zu behalten und feingranulare Datenzugriffsrechte zu erteilen und zu verwalten, ohne dass dadurch die Leistungsfähigkeit oder die Sicherheit des Systems beeinträchtigt werden. Konkret stellen wir drei Ansätze für eine sichere IoT-Umgebung vor:

(i) Talos: Hierbei handelt es sich um ein System, das IoT-Daten sicher in einer Cloud-Datenbank speichert und dennoch die Abfrageverarbeitung auf den verschlüsselten Daten ermöglicht. Talos schützt damit die Daten, auch wenn der Server selbst kompromittiert ist. Wir ermöglichen dies, indem wir die IoT-Daten mittels einer Reihe kryptographischer Verfahren verschlüsseln, darunter ordnungserhaltende und partiell-homomorphe Verschlüsselungen. Talos berücksichtigt die Ressourcenasymmetrie des IoT, die sich vor allem in leistungsmässig stark reduzierten IoT-Geräten manifestiert. Wir untersuchen die Machbarkeit von Talos auf Low-Power-Geräten mit und ohne kryptographische Hardwarebeschleuniger und quantifizieren den Overhead bezüglich Energiebedarf, Berechnungszeit und Latenz. Mitttels einer umfassenden Evaluierung unserer Prototypimplementierung weisen wir nach, dass Talos ein praxistaugliches System ist, welches ein hohes Mass an Sicherheit bei angemessenem Overhead bieten kann.

(ii) Pilatus: Die Speicherung von Daten in Cloud-Diensten erleichtert zwar den Datenaustausch mit Diensten von Drittanbietern und anderen Benutzern, birgt jedoch Risiken bezüglich der Privatsphäre und des Datenschutzes. Pilatus stellt eine Erweiterung von Talos dar und realisiert eine Datenschutzplattform, bei der in der Cloud die Daten nur in verschlüsselter Form vorliegen und sie dort dennoch mit einer eingeschränkten Menge von Datenbankfunktionen (z.B. Abfrage bezüglich eines Zeitintervalls oder Abfrage zur Ermittlung eines Summenwertes) verarbeitet werden können. Pilatus realisiert ein neuartiges Konzept zum Teilen verschlüsselter Daten, das auf Wiederverschlüsselung (re-encryption) beruht und Sperrfunktionen sowie In-Situ-Schlüsselaktualisierung bietet. Unsere Lösung umfasst eine Reihe neuartiger Techniken, die eine effiziente partiell-homomorphe Verschlüsselung und Entschlüsselung sowie ein effizientes Teilen von Daten ermöglichen. Um die kryptographischen Prinzipien für mobile Plattformen nutzbar zu machen, sind Leistungsoptimierungen nötig; wir konnten gegenüber anderen gegenwärtigen Realisierungen bei unserem umfassend evaluierten Prototypen einen Leistungsgewinn von einer Grössenordnung erzielen.

(iii) Droplet: Droplet ist ein sicheres Datenverwaltungssystem, das wir von Grund auf so entworfen haben, dass es die verteilte Natur des IoT berücksichtigt. Die unzähligen isolierten Datensilos,

welche die klassischen IoT-Architekturen nach sich ziehen, sind nicht nur schwer zu verwalten, sondern behindern in starkem Masse die Interaktion heterogener Anwendungen mit den IoT-Daten. Um diese Herausforderung zu meistern, nutzen wir die Blockchain-Technologie. Somit entwerfen wir ein allgemein vertrauenswürdiges, kryptographisch abgesichertes, feinkörniges und skalierbares Zugriffskontroll- und Datenmanagementschema. Droplet verarbeitet Zeitreihendaten und ermöglicht heterogenen Anwendungen eine zuverlässige gemeinsame Datennutzung ohne zwischengeschaltete Vertrauensentitäten. Um die gemeinsame Datennutzung in Zeitintervallen und eine kompakte Schlüsselverteilung zu ermöglichen, wird ein auf Hash-Ketten beruhender Schlüsselverwaltungsmechanismus genutzt. Die bei Blockchains implizit vorhandene Kryptowährungskomponente erlaubt die Integration von wirtschaftlichen Anreizen in unser System. Diese Eigenschaften ermöglichen eine Vielzahl von Anwendungen, die mit existierenden Systemen nicht einfach realisierbar sind.

Zusammengefasst zeigen die in der vorliegenden Dissertation diskutierten Methoden und Systemkomponenten in prototypischer Weise, dass in IoT-Umgebungen eine sichere Ende-zu-Ende-Verschlüsselung sowie ein sicheres Teilen mit einem geringen Overhead und ohne wesentliche Zusatzbelastung des Nutzers erreicht werden kann.

Dedicated to my parents.

Contents

List of Figures

List of Tables

1

Introduction

Privacy remains a constant concern while migrating most aspects of our lives to the digital space. Today developers predominantly utilize remote servers and cloud services to provide storage and ubiquitous access to our data. Though the cloud-centric design naturally facilitates data sharing and access to vast computational resources, it leaves users, however, with no choice but to trust centralized parties with their private data. The increased number of security and privacy incidents, such as system compromises or unauthorized trade with users data, show that this trust is often unjustified. The security and privacy implications exacerbate with the emerging new technologies, such as the Internet of Things (IoT), which collect an unprecedented amount of sensitive data. The IoT introduces novel sources of nonintrusive data acquisition, ultimately laying the groundwork for novel data analytics, richer and potentially automated decision making process, and artificial intelligence. Such networked ambient sensing devices are for instance installed in buildings, at homes, and on human bodies. Collected data in the IoT space is inherently privacy-sensitive, since it embodies multi-dimensional representations of our immediate environment. In the face of the rapid pace of technological innovations in the space of IoT, it is essential to integrate data security and privacy aspects throughout the whole design and development process. Past and current security and privacy-related threats proved to have far-reaching implications and at times deterring broad adoption of new technology. We have witnessed such implications with smart meter technologies, hindering their widespread adoption [65]. Moreover, studies on perceptions of privacy in IoT [99] show that consumers are highly concerned about the privacy of their collected data,

e.g., companies selling them to third parties without their consent. Consequently, lack of trust and confidence are major challenges these technologies have yet to overcome. To address these concerns, we envision a new design paradigm that allows reducing the trust end-users are required to put into third parties, specifically towards the cloud storage and service providers, without, however, compromising on user experience nor comprehensive services. More importantly, we take a new attempt at empowering users with data ownership and fine-grained access control of their data, leveraging cryptographic techniques.

The fundamental goal of this thesis is to develop algorithmic and system foundations to build practical, secure systems that compute on encrypted data and retain control of data to users. Moreover, we delve into the fundamental questions of how to design computing systems to be more secure, private, and egalitarian.

Scope of thesis. This thesis focuses on data security and privacy in the IoT space. These systems often comprise IoT devices which collect data and store it on third party storage providers. Our goal is to conceal data from unauthorized parties, such that service providers do not learn anything about users data, while still being able to provide their services. This level of protection would as well prevent unauthorized data access due to unintended data leakage, as a consequence of internal or external attacks, e.g., a compromised server. We intend to empower users to be fully in control of who can access their data and to what extent. We require a secure sharing mechanism to be in place, allowing the data owner to transparently and selectively share their data. Data owners should be able to securely and privately interact with their data, while being able to exploit external services, e.g., specialized analytical services. For instance, consider health tracking wearables that enable novel applications, such as personalized medical care or improved personal vitality, with the help of the unique longitudinal data over human body's vital signs. The data owner should be able to selectively and securely share their health data with partner/family/friends or medical practitioners without necessarily revealing any personal data to the infrastructure providers. Moreover, the data owner can potentially be interested beyond the above bilateral data sharing scenarios, and be willing to contribute to larger datasets (e.g., to improve the accuracy of machine learning models, statistical databases, or exploratory research purposes), as long as the privacy of their data is guaranteed.

Empowering users with data ownership and control over their data, requires designing systems and devising algorithms that are secure and privacy-aware compared to the current systems. More generally, we define data ownership as having the right and control over personal data,

wherein the owner can define/restrict access to their data, restrict the scope of data utility (e.g., sharing aggregated/transformed/homomorphically-encrypted data, instead of raw data), delegate these privileges, or give up ownership entirely. A true realization of this definition requires work on two fronts: *(i)* privacy-preserving computation and *(ii)* decentralized access control without centralized trust entities, to ensure private access to externally hosted data, with strong confidentiality guarantees. In this dissertation, we propose novel system designs that contribute to the current state on both of these fronts. We propose three individual systems, where the first two (i.e., Talos and Pilatus) focus on computing on and sharing of encrypted data and the third one (i.e., Droplet) enables a trustless decentralized access control with cryptographically-enforced data access.

Research Questions. Massive leakage of confidential data plagues the computing systems of today, with ever-increasing data breaches. Additionally, large high dimensional data collections face privacy and security risks due to lack of transparency with regards to adequate data protection and consensual data sharing mechanisms. One effective approach to ensure confidentiality is to store only encrypted data on remote/third-party servers without revealing any encryption keys to servers. While ensuring ownership (i.e., only authorized parties gain access to decryption keys) and providing the highest level of protection (i.e., an attacker can only see encrypted data and is prevented from seeing data content), a strawman realization of this approach falls short in utilizing the available resources on the cloud and lacks the required expressiveness for a fine-grained data sharing. Hence, throughout this thesis, we seek to answer the following two main research questions.

How to benefit from cloud computing (i.e., storage and query processing) without compromising data control and security? The strawman solution of storing encrypted data with traditional symmetric encryption schemes, such as AES, would offer protection but renders the data unsearchable and leaves out the challenge of secure sharing. Alternatively, fully homomorphic encryption schemes enable arbitrary computations on encrypted data but are presently impractical [184]. A large body of research efforts has shown the potential of practical encrypted query processing systems [184, 185, 231], which utilize a combination of different encryption schemes to process queries on encrypted data. In this thesis, we focus on employing Partially Homomorphic Encryption (PHE) schemes and in particular additive homomorphic schemes. These are practical solutions that enable an important set of queries [231], such as encrypted data aggregation – a common operation in IoT applications when history data needs summing or averaging. Also note that with

limited involvement of the client side, more complex computations (e.g., linear regression) can also be achieved [231].

How to bring sharing with cryptographic guarantees to the IoT ecosystem? The current PHE approaches are either targeted at single-key encrypted data [184, 231, 213] (no support for sharing) or consider only text-based data [185, 105] (of limited use in an IoT context). Existing protocols for sharing, such as OAuth [153], fall short in providing strong assurances about the policy enforcement. Crypto-based sharing approaches [234], on the other hand, support no query processing over encrypted data.

In short, existing solutions either support encrypted query processing or secure sharing but not both. Moreover, due to their heavy computational overhead, PHE schemes have been considered unsuitable for low-power mobile and IoT devices. In this work, we show the feasibility of PHE-based schemes on low-power devices (Chapter 2) and tackle the challenges of cryptographically-protected and efficient sharing of *(i)* PHE data (Chapter 3), and *(ii)* time-series data (Chapter 4). Before discussing the contributions of this thesis, we give an overview of the state-of-the-art research efforts in the area of data privacy and data security, and discuss our perspective on IoT data.

1.1 Data Privacy and Security Overview

To protect the privacy of data, we can distinguish between the following five categories of approaches, which are not mutually exclusive. We now briefly review each of them and refer to the corresponding chapters for a more detailed discussion of the relevant approaches to our work.

(i) **Authorization**. With respect to externally hosted data, authorization defines whether a request to access a resource, in our case data, must be granted or denied. The most primitive and common form of authorization can be realized via rudimentary password-based schemes, which enable a coarse-grained level of access. Fine-grained access control is more complex and challenging, specifically in cases where resources and the involved players are not in a closed system, where all authorized principals are known in advance. We distinguish between two types of authorization with regards to data: policy-based and crypto-based.

Distributed Access Control. Authorizing a request involves identifying the principal (i.e., authentication) and verifying the validity of the request based on the resource owner's access control policy (i.e., access control). Authorization in distributed systems is complex, as resources are spread across the network, potentially under multiple administrative domains, where principals most likely are not known beforehand, rending

authentication challenging. Such open networks require accounting for all types of adversaries and malicious entities.

Current distributed access control schemes, such as OAuth2 [153], and Macaroons [31], rely on tokens issued by a trusted intermediary serving as an identity provider. Users present these tokens (i.e., credentials) to a gatekeeper of a resource, e.g., data. The gatekeeper forwards the token to the issuer who in return confirms the validity of the token or rejects it, if invalid. These schemes are dependent on trusted authorities for token issuance and validation. Today only a handful of centralized authorities control this space, e.g., Google, Facebook, and Amazon, who as well learn about all the services a user calls on.

Though signature-based schemes (e.g., public-key based certificates [33, 66]) do not suffer from these limitations; they require a centralized, hierarchical network of certification authorities (CA) to issue certificates, which come with their weaknesses [158]. Alternative decentralized public-key based approaches, e.g., SPKI/SDSI [66] and follow-up schemes [67], eliminate the need for complex X.509 public key infrastructure and CAs. However, these schemes are either based on the idea of local names and suitable for deployments under a single administrative domain (e.g., smart home) or build upon an organically growing trust model (i.e., Web of Trust [243]).

Crypto-enforced Data Access. Client-side encryption provides a stronger level of protection, as the service providers and resource gatekeepers only see encrypted data. However, it adds constraints on sharing. Data in this setting can only be shared at a coarse-grained level. Various cryptographic schemes [36, 13] have been introduced to overcome this limitation, among which attribute-based encryption (ABE) [97, 234, 197, 96] offers the best expressiveness in this space. At a higher level, in ABE data is encrypted towards a policy (i.e., associated with a set of attributes), and only those with the secret keys satisfying the policy can decrypt the data. However, ABE comes with the following limitations: *(i)* keys are not identity-based (i.e., not based on public keys) requiring a key distribution mechanism, *(ii)* access cannot be verified without a decryption test, leaving the storage provider susceptible to download attacks, as anyone can claim to have the right set of keys, *(iii)* enc/decryption overhead, which is expensive due to the underlying pairing-based cryptography, grows linearly with the number of attributes, limiting the granularity of access control due to prohibitive computational burdens [80, 3].

(ii) **Perturbation-based techniques** include various tools of generalization, suppression, swapping, and the addition of noise to data [168]. Differential privacy [62, 63] is a prominent technique in this category enabling interaction with a statistical database without being able to de-

identify individual records. Conceptually, differential privacy introduces controlled randomness (i.e., noise) into the data, while maintaining a defined quality of queries. The degree of added noise is set by the privacy budget, which defines the total allowed leakage and determines the total number of queries. A high privacy budget means a higher probability of leaking data, whereas a too low budget might render the query results useless. Initial schemes required a trusted database operator who enforces the privacy budget. More recent schemes [68] overcome this limitation via the randomized response technique where no raw data is collected.

(iii) **Secure multi-party computation**. In traditional Secure Multi-Party Computation (MPC) [238], private functions are computed among a set of users without a trusted party. Hereby individual values from participating users are kept confidential, while the outcome can be public. Though MPC requires high interaction between users, which would drain the limited resources of mobile platforms, there have been efforts to improve their performance [53]. With the rise of cloud computing, server-aided/outsourced MPC approaches have emerged. However, these schemes are either only of theoretical interest [154] or require at least two non-colluding servers, where for instance one server has only access to encrypted data, and the other server has access to the encryption keys [233, 182, 175].

(iv) **Encrypted data processing.** Recent advancements of fully homomorphic encryption [83] have resulted into implementable schemes [84, 45, 204, 54], which are however presently yet too slow for real-world applications. Searchable encryption schemes support only a limited set of operations, but can be efficiently used in specialized domains. Initial encrypted search schemes [223] target encrypted text files. The basic idea here is based on deterministically encrypting the meta information of files, and hence being able to search for them. More capable search schemes [184, 213, 219, 38, 199, 204, 231], targeted for structured databases, employ additional techniques such as partially homomorphic and order-preserving encryptions. It is important to note that access patterns to encrypted data still leak sensitive information about the plaintext data. This shortcoming can be addressed with Oblivious RAM approaches [195, 227]. Moreover, practical encrypted query processing systems come with a functionality-security tradeoff, which requires careful consideration while deciding on the required level of data protection. For instance, order-preserving encryption, per definition reveals the ordering information among the ciphertexts [171]. While for specific data types, the traded leakage for performance is acceptable for others it might pose the risk of complete disclosure. This is especially the case for low-entropy values.

(v) **Anonymity.** The rationale behind anonymity is to make users not identifiable based on their data in a dataset. One standard technique to realize anonymity is to remove any personally identifiable information from the data. However, the privacy is protected only as long as the adversary is not able to re-link the identity of an individual with their data record in the dataset. Anonymity based on de-identification techniques is widespread and widely deployed, as it is easily and cheaply realizable. However, despite being practical, de-identification should not be considered as a universal privacy-protection mechanism, as it leaves open the types of supported computations on the dataset. Narayanan et al. [168, 170] argue that a meaningful definition of privacy can only be reached as a property of specific computations, as achieved by differential privacy, and not as a property of the dataset. Hence, the increased reliance on syntactically defined anonymity will have serious implications for privacy in data sharing, e.g., as recently demonstrated by the Strave [23] heat-maps of aggregated running routes disclosing locations of sensitive governmental facilities.

1.2 Internet of Things

Though our schemes are conceptually generic, they are tailored towards the IoT with regards to data type and resource asymmetry. The IoT [159] consists of varies types of interconnected devices that collect data from our surroundings. IoT deployments today are mostly in silos within organizations that control the various involved software and computing stacks, including the smart devices, processing of the collected data on the cloud, and interactive web or smartphone apps to access and monitor the collected data. IoT devices communicate directly (or via a gateway) to the cloud and even interact with each other through Internet services. The vision of a vibrant ecosystem in which data can be shared towards a wide range of heterogeneous applications and services is yet to be realized.

Typical applications fall into two main categories; *(i)* ambient data collection, including microcontrollers that can be installed within appliances, at homes, or on humans via wearables, and *(ii)* real-time applications, including autonomous systems, e.g., various types of robots, and connected cars. All these applications have serious privacy implications due to the sensitivity of data they collect. We distinguish between two main classes of constrained IoT devices: *(i)* IP-enabled, e.g., microcontroller platforms, mini PCs, or smartphones, and *(ii)* short-range wireless devices such as wearables that rely on a personal gateway (e.g., smartphone) for IP communication. For many application scenarios,

the role of the cloud appears to be inevitable, as it allows resource-limited IoT devices to upload their collected data, and enables ubiquitous access to the data. The asymmetry of available resources in terms of bandwidth, computation, and energy within different entities of the IoT necessitates a careful design of security solutions for the IoT ecosystem. IoT devices are typically equipped with limited resources regarding computation, memory, and bandwidth. They can embed low-power hardware crypto accelerators for efficient computation of cryptographic operations, enabling a new class of secure applications, as envisioned by this dissertation.

In this thesis, we focus on bilateral data sharing settings, which in contrast to multilateral sharing (e.g., crowd-sourcing, model training) cannot benefit from group privacy, e.g., as enabled by differential privacy [68]. This is specifically important, as today IoT services collect and control growing amounts of sensitive personal data (e.g., health, home, car) with little or no transparency. Privacy and security concerns have been steadily rising, notably for the IoT, due to the surge in data breaches [134, 18], misconduct of service providers [59, 29], and surveillance [186, 23].

Throughout this work, we consider the running examples of applications collecting sensitive data that require processing and sharing, e.g., fitness and health trackers. These applications store personal data in the cloud that can reveal privacy-sensitive information, e.g., illness, lifestyle, or location. Though insightful, logging such private information raises serious privacy concerns, and requires adequate protections in place.

At the same time, secure and transparent sharing plays an essential role in bringing such applications to their full potential. When users are willing to share data with experts (e.g., medical practitioners), analytical services, or just casually with friends, they must be in full control over who can access and what can be accessed. Moreover, it is essential to support query processing capabilities directly in the cloud on encrypted data, since downloading the entire data volume for on-device processing becomes impractical as the data grows.

1.3 Dissertation Contributions

The system designs presented in this dissertation have been developed and made available as reference implementations and evaluated thoroughly. Our evaluations reveal that while secure systems come with additional overheads due to more expensive computations,

the user-experience can remain unaltered. The detailed results of each system design are discussed in the individual chapters. The specific contributions of this dissertation are highlighted below.

1.3.1 Talos: Encrypted Data Processing for the IoT

Several encrypted query processing approaches [184, 185, 223, 120, 38] have been introduced in the past years, which utilize cryptographic techniques (e.g., partially homomorphic encryption) that allow computations to be carried out on encrypted data. One major barrier to employing such cryptographic primitives on IoT devices is their resource constraints. IoT devices are inherently limited with regards to energy, memory, CPU, and bandwidth. This challenge is exacerbated by computationally heavy asymmetric-crypto-based schemes, such as additive homomorphic encryption schemes.

To overcome these challenges, we introduce Talos, a system that stores IoT data securely in a cloud database while still allowing query processing over the encrypted data. We enable this by encrypting IoT data with a set of cryptographic schemes such as order-preserving and partially homomorphic encryption. We tailor Talos to accommodate for the resource asymmetry of the IoT, particularly towards constrained IoT devices. Talos leverages a batching technique to utilize the large ciphertextsize in the Paillier cryptosystem to boost the performance of crypto operations (e.g., encryption, decryption, homomorphic addition) due to the concept of Single Instruction, Multiple Data (SIMD), where for instance one single homomorphic addition is required for the entire batch. Additionally, Talos proposes EC-ElGamal as a viable and efficient alternative to the Paillier cryptosystem. EC-ElGamal has the additive homomorphic property and scores with more efficient key generation and encryption. However, decryption is a bottleneck, as it requires solving a discrete log problem. We overcome this challenge, with a multi-threaded baby-step-giant-step algorithm for 16-bit and 32-bit integer values.

We assess the feasibility of Talos on low-power devices with and without cryptographic hardware accelerators and quantify its overhead regarding energy, computation, and latency. With a thorough evaluation of our prototype implementation, we show that Talos is a practical system that can provide a high level of security with reasonable overhead.

Chapter 2 describes the detailed architecture of Talos and discusses results of our evaluation.

1.3.2 Pilatus: Partially Homomorphic Encrypted Sharing

Storage of data on cloud services naturally facilitates data sharing with third-party services and other users, but bears privacy risks. We present Pilatus, a data protection platform that builds on Talos to support secure sharing of encrypted data stored on cloud services, yet being able to process a defined set of database queries (e.g., range or sum). Pilatus features a novel encrypted data sharing scheme based on re-encryption, with revocation capabilities and in situ key-update. Our key revocation mechanism allows users to terminate their data sharing at any time. The in situ key-update at the cloud, protects as well old data with the owner's new key, without trusting the cloud with any private keys. Our solution includes a suite of novel techniques that enable efficient partially homomorphic encryption, decryption, and sharing. We present performance optimizations that render these cryptographic tools practical for mobile platforms. More specifically, we address the performance optimization with the Chinese Remainder Theorem, which enables smaller, faster, and parallel computations. Our optimizations achieve a performance gain within one order of magnitude compared to state-of-the-art realizations.

Chapter 3 expands on Pilatus architecture and components, describes its reference implementation, and details results of our evaluation.

1.3.3 Droplet: Decentralized Authorization for Data Streams

The immense possibilities of data acquisitions through the IoT are contributing to an increased number of new data sources and many novel applications yet to come. The emerging IoT developments, however, leave data owners with little control over their data and require their blind trust in protecting their data. To address this challenge, we present *Droplet*, a decentralized data access control service, which operates without intermediate trust entities. Droplet enables data owners to securely and selectively share their data, while guaranteeing data confidentiality against unauthorized parties. Droplet leverages the blockchain technology to bootstrap trust, for our decentralized, secure, and resilient access control management. Droplet handles time-series data, and features a cryptographically-enforced fine-grained and scalable access control for encrypted data streams. With a prototype implementation of Droplet on a public blockchain, we quantify Droplet's overhead and compare it to the state-of-the-art systems. We discuss the experimental results of three case-study applications on top of Droplet: the Fitbit activity tracker, the Ava health tracker,

and the ECOviz smart meter dashboard. When deploying Droplet with Amazon's S3 as a storage layer (a popular cloud storage service), we experience a slowdown of only 3% in request throughput. Moreover, we show the potential of Droplet as authorization service for the serverless computing domain, which requires request-level authorization.

Chapter 4 describes the detailed architecture Droplet and elaborates on the individual components of Droplet.

1.4 Roadmap

Each chapter of this dissertation is organized in a chronological and didactic order and can be read as a separate self-contained component.

We start with the description of Talos in Chapter 2. We elaborate on the design of Talos and discuss how we overcome the resource constraints of IoT devices, to enable a secure system based on advanced cryptography. We discuss the design components of Talos and the technical challenges Talos had to overcome. In Chapter 3, we present the design of Pilatus, which builds on the design of Talos, to enable secure sharing. We elaborate how Pilatus drastically optimizes the performance of EC-ElGamal decryption by leveraging the Chines Remainder Theorem. Then, we introduce the design of the Pilatus centerpiece, namely, secure sharing, based on re-encryption. We discuss the performance overhead of Pilatus on real-world applications. We then introduce Droplet, a decentralized access control system, in Chapter 4. We elaborate how Droplet ensures data ownership with a crypto-based fine-grained access control mechanism, which eliminates the need for trusted third parties. We discuss the design components of Droplet and elaborate on the technical challenges we had to overcome. We conclude this dissertation in Chapter 5 with a summary of our core contributions and a discussion of future research avenues.

1.5 Publications

The following list includes publications [206, 207, 205, 210, 211, 212, 213] that form the basis of this thesis in chronological order and of which I am the first author and the main contributor. The corresponding chapters are indicated in parentheses.

Hossein Shafagh, Lukas Burkhalter, Simon Duquennoy, Anwar Hithnawi. **Droplet: Decentralized Authorization for IoT Data Streams.** In ArXiv, arXiv:1806.02057v1 [cs.CR], June 2018. (Chapter 4)

Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, Simon Duquennoy. **Towards Blockchain-based Auditable Storage and Sharing of IoT Data.** In *Proceedings of the 9th ACM Cloud Computing Security Workshop (CCSW).* Dallas, TX, USA. pp. 45-50, November 2017. (Chapter 4)

Hossein Shafagh, Anwar Hithnawi, Lukas Burkhalter, Pascal Fischli, Simon Duquennoy. **Secure Sharing of Partially Homomorphic Encrypted IoT Data.** In *Proceedings of the 15th ACM Conference on Embedded Networked Sensor Systems (SenSys).* Delft, Netherlands. pp. 1–14, November 2017. (Chapter 3)

Hossein Shafagh, Anwar Hithnawi. **Privacy-preserving Quantified Self: Secure Sharing and Processing of Encrypted Small Data.** In *Proceedings of the ACM SIGCOMM 2017 Workshop on Mobility in the Evolving Internet Architecture (MobiArch).* Los Angeles, CA, USA. pp. 25–30, August 2017. (Chapter 3)

Hossein Shafagh, Anwar Hithnawi, Andreas Dröscher, Simon Duquennoy, Wen Hu. **Talos: Encrypted Query Processing for the Internet of Things.** In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys).* Seoul, South Korea. pp. 197–210, November 2015. (Chapter 2)

The following list includes selected publications [136, 135, 109, 204, 107, 112, 111, 118, 209, 208, 108, 110, 196] that I have co-authored during my doctoral studies and are not part of this dissertation.

Jun Young Kim, Wen Hu, Hossein Shafagh, Sanjay Jha. **SEDA: Secure Over-The-Air Code Dissemination Protocol for the Internet of Things.** In *Proceedings of IEEE Transactions on Dependable and Secure Computing (TDSC).* December 2016.

Anwar Hithnawi, Su Li, Hossein Shafagh, James Gross, Simon Duquennoy. **CrossZig: Combating Cross-Technology Interference in Low-power Wireless Networks.** In *Proceedings of the 15th ACM International Conference on Information Processing in Sensor Networks (IPSN).* Vienna, Austria. pp. 1–12, April 2016.

Hossein Shafagh. **Toward Computing Over Encrypted Data in IoT Systems.** In *XRDS: Crossroads, The ACM Magazine for Students Volume 22 Issue 2.* Winter 2015. pp. 48-52, December 2015.

Anwar Hithnawi, Vaibhav Kulkarni, Su Li, Hossein Shafagh. **Controlled Interference Generation for Wireless Coexistence Research.** In *Proceedings of the Software Radio Implementation Forum (SRIF) at ACM MobiCom.* Paris, France. pp. 19–24, September 2015.

Anwar Hithnawi, Hossein Shafagh, Simon Duquennoy. **TIIM: Technology-Independent Interference Mitigation for Low-power Wireless Networks.** In *Proceedings of the 14th ACM International Conference on Information Processing in Sensor Networks (IPSN).* Seattle, WA, USA. pp. 1–12, April 2015.

Anwar Hithnawi, Hossein Shafagh, Simon Duquennoy. **Understanding the Impact of Cross Technology Interference on IEEE 802.15.4.** In *Proceedings of the 9th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH) at ACM MobiCom.* Maui, HI, USA. pp. 49–56, September 2014.

René Hummen, Hossein Shafagh, Shahid Raza, Thiemo Voigt, Klaus Wehrle. **Delegation-based Authentication and Authorization for the IP-based Internet of Things.** In *Proceedings of the 11th IEEE International Conference on Sensing, Communication, and Networking (SECON).* Singapore. pp. 284-292, June 2014.

2

Talos

The Internet of Things, by digitizing the physical world enables novel interaction paradigms with our surroundings. This creates new threats and leads to unprecedented security and privacy concerns. To tackle these concerns, we introduce Talos, a system that stores IoT data securely in a Cloud database while still allowing query processing over the encrypted data. We enable this by encrypting IoT data with a set of cryptographic schemes such as order-preserving and partially homomorphic encryption. To achieve this on constrained IoT devices, Talos relies on optimized algorithms that accelerate order-preserving and partially homomorphic encryption by 1 to 2 orders of magnitude. We assess the feasibility of Talos on low-power devices with and without cryptographic accelerators and quantify its overhead regarding energy, computation, and latency. With a thorough evaluation of our prototype implementation, we show that Talos is a practical system that can provide a high level of security with reasonable overhead. We envision Talos as an enabler of secure IoT applications.

2.1 Introduction

With the advent of the Internet of Things (IoT), there has been a rise in the number of devices empowered with sensing, actuating, and communication capabilities. These devices are typically connected to Cloud services, but are physically integrated with our living space. Hence, they deal with sensitive and private data that could be misused to infer privacy-violating information. This consequently raises security

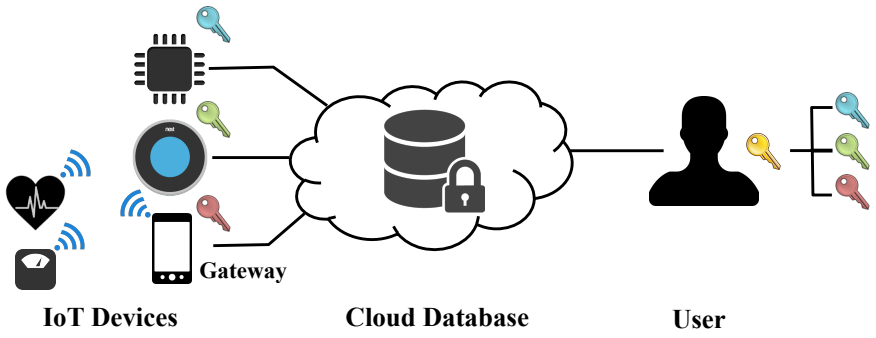


Figure 2.1: Talos enables protection of IoT data already at the origin. The Cloud database has no access to the encryption keys, but is able to process queries over encrypted data. All keys are derived from a master secret held by the user.

and privacy concerns, which need to be addressed in the IoT ecosystem.

Conventional security solutions for the IoT utilize, at best, an *end-to-end* (E2E) secure channel to store IoT data on a Cloud database. A secure E2E channel protects the communication against unauthorized entities (e.g., eavesdropping and modification attacks), but leaves the data unprotected on the Cloud. Storing data in such form leaves it vulnerable to breaches [187], caused by hackers and curious administrators [1]. Moreover, financial incentives might lure our today's trusted Cloud service providers into disclosure of sensitive information derived from our data or unauthorized sharing/selling of our data [104, 131].

Encrypted Query Processing. An intuitive approach to counter such attacks is to store data in encrypted form in the Cloud database, and have all data en-/decryption performed at the user-side. This, however, is impractical, as it prevents any server-side query processing and results in undesirable application delays. To overcome this limitation, several encrypted query processing approaches [184, 185, 223, 120, 38] have been introduced in the past years. These approaches utilize cryptographic techniques (e.g., order-preserving encryption and homomorphic encryption) that allow computations to be carried out on encrypted data.

CryptDB [184] is one of the first practical systems that integrates efficient encrypted query processing into the database management system. In CryptDB, the cloud can perform traditional database queries over encrypted data and reply with the encrypted result. To achieve this, CryptDB relies on a trusted proxy which intercepts the communication and applies en-/decryption transparent to the user. This approach does not require any modification of the database nor the client-side and adds a computation overhead of 25% [184]. CryptDB is designed with web applications in mind and is not suitable for IoT application

scenarios, mainly because: *(i)* it employs cryptographic schemes that are prohibitively expensive for constrained IoT devices and *(ii)* it relies on a trusted proxy, which has access to the encryption keys and plaintext information.

Talos: Encrypted Query Processing for the IoT. In this chapter, we present Talos[1], an IoT data protection system which securely stores encrypted IoT data on a Cloud database, while allowing for efficient database query processing over the encrypted data (see Figure 2.1). In our design, we move away from CryptDB's focus on web applications only. Instead, we design a secure E2E system that stores encrypted data from IoT devices on a Cloud database, where data protection is executed at the data source. Thus, we *dispense with the role of a trusted proxy which has access to all keying material*. This allows us to address a stronger threat model, where only the end-user has access to the secret keys.

To put the use case of Talos into context, let us consider the application scenario of a health monitoring device similar to Fitbit Tracker[2] which logs heart rate, location, and timestamps. The heart rate measurements can be used to infer sensitive information about a person, such as stress, depression, and heart-related diseases. Hence, heart rate information should be protected from untrusted parties. To still allow certain computations, e.g., average, over the protected heart rate data, Talos utilizes additive homomorphic encryption (see Section 2.3.1 for the detailed descriptions of different cryptographic terms). The location is potentially also sensitive. Thus, Talos applies deterministic encryption, allowing encrypted queries to correlate heart rate with location. Finally, the timestamps are encrypted with order-preserving encryption, to allow order-related searches.

One important barrier to employing cryptographic primitives on IoT devices is their resource constraints. IoT devices are inherently limited with regards to energy, memory, CPU, and bandwidth. These challenges are exacerbated by computationally heavy public-key-based cryptographic schemes, such as order-preserving and additive homomorphic encryption. Hence, we apply optimizations to these encryption schemes to make them suited towards IoT devices, yet without scarifying their level of security. Our work covers *(i)* an optimization of Paillier's additive homomorphic encryption scheme for integer data items, *(ii)* a solution enabling the elliptic curve

[1]In ancient Greek mythology, Talos is the protector and patron of just rulership and civil society.

[2]Fitbit Tracker Flex comprises a low-power ARM Cortex M3 Microcontroller similar to the one we based our prototype implementation on: `https://www.ifixit.com/Teardown/Fitbit+Flex+Teardown/16050`

ElGamal encryption scheme (EC-ElGamal) as an alternative additive homomorphic encryption, and *(iii)* employing an interactive order-preserving encryption scheme.

Contributions. This chapter can be summarized with the following contributions:

- Design and evaluation of Talos, a fully-implemented E2E secure system for IoT. Talos is compatible with the core CryptDB, which implements *SQL-aware encryption schemes*. We make our prototype implementation for the Contiki OS [61] publicly available[3]. Moreover, Talos is a software platform enabling additional research on data protection and could be seamlessly integrated into various IoT application scenarios.
- We propose practical solutions to enable different cryptographic primitives in constrained devices, in particular for order-preserving and homomorphic encryptions. We introduce an optimization to Paillier (additive homomorphic encryption) tailored for use with integers, and propose an effective way of mapping integers to elliptic curve points to enable EC-ElGamal as an alternative additive homomorphic encryption scheme.
- We demonstrate experimentally the feasibility of Talos. We quantify the performance of Talos regarding energy, computation, communication, and latency. First, we microbenchmark the performance of the considered cryptographic primitives, both with and without hardware accelerator. Second, we quantify the overall system performance in Flocklab [151, 69], a public testbed, emulating IoT-typical scenarios.

This chapter is based on the contributions made in [206, 212, 213].

2.2 Overview

We now briefly discuss background information and the security model our system addresses. Alongside, we give an overview of Talos.

We design Talos with three main actors in mind (see Figure 2.1): *(i)* the user who is interested in the IoT data, for instance, the peak heart rate in the past month. *(ii)* the IoT devices where the data originates from. IoT devices are inherently resource-limited, specifically with regards to memory. Thus, it is necessary for IoT devices to offload their data into

[3]Talos can be downloaded from `https://github.com/hosseinsh/Talos`

a Cloud database regularly. In case the IoT device lacks any Internet-connectivity, the personal gateway (e.g., the smartphone for wearables) runs the Talos engine. *(iii)* the Cloud database, which stores IoT data securely and can process queries over encrypted data. Note that IoT devices would potentially need to query the data in the Cloud to make certain actuation decisions, for instance in case of automated heating systems. However, in our current design, we only consider the data producing (i.e., sensing) IoT device and address the data consuming (i.e., actuating) IoT device in future work.

IoT data consists of sensor readings (e.g., integer/float), meta-data (e.g., time, location, and identifier), and image/audio/video files. We consider text files to be significantly less represented in IoT data than in web and smartphone application data. Application developers should be aware of the sensitivity of data items and encrypt them with the adequate type of encryption. In Section 2.3, we detail the four main types of encryption and explain what functionality and security each type provides. We execute the data protection already at the IoT device, to reduce the attack surface and limit the need for trusted parties (see Figure 3.2). For the Cloud database, we rely on our extended version of CryptDB [184] which is tailored towards IoT-application scenarios.

2.2.1 Background

CryptDB brings together powerful cryptographic tools, to create an encrypted query processing system targeted for database management systems (DBMS). To remain transparent and seamless, the en/-decryption process is performed on a trusted proxy which has access to the keying material (see the upper part of Figure 3.2). The DBMS remains unmodified and is only extended with additional user-defined functions (UDF) which perform the encrypted query processing. CryptDB addresses the threat models of honest-but-curious [222] database administrators and the concurrent compromise of the application server, proxy, and DBMS. In the latter threat, only data of logged-in users is disclosed.

CryptDB leverages the fact that most SQL-like database queries are composed of simple mathematical operations, such as equality check, order comparisons, aggregation, and joins. To allow these operations over encrypted data, known cryptographic schemes, such as order-preserving and additive homomorphic encryptions, are employed. However, only a few data types must be encrypted with these encryption schemes, to enable query executions. Hence, most data items can be protected with efficient and secure symmetric key encryptions.

For SQL-aware encryption, CryptDB defines the following main encryption types: *random* with the highest security as it provides

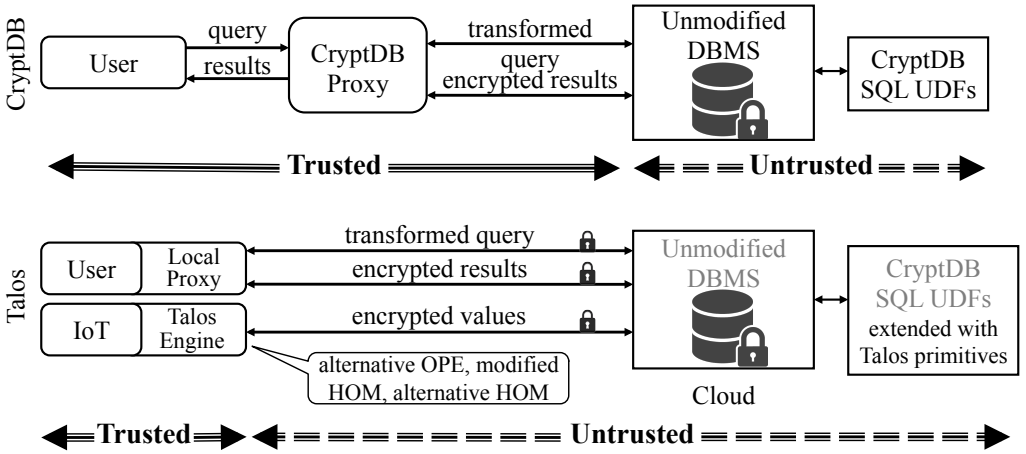


Figure 2.2: Talos extends CryptDB [184] to secure IoT data. Talos supersedes a trusted proxy with access to all keys. Instead, data protection is performed at IoT devices or personal gateways, e.g., smartphones.

semantic security (indistinguishability under an adaptive chosen-plaintext attack), *deterministic* which reveals equality information, *order-preserving encryption* revealing order information, *additive homomorphic encryption*, enabling addition over encrypted data, and *keyword search* over encrypted texts.

We explain these encryption types in more details in Section 2.3, as we elaborate on the specific encryption schemes we employ in Talos. Specifically, for order-preserving and additive homomorphic encryptions, which are computation- and bandwidth-intensive, we explore and utilize alternative approaches to reduce overheads and render them more efficient. We intentionally do not yet support encrypted word search, as we do not yet see the use case of this scheme for IoT data.

2.2.2 Security Analysis

CryptDB addresses two important threat models consisting of DMBS compromise, and a more severe one including the compromise of the application server, proxy, and DBMS. We not only inherit the security threat model addressed by CryptDB, but we address an even stronger database threat model, without a trusted proxy, as illustrated in the lower part of Figure 3.2. In addition, we address a network-based threat model. Note that IoT devices are vulnerable to physical node capture attacks. We do not address this attack specifically, however, we weaken it by utilizing a memory-protected area for key storage.

Threat 1: Cloud Database Compromise. The Cloud database provides confidentiality, i.e., secrecy of data, and no other security properties, such

as integrity, correctness, or availability. The attacker is assumed to be passive, i.e., with read-only access to all database data and to the RAM of the physical machines. The attacker is however not able to modify the queries nor the encrypted data. This threat is getting increasingly important in today's Internet, with the flourishing of third-party Clouds.

Unlike CryptDB, Talos is not designed to run with unmodified Web clients, but rather to facilitate an end-to-end integration with IoT devices. Therefore, we do not require a trusted proxy, thus providing stronger security than CryptDB. Talos provides the following guarantees: *(i)* at no time, the Cloud database has access to any keying material, *(ii)* during query processing, the data remain encrypted.

Note that the security of Talos is not perfect, as it reveals relationships among data items that allow for equality checks or ordering. Such data items, depending on the application scenario, are only column-wide. The remaining data items leak no further information, as long as encrypted with probabilistic encryption. Consequently, an attacker can learn the occurrence of a data item (e.g., via the histogram attack), however, he can not gain access to the actual plaintext.

This data leakage could be theoretically avoided by utilizing recent advancements in theoretical cryptography (i.e., fully homomorphic cryptosystem [82]) that enable any computations over encrypted data, without revealing any information. However, these approaches are still computationally expensive, rendering them impractical [183]. Talos is a practical system, providing strong security for most data items, while tolerating leakage of relational information for less sensitive data items.

Approach. Talos builds on the capabilities of CryptDB to allow processing of SQL-like queries over encrypted data. In addition to the CryptDB encryption schemes, we introduce a set of cryptographic primitives tailored for constrained IoT devices. This is in particular important for the order-preserving and additive homomorphic encryptions, which are the computationally most intensive schemes. We introduce and elaborate our findings and the resulting design decisions further in a dedicated section (Section 2.3).

Moreover, Talos assumes state-of-the-art database security mechanisms to be in place. For instance, the Cloud database should additionally store encrypted backups of the database. We discuss current research approaches aiming at providing Cloud security and related cryptographic approaches in Section 2.7.

Threat 2: Network-based Attacks. This threat targets communication between the IoT device and the Cloud. This threat can be carried out by passive or active attackers, as discussed in the following. A passive attacker can launch non-intrusive eavesdropping and traffic analysis

attacks. Talos addresses this threat by establishing a transport layer-based E2E secure channel between the IoT device and the Cloud (see Figure 3.2). This allows providing confidentiality, integrity protection, and authenticity of the data.

Talos utilizes weaker encryption schemes on database elements to enable encrypted query processing. With the secure E2E communication, we avert any relational data leakages, guarantee integrity protection, and authenticity of the data. An active attacker can launch a man-in-the-middle attack, to gain access to the plaintext values originated at the IoT device or make the IoT device believe it is communicating with the Cloud database (i.e., impersonation of the Cloud). Talos addresses this threat by a public-key-based authentication scheme for the E2E secure channel. Consequently, the IoT device can ensure the identity of the communicating peer and vice versa.

Approach. Datagram Transport Layer Security (DTLS) [166] provides confidentiality, authenticity, and integrity protection of communication. This is achieved using AES-CCM, which encrypts a given message while providing data-origin authentication and integrity protection. The main challenge with DTLS within constrained environments is the channel establishment, where the session keys are negotiated. The conventional approach is to rely on pre-shared keys. However, stronger security is obtained with the public-key-based version of DTLS [203, 117, 193]. Talos applies DTLS in the public-key mode with support of X.509 certificates and raw public keys, where the crypto operations could be accelerated by means of cryptographic accelerators. This way, both the IoT device and the Cloud database can authenticate each others' identity.

2.3 Design

Encrypted query processing is an emerging research field, enabling better protection of user's private data and resolving many privacy-related issues in Cloud computing. Inspired by the recent advancements in this field, we intend to bring the benefits of encrypted query processing to the IoT domain.

2.3.1 Encryption Types

Encrypted query processing allows storage of encrypted data at a third party database, yet simultaneously enabling efficient search and computation over the stored encrypted data. Although it would be desirable to utilize fully homomorphic encryption [82] to allow arbitrary

computations, we are yet bound to computationally feasible and efficient approaches, which enable only a subset of computations over encrypted data.

To support common SQL-like queries, it is necessary to be capable of performing equality checks and have knowledge about the order of encrypted values. However, enabling computation over encrypted data also means leaking information, i.e., any order-preserving encryption scheme will, by definition, reveal order relations. The encryption scheme is chosen per column and accounts for the intended query type, e.g., *min*, *order by*, etc. Data items that are not involved in the processing of queries are encrypted with the strongest cryptographic scheme (i.e., probabilistic encryption).

In the following, we describe the four types of encryption schemes supported in Talos.

RAND. Probabilistic or random encryption is the strongest security scheme, allowing no operation over encrypted data. This scheme is the conventional scheme, widely used in computing and storage. It has the property that the encryption of the same plaintext m results into two different ciphers c_1 and c_2 such that c_1 and c_2 are by no means related (i.e., semantically secure under a *chosen plaintext attack* (CPA)).

AES in CBC mode is a good candidate for this type of encryption. AES-CBC is a 16 Byte block-cipher encryption, producing outputs of size multiple times of the block-size. To this end, the input is, if necessary, padded to have a modulo 16 Byte size. Efficient hardware implementations of AES encryption routines are integrated into most IoT devices. Considering the fact that IoT data typically have a smaller size than AES's block size, the Blowfish block-cipher encryption [202] with 8 Byte blocks is a good alternative, producing smaller ciphers. Blowfish was as well a candidate for AES (Rijndael was selected for the final AES), but was considered inefficient for large file sizes due to the 8 Byte block-size. We selected Blowfish among several 8 Byte block ciphers (e.g., RC5, Skipjack) due to its higher efficiency [224]. Talos employs both Blowfish and AES in CBC mode. We apply the former for data smaller than 8 Byte and the latter for data larger than 8 Byte.

DET. Deterministic encryption allows for equality checks. The encryption of the plaintext m results always into the same cipher c.

AES-ECB is a block-cipher encryption with such a property. Due to this deterministic property it is in general advised not to use ECB for encryption of large packets, as an attacker can: *(i)* change the order of the blocks or replace a block in an indistinguishable manner (i.e., substitution attack), or *(ii)* learn information about the plaintext with a histogram of repeated blocks. Therefore, for maximum security, AES-ECB should be

only applied for plaintexts smaller or equal to 16 Byte. For bandwidth efficiency reasons, we employ Blowfish for plaintexts with a size smaller or equal to 8 Byte.

AES and Blowfish in CBC mode with a fixed initialization vector (IV) have the deterministic property, however due to undesirable leaks of prefix equality, they are not secure options for plaintexts larger than the block size. Therefore, we utilize AES-CMC [102] for plaintexts larger than 16 Byte, as recommended by CryptDB. AES-CMC is a tweaked combination of AES-CBC with a zero IV, where AES-CBC is applied twice on the input. The second CBC round is applied in the reverse order, i.e., from the last block to the first block. This way, the first blocks become deterministically random, and do not leak equality within a data item.

OPE. In *order-preserving encryption* (OPE) the order relationship between the plaintext inputs m_1, m_2, and m_3 is preserved after encryption, i.e.:

$$if\ m_1 \leq m_2 \leq m_3,\ then\ c_1 \leq c_2 \leq c_3 \tag{2.1}$$

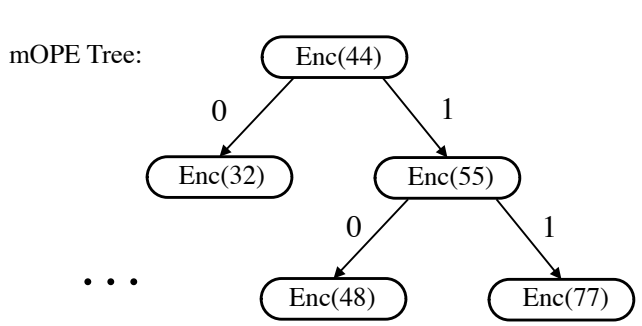
This way, the order information among the encrypted data items c_i is revealed, but not the actual data itself.

Order comparison is a common operation in SQL-like databases, e.g., for sorting, range checks, ranking, etc. OPE enables powerful operations and still offers relatively strong security, such that some research fields only focus on enabling secure databases or web applications using OPE [199, 41]. One of the first provably secure OPE schemes is the approach introduced by Boldyreva et al. [35]. This OPE scheme is, however, as computationally-intensive as asymmetric encryption.

To cope with resource constraints of IoT devices, we rely on a more recent interactive OPE approach by Popa et al. [183], which solely relies on symmetric cryptography and trades computation overhead for latency (i.e., it involves more communication). We refer to this lightweight OPE scheme, as *mutable order-preserving encoding* (mOPE) as the order encodings are mutable. Popa et al. [183] prove that mOPE fulfills the ideal security (IND-OCPA), i.e., no additional information than the order is revealed. mOPE is more secure than any other OPE approach and yet 1-2 orders of magnitude less computationally-intensive than traditional OPE schemes.

We detail the original mOPE and our optimizations to reduce the communication overhead further in Section 2.3.2.

HOM. Research on fully homomorphic cryptosystems has made significant advancements in the recent years, and been able to show that arbitrary computations on encrypted values are implementable [82]. However, the involved computations are yet prohibitively high [183] even for full-fledged devices and by far infeasible for resource-constrained



ciphertext	mOPE encoding **[path]**10..0
Enc(44)	**[]**10..0
Enc(32)	**[0]**10..0
Enc(55)	**[1]**10..0
Enc(48)	**[10]**10..0
Enc(77)	**[11]**10..0

Figure 2.3: Illustration of *mutable order-preserving encoding* (mOPE) with a balanced binary search tree [183]. The order encoding (path to the node) is appended with a 1 and 0s, signaling the end of the encoding.

devices. To support sum and average operations over encrypted data, it is, however, sufficient to utilize additive homomorphic encryption schemes, such that:

$$decrypt(c_1 \circ c_2) = decrypt(c_1) + decrypt(c_2) \tag{2.2}$$

Several cryptosystems exhibit homomorphic properties [74]. To start with, the textbook RSA and ElGamal's cryptosystem are multiplicatively homomorphic. Goldwasser-Micali's (GM) [92] scheme is among the first additive homomorphic cryptosystems achieving highest security level (i.e., probabilistic public-key encryption), which inspired several later cryptosystems. Unfortunately, GM exhibits a strong drawback as its input consists of a single bit plaintext. Moreover, the expansion of encryption results into large ciphertexts (i.e., $|pq|$ bits) which, given the single bit inputs, renders this scheme impractical. Benaloh [27] introduces a generalization of GM, which supports encryption of plaintexts with higher bit-length k. This, however, comes with a higher cost of decryption. The decryption cost is dependable of k, which eliminates the gain of a higher k. The Paillier [180] cryptosystem, one of the most well-known homomorphic schemes, improves the previous schemes by reducing the degree of expansion while allowing a large k (i.e., k is equal to key-length $|pq|$). At the same time the en-/decryption costs are reasonable (i.e., exponentiation and multiplication of big numbers modulo |pq|). Efforts [121] to reduce the encryption expansion of Paillier from 2 times k have the side-effect of significantly higher computation costs.

The Paillier [180] cryptosystem is employed by CryptDB. It is, however, with regards to IoT resources, computationally intensive and results into a large ciphertext size of 256 Byte, given a key size of 1024 bit (see Table 2.1). In Talos, we apply a slight modification to Paillier, inspired by Ge and Zdonik [81], rendering it more efficient in terms of average bandwidth and computation per data item. Moreover, we explore the

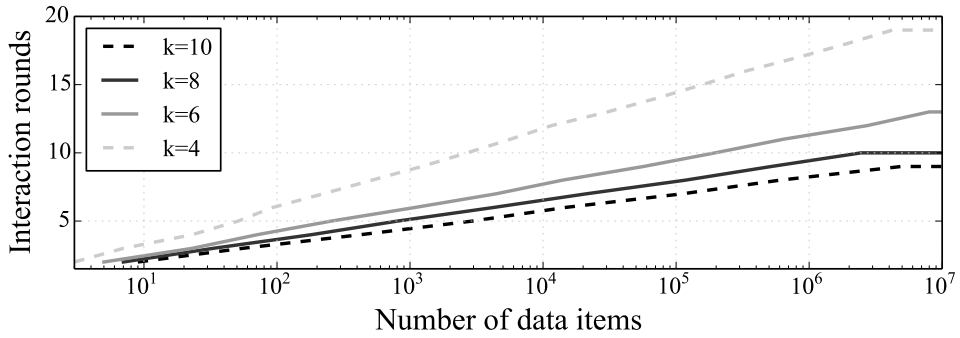


Figure 2.4: mOPE is an interactive approach. The number of interaction rounds depends on the current size of the database (i.e., the tree) and the parameter k of the k-ary tree in use. Talos selects k=10 in favor of fewer interaction rounds, resulting in higher average rewrites per item in the Cloud (e.g., factor 4.7 for 10^7 items).

elliptic curve (EC)-ElGamal encryption scheme as an alternative additive homomorphic encryption scheme.

In Section 2.3.3, we detail our modification to the Paillier cryptossytem, our findings about EC-ElGamal, and the efficiency of each approach.

2.3.2 Optimized Order-Preserving Encryption

The traditional OPE by Boldyreva et al. [35] is computationally five orders of magnitude more intensive than symmetric encryption. mOPE [183] is a recent interactive order-preserving encryption scheme that allows us to reduce this overhead drastically. mOPE utilizes lightweight symmetric encryption and balanced search trees to preserve the order information among ciphertexts. Intuitively, mOPE derives the order relations from the structure of the tree. A tree node holds a deterministically encrypted value where the order-preserving encoding is the path from the root of the tree to the node, as illustrated in Figure 2.3. For example *encrypt*(77) has the encoding 11 concatenated with 100000 (assuming 8-bit encodings) to indicate the end of encoding. The encoding reveals that *encrypt*(77) is the largest value in this tree.

mOPE is a client-server approach. The client intends to apply mOPE on a value, while storing it in a database. The server constructs the encoding, without learning the plaintext value, and later stores the final encoding in the database. For each new value, the server only learns the relation of the new value with regards to existing ones. The protocol starts with the client sending the new ciphertext to the server, accompanied with the request to `insert`. The server starts with sending the encrypted value at the root, to learn if the new value is larger or smaller. The client decrypts

Algorithm	Plaintext [Byte]	Ciphertext [Byte]
Blowfish-ECB	(0, 8]	8 (+ 8 RAND)
AES-ECB	(8,16]	16 (+ 16 RAND)
AES-CMC	(16, 16 + n]	$16 \times \lceil \frac{n}{16} \rceil$ (+ 16 RAND)
mOPE	(0, 8]	16 (8 + 8 Byte encoding)
Paillier (1024-bit key)	(0, 128]	256
EC-ElGamal (192-bit curve)	(0, 4]	50 (2x 25* Byte EC-points)
OPE [35]	(0, 8]	16

Table 2.1: Plaintext-space to ciphertext-space. *EC-ElGamal's ciphertext consist of 2 EC-points, which could each be compressed to 25 Byte. In RAND, the initialization vector (IV) is added to the ciphertext.

the values and replies. The server traverses the tree (i.e., in worst case $O(\log n)$ interactions) until it finds the right spot to insert the new value. As we show later in our evaluation in Section 2.5, the communication overhead of mOPE is lower than the computation cost of the traditional OPE.

To avoid long paths in the search tree, the tree needs to be rebalanced regularly. The server can rebalance without the help of the client, based on the order relation between nodes. However, as the encoding is based on paths in the tree, rebalancing results in mutated encodings. Hence, the name mutable OPE (mOPE). Note that the encoding is only used at the server side, to reveal ordering of encrypted data. The client does not store any encoding, as it would become obsolete after the next rebalancing.

The length of the encoding corresponds to the maximum depth of the tree, where the end of the encoding is signaled with a 10..0 padding, as depicted in Figure 2.3. We choose an encoding length of 64 bit and apply our previously described deterministic encryption strategy on the data items. This implies data items smaller or equal to 8 Byte are encrypted with Blowfish, data items between 8 and 16 Byte length are encrypted with AES-ECB, and data items larger than 16 Byte are encrypted with AES-CMC.

For the search tree implementation, a k-ary tree is used to achieve a lower number of interactions. This way, in each interaction round, the server sends the current node containing $\leq k$ data items. The client replies with an index and an equality flag. The index refers to the index of the data item, where the new value is equal or smaller than it. The flag indicates equality of the new value to the item at the index. In the latter case, the encoding of the existing value is taken for the new value.

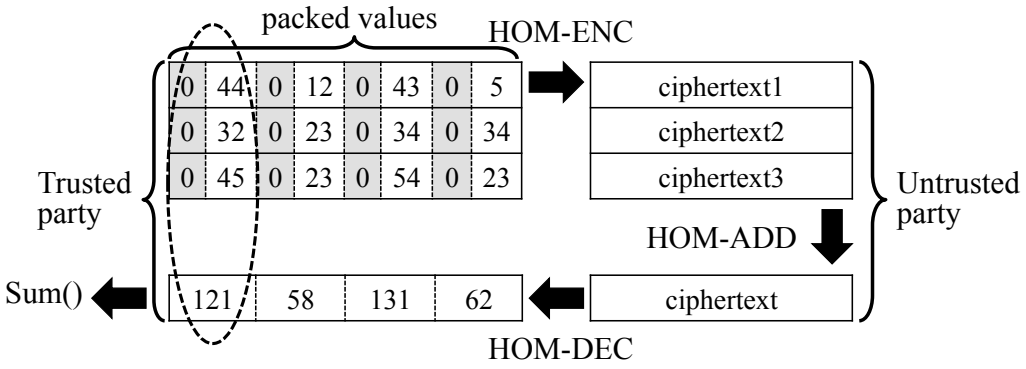


Figure 2.5: Illustration of our Paillier optimization. Several values are packed into one block, considered as one large number. The structure of packed values is maintained after decryption. After decryption, the sum of the final block is equal to the sum of all packed values.

Otherwise, the interaction continues until a leaf node is reached.

We select a 10-ary tree which offers a good trade-off between the maximum length of an interaction packet and the total number of interactions, as depicted in Figure 2.4. Increasing k from 4 to 10, allows us to reduce the average interaction rounds by more than half. Even though this increases the number of decryptions and comparisons per round, our results (see Section 2.5) suggest that the savings from having fewer interaction rounds outweigh this computational overhead. Note that a drawback of such tree-based interactive approach is that the worst-case number of interactions depends on the tree size. In our case the worst-case is $O(\log_{10} n)$ interactions (e.g., with 10^9 existing items the worst-case is 9 rounds).

In Talos, we rely on the prototype implementation of mOPE [183] and extend it with support for UDP and IPv6. To cope with the connectionless UDP, retransmission timers on both ends reassurs the termination of insert operations. More importantly, we adjusted the interaction protocol to be more concise. We transmit 2 Byte of header information appended with raw integer data (instead of ASCII representation of ciphertexts). This allows us to drastically reduce the communication overhead, up to a factor of 8. As summarized in Table 2.1, both OPE schemes produce final ciphertexts with the same length.

2.3.3 Optimized Homomorphic Encryption

The Paillier cryptosystem is an additive homomorphic encryption scheme which is based on asymmetric cryptography. We briefly explain the mathematical operations involved in Paillier, to be able to explain how we improve its efficiency with regards to encryption expansion.

The user defines the public key (n, g) and the private key d by selecting two large primes (p and q) of the same bit-length and g, a large random number modulo n:

$$n = pq, \;\; d = (q-1)(p-1) \tag{2.3}$$

Encryption of the message m is performed as follows:

$$c = (g^m)(r^n) \bmod n^2 \tag{2.4}$$

where r is a large random number modulo n. Decryption of cipher c is performed as follows:

$$m = L(c^d \bmod n^2)\ \mu \bmod n \tag{2.5}$$

$$\text{where } L(x) = \frac{(x-1)}{n}\ , \;\; \text{and } \mu = (L(g^d \bmod n^2))^{-1} \bmod n$$

The homomorphic addition function is defined by multiplication of the ciphers modulo n^2. Note, the ciphertext size is $2n$, i.e., for a 1024 bit key, 2048 bit (256 Byte).

To render Paillier more efficient, we apply a trick introduced by Ge and Zdonik[81] and illustrated in Figure 2.5. We use the fact that Paillier plaintext can be as large as $n = 1024$ bits, whereas IoT data often only consists of integers. The idea is to concatenate several values to form a single larger plaintext that will be encrypted in a single Paillier operation. We leave enough space for the carry bits of each value: <space, value_1, space, value_2, ... , space, value_i>. Assuming 32-bit values and 32-bit spaces, 16 values can be packed into a ciphertext.

This way, we amortize the computation and space overhead of Paillier among several values. This approach is possible since during the homomorphic addition operation (i.e., multiplication of ciphertexts) the aligned values are summed together (see Figure 2.5). A user interested in the sum of a data item, now receives a ciphertext of this form: <sum_1, .. , sum_i>. After the decryption process, she can extract any sum_i and ultimately compute the total sum.

EC-ElGamal. Paillier encryption is an expensive operation on IoT devices, where it can take up to 3.1 s (discussed in Section 2.5). As an alternative additive homomorphic encryption scheme to Paillier, we present in the following EC-ElGamal in more details. EC-ElGamal's security is based on the elliptic curve discrete logarithm problem (ECDLP). This means given two points P and Q on the curve, finding the scalar k such that P = kQ, is a hard problem. Note that the scalar multiplication kQ is calculated as k times the elliptic curve addition of Q

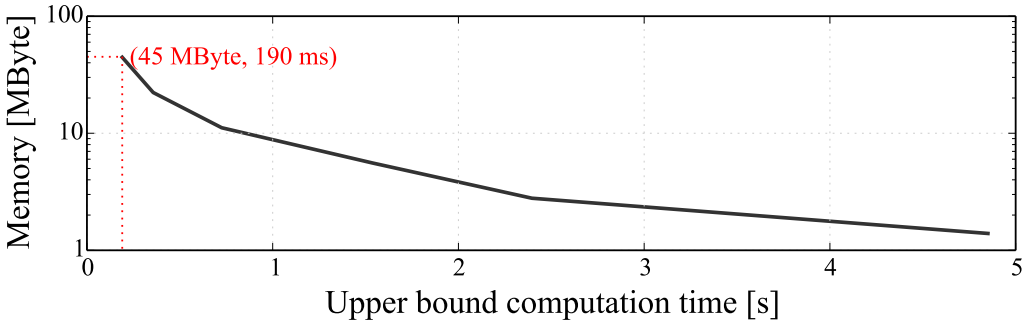


Figure 2.6: Memory-computation tradeoff in the Baby-Step-Giant-Step algorithm on a Google Nexus 5. Talos stores a pre-computed look-up table of 45 MByte and solves an ECDLP for an unsigned 32-bit value in maximum 190 ms with only one thread.

(for more details on elliptic curve cryptography (ECC) refer to [222]). In EC-ElGamal, after defining the elliptic curve (EC) parameters such as the generator point G, the encryption of message M (a point on the curve) is defined as two points on the curve C' and C'':

$$C = (C', C''), \text{ where } C' = M + rQ,\ C'' = rG \tag{2.6}$$

As it is common in ECC, $Q = dG$ is the public key, d the private key, and r is a large random number. Decryption of a cipher C is performed as follows:

$$M = C' - dC'' \tag{2.7}$$

To perform the homomorphic addition operation, the cipher components, which are each EC-points, are added to each other (i.e., EC-point addition):

$$\begin{gathered} C_1 + C_2 = (C_1',\ C_1'') + (C_2',\ C_2'') = \\ (M_1 + M_2 + r_1Q + r_2Q,\ r_1G + r_2G) \\ M_{1+2} = C'_{1+2} - dC''_{1+2} = \end{gathered}$$

$$M_1 + M_2 + r_1Q + r_2Q - d(r_1G + r_2G) \tag{2.8}$$

where $rdG = rQ$, since $Q = dG$.

Representation of Plaintext as EC-Point. A challenge in making practical use of EC-ElGamal is that it operates on EC-points rather than arbitrary messages. Efficiently and deterministically mapping of an arbitrary message into an elliptic curve point is an open research problem [137]. Koblitz suggests encoding of a message m into the x-coordinate of an elliptic curve [137]. This approach, however, is not homomorphic and therefore not suitable for EC-ElGamal.

We use a theoretical assumption from cryptography [222], which becomes practical in IoT scenarios, where we often deal with small integers, i.e., 32-bit. To map an integer m to an EC point M, we multiply m to a publicly known point G on the curve, i.e., $M = mG$. Such scalar EC multiplications (i.e., m times addition of G) can efficiently be performed on an IoT device. This approach is homomorphic, since

$$dec(C_1 + C_2) = M_1 + M_2 = m_1G + m_2G = (m_1 + m_2)G \tag{2.9}$$

At decryption time, we need to map M back to m, only with the knowledge of G. This requires solving an elliptic curve discrete logarithm problem (ECDLP). Although this is computationally infeasible for large numbers, solving it for a 32-bit integer m can be realized in a reasonable time (see Figure 2.6). Using the Baby-Step-Giant-Step algorithm [216], Talos can retrieve m from M in maximum 190 ms (benchmarked on a Google Nexus 5, a typical device where decryption takes place). To achieve an upper computation bound of 190 ms, a pre-computed look-up table of 45 MByte is necessary. In Chapter 3, we introduce an optimization (detailed in Section 3.5.2) to the EC-ElGamal encryption scheme that renders the decryption time to only a few milliseconds, while as well enabling efficient decryption of larger plaintext values, i.e., 64-bit integers.

Note that this procedure does not affect the overall security: we solve the ECDLP to obtain m from M, but M itself is protected with a strong cryptography, in our case 192-bit ECC (more secure than 1024-bit RSA).

2.3.4 Access Control

In the following, we briefly present key components of the access control mechanism adopted in Talos.

Authorization. To ensure that only authorized entities can access/add data in the Cloud database, Talos employs the OAuth2 protocol [103] to grant IoT devices authorized access to the Cloud database. In the OAuth2 protocol, the IoT device initiates a request to the Cloud. Consequently, the Cloud replies with an authentication URL, which is used to authenticate the user to the Cloud. After a successful login, the user can define the type and duration of the authorization. The next connection request from the IoT device is answered with an access token used for subsequent Cloud connections.

Key Management. Talos foresees the storage of the master secret by the user. This master secret is used to derive all the keying material used to protect the data. A PRF (Pseudo-Random Function, e.g., SHA-256) can be used to generate i deterministic keys key_i to be used by the IoT devices. For this we use a *key chaining* approach [184] that

Component	ROM [Byte]	static RAM [Byte]
Cryptographic accelerator	312	-
BigNumber operations*	2,832	-
EC operations*	1,144	-
ECDSA* + ECDH*	1,840	884
EC-ElGamal*	840	644
Paillier encryption*	712	780
mOPE client	2,322	1,396
AES (ECB, CBC, CCM)*	1,820	16
Blowfish (software)	4,548	4,168
SHA-256*	660	32
Subtotal	17,030	7,920
DTLS engine + client	16,942	7,370
Sum	33,972	15,290

Table 2.2: Memory size of cryptographic components of Talos on OpenMotes (considering max sizes). *All algorithms based on hardware crypto accelerator can be substituted by software implementations.

concatenates a well-defined identifier (ID), e.g., column-name or data type, to the master secret (MS):

$$\text{PRF}(\text{MS}|\text{ID}_i) = key_i \tag{2.10}$$

The master secret never leaves the user device. The derived keys are securely placed on the corresponding IoT devices.

In case *key revocation* is needed, for instance in case of disposed/compromised IoT devices, the user revokes the access token of the IoT device and re-encrypts the data in the Cloud database with a fresh key. Moreover, *data sharing* is a relevant feature that could be integrated into Talos. We introduce a novel secure data sharing scheme in Chapter 3.

2.4 Implementation

We have implemented a prototype of the Talos system. Our prototype implementation consists of two main components: *(i)* The IoT component of Talos is implemented for OpenMotes[4] in the Contiki OS 2.7 [61] and *(ii)* the Cloud database component is an extended implementation of

[4]OpenMote platform: `openmote.com`

Task	current [mA]	σ
CPU idle @32 MHz	12.8	0.11
CPU active @32 MHz	20.8	0.11
AES/SHA hardware accelerator	23.6	0.13
Public-key hardware accelerator (CPU idle)	25.9	0.12
Radio Transmission (TX), CPU idle	24	-
Radio Reception (RX), CPU idle	20	-

Table 2.3: Current drawn during computations on OpenMotes. Given the supply voltage (2.1 V) and the duration of a task, we calculate the drawn energy: time [s] × voltage [V] × current [mA] = energy [mJ].

CryptDB [184]. For space reasons, we discuss only the former in more details.

OpenMotes are based on the TI CC2538 microcontroller [229], i.e., 32-bit ARM Cortex-M3[5] SoC at 32 MHz, with a public-key cryptographic accelerator running up to 250 MHz. They are equipped with 802.15.4 compliant RF transceivers, 32 kB of RAM and 512 kB of ROM.

Talos is platform independent and our findings can be applied to any other platform. We chose the OpenMote as our prototype platform, because it has a public-key cryptographic accelerator (including SHA-256) on board and due to the promising potential of 32-bit platforms in next-generation embedded platforms [141, 6]. Moreover, low-priced and energy-efficient cryptographic accelerators are predicted to find their ways into low-power platforms [181, 138]. We assume the on-device random number generators to be secure and robust [146]. For the cryptographic accelerator, we implement generic drivers for big number arithmetic operations (utilized by RSA and Paillier) and ECC operations (utilized by ECDSA, ECDH, and EC-ElGamal). In case no cryptographic accelerator is available, these fundamental operations are provided by a software implementation. We used the crypto library Relic Toolkit[9] for this purpose.

While relying on the cryptographic accelerator Talos requires 2.4 kB of RAM and 10 kB of ROM for the crypto components. In case no cryptographic accelerator is on board, a considerable amount of memory is dedicated to Relic Toolkit. The exact memory size is dependent on several configuration parameters to optimize Relic Toolkit. It is however in the range of 8 kB of RAM and 66 kB of ROM. The breakdown of memory requirements in Talos is shown in Table 2.2.

[5]Fitbit Tracker utilizes a microcontroller with similar capabilities.

States	Time	Energy
Public-Key Crypto (ECDSA, ECDH)	1,191.75 ms	59.1 mJ
CPU	9.46 ms	0.5 mJ
TX	47.73 ms	3.4 mJ
RX	43.46 ms	2.2 mJ
Symmetric Crypto (AES-CCM, SHA)	5.86 ms	0.48 mJ
Total (without idle)	1,232.8 ms	65.4 mJ
idle/sleep (1-2 wireless hops)	455 ms - 2 s	< 2.52 mJ

Table 2.4: Secure E2E channel establishment (certificate-based DTLS handshake) on the client. The number of wireless hops affects the idle time.

We use the tools `arm-none-eabi-readelf` and `arm-none-eabi-size` to perform our memory analysis on the binaries. Hereby, DTLS makes a major contribution with 7 kB of RAM and 17 kB of ROM. We measure a maximum stack size of 2 kB. It is important to mention that our memory values cannot be directly translated for 8- or 16-bit platforms, as our platform comes with 32-bit registers. Note that recent works, such as SPLIT [244] support on-demand loading of DTLS functionality stored as modules on the significantly less constrained flash storage, hence reducing the maximum RAM requirements due to DTLS.

We assume hardware AES block encryption to be available (which has been integrated into most RF transceivers for more than a decade). We extend existing drivers for AES to support additional modes, such as the AES-CMC for deterministic encryption. Unfortunately, Blowfish is not supported in hardware. Our ported Blowfish implementation requires 380 Byte of RAM and 4168 Byte of ROM. For DTLS integration, we modify the tinyDTLS[6] implementation to support by demand cryptographic accelerator or software implementation (i.e., Relic Toolkit). Moreover, we extend it with a basic X.509 certificate parser.

2.5 Experimental Evaluation

In this section, we present the experimental evaluation of Talos on OpenMotes, representing a typical IoT device. We do not discuss the performance of the Cloud database, represented by a modified and extended version of CryptDB instance, as it is not the core of our contribution. However, we quantify the network overhead of Talos during its interaction with the Cloud database.

[6]tinyDTLS; `https://projects.eclipse.org/projects/iot.tinydtls`

In the following, we first define our evaluation objectives, describe the setup, metrics, and methodology. We continue in Section 2.5.1 with a brief discussion of our results for secure E2E communication, followed by a detailed discussion of our results for encrypted data processing in Section 2.5.2.

Goals. Throughout this section, we intend to answer the following questions: *(a)* is Talos feasible on resource constrained devices? *(b)* what is the overhead of Talos in terms of computation, energy, and bandwidth? *(c)* what is the impact of the availability of cryptographic accelerators on the performance and feasibility of Talos?

Evaluation Setup. For our evaluation, we rely on Flocklab [151], a public wireless sensor network testbed. Flocklab[7] supports over the Internet communication of sensor nodes with a remote host, thus emulating IoT scenarios. Our Cloud database resides on a normal desktop connected via the Internet to Flocklab nodes, with an average transmission delay of 10 ms. Our communication setup involves at least one wireless hop within Flocklab.

Metrics. We quantify the impact of Talos in terms of computation and communication overheads, and calculate the corresponding energy consumption. All tests are repeated at least 100 times, if not indicated otherwise. Standard deviation is reported only when not negligible, since it is mostly the case with CPU computation on a non-preemptive OS. Inspired by TinySec [130], we use the metrics *Byte-time* and *Byte-energy* to put computation in relation to radio transmission time and energy. In other words, we normalize our time measurements based on the transmission time of 1 Byte in 802.15.4 (i.e., 32 μs) and the energy measurements based on the energy required to transmit 1 Byte on OpenMotes (i.e., 1.613 μJ). The latter is calculated as: transmission time $\times$ voltage $\times$ transmission current = 32 μs $\times$ 2.1 V $\times$ 24 mA = 1.163 μJ.

Methodology. We verify the current draw in our crypto functions by utilizing a mixed signal oscilloscope, as summarized in Table 2.3. We then characterize the accuracy of system clocks, which we use for time measurements[8]. To this end, we use digital inputs of our oscilloscope connected to OpenMote pins to encode the start and end of events. We rely on Contiki's timer with a resolution of 30 μs and a dedicated hardware timer with an accuracy of 1 μs. We measured a maximum timer inaccuracy of 0.4%. Additionally, we leverage the energy measurement features of Flocklab during our evaluation.

[7] For our project, we extended Flocklab with OpenMotes: `flocklab.ethz.ch`

[8] We open-source our test interface utilizing accurate system timers: `https://github.com/hosseinsh/Talos`.

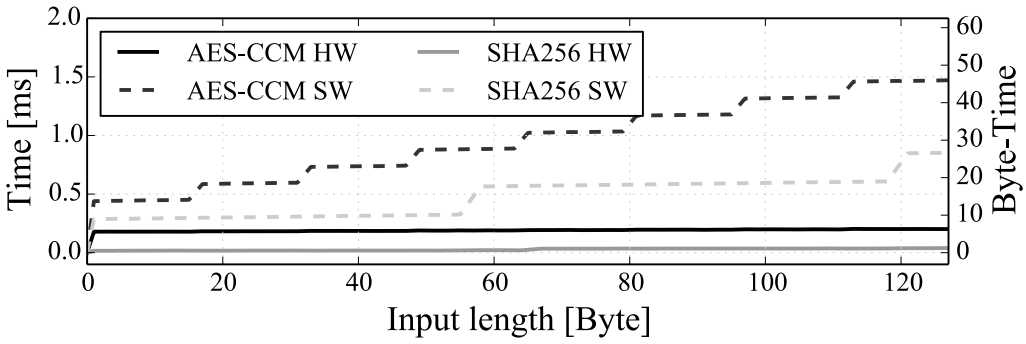


Figure 2.7: Secure E2E channel (AES-CCM, SHA256) with support of cryptographic accelerator (HW) compared to pure software implementation (SW).

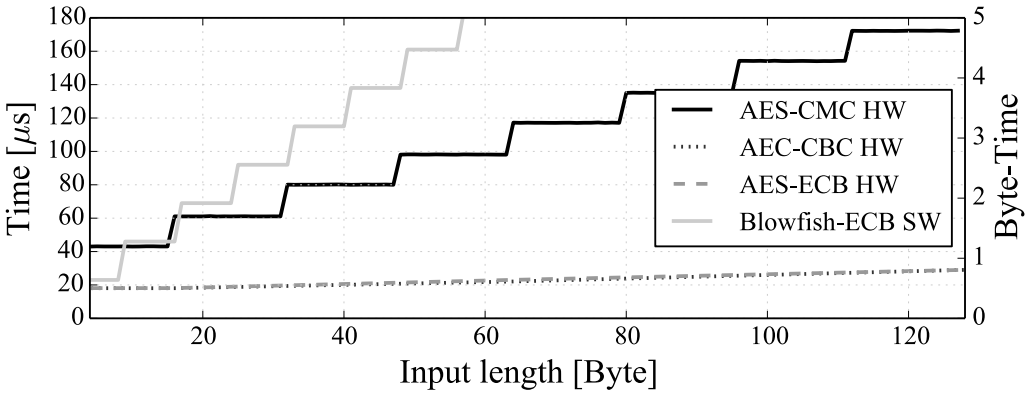
2.5.1 Secure E2E Communication

Talos employs the certificate-based DTLS to establish a secure E2E channel. Due to space limitations, we only briefly discuss the DTLS results.

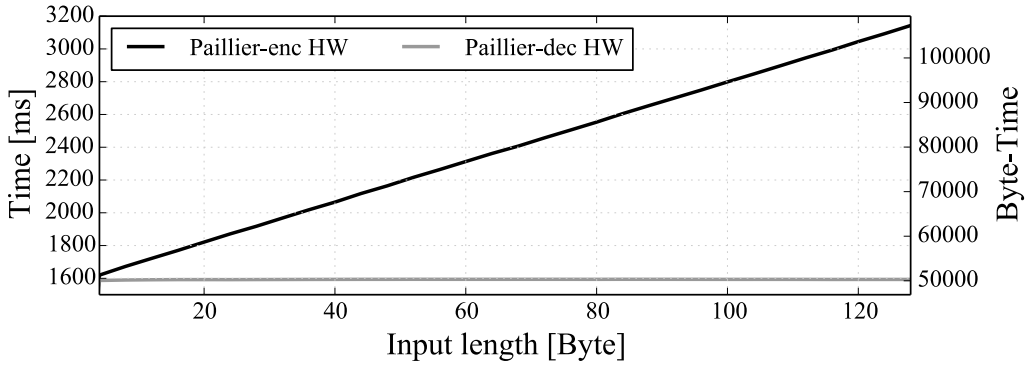
In our setup, a DTLS client from Flocklab establishes a secure channel with the Internet host. The handshake, since it involves public-key-based operations (ECDSA, ECDH), is the most expensive part of a secure E2E channel. The public-key-based operations contribute to the major part (90%) of the total energy consumption per handshake (65.4 mJ). Hence, a DTLS session should be kept alive as long as possible, for instance by means of session resumption [118]. Note, the cryptographic accelerator allows concurrent computations on the main CPU. Hence, during the long computations ideally other tasks could be scheduled.

Table 2.4 lists the computation time and energy costs of the different components of a handshake. The total duration of the handshake is mainly impacted by the number of the wireless hops, as indicated by the time spent in *idle* state. How much energy is drained in this state depends on the specific MAC layer in use, e.g., the level of radio duty cycling and the sleep mode.

Once the session is established and the keying material is agreed upon, a modest crypto overhead is caused by AES-CCM. AES-CCM has a performance between 31 to 55 μs for 16 to 128 Byte packets, as depicted in Figure 2.7. This overhead is considerably larger when using software (i.e., 293 to 1311 μs). Although the crypto accelerator draws about 13% more current, its faster execution time leads to energy savings by a factor of up to 20 for full frames (see Table 2.6). More importantly, the computation of AES-CCM in hardware can run in parallel to the transmission of the preamble (8 symbols $\times$ 16 μs = 128 μs)



(a) Symmetric block cipher encryption, as used in random and deterministic encryption schemes. Except for Blowfish, which is implemented in software (SW), the remaining ciphers utilize the cryptographic accelerator (HW).



(b) Additive homomorphic encryption by means of Paillier utilizing the cryptographic accelerator (HW). Paillier's plaintext-size can be as large as the key size, in our case 1024 bit (128 Byte).

Figure 2.8: Microbenchmark of the cryptographic algorithms in Talos on a typical IoT device (i.e., OpenMote).

Note that the steps in Figure 2.7 in the software implementation are due to padding to full block sizes. AES has a block size of 16 Byte, hence the steps are at 16 Byte steps. SHA has a block size of 64 Byte. The effect due to padding is optimized in the hardware-based modes.

2.5.2 Encrypted Data Processing

To support encrypted data processing, Talos utilizes four types of encryption. The strongest level of security is provided by probabilistic encryption (RAND), additive homomorphic encryption (HOM), followed by determinist encryption (DET), and order-preserving encoding (OPE). Each of these schemes comes with certain security-functionality tradeoffs, discussed in Section 2.3. In the following, we first discuss the performance

Algorithm	Input Size [Byte]			
	4-8	16	64	128
AES-ECB	-	0.9 μJ	1.1 μJ	1.4 μJ
-CBC	-	0.9 μJ	1.1 μJ	1.4 μJ
-CMC	-	2.1 μJ	4.8 μJ	8.5 μJ
Blowfish-ECB	1 μJ	2 μJ	8 μJ	16 μJ
Paillier	88-91 mJ	99 mJ	128 mJ	171 mJ
EC-ElGamal	11-23 mJ	46 mJ	184 mJ	368 mJ

Table 2.5: Energy consumption of data-protection components of Talos based on the *current* values in Table 2.3. Except for Blowfish, all crypto operations utilize the hardware crypto engine.

of the individual crypto algorithms, and then elaborate on the overall system performance.

2.5.2.1 Microbenchmarks

We now discuss the time and energy measurements of the individual crypto algorithms, as summarized in Figure 2.8 and Table 2.5, respectively. The performance results combined with the ciphertext overheads of each algorithm (see Table 2.1) contributed to the design decisions in Talos.

RAND/DET. For random and deterministic encryptions, we employ various types of symmetric block cipher encryptions (AES and Blowfish). We utilize the cryptographic accelerator for AES modes, however, our Blowfish is software-based. Blowfish is known for its long initialization phase after setting the key, which in our case amounts to 12 ms, independent of the key size. Since in Talos the key is rarely changed, this overhead is acceptable. Blowfish with 23.5 μs shows a modest performance, in the same order as AES-ECB (14 μs) and AES-CBC (18 μs). Note that AES in software is by factor 3 to 10 slower, depending on plaintext size.

As shown in Figure 2.8(a), Blowfish, which has a blocksize of 8 Byte, quickly becomes expensive for large data. Consequently, Talos uses Blowfish for data ≤8 Byte. Among the different AES modes (ECB, CBC, CMC), AES-CMC has the highest overhead. This is because AES-CMC applies AES-CBC twice to avoid early block leakage. Talos limits the use of AES-CMC only for data ≥16 Byte.

OPE. The traditional OPE, as introduced by Boldyreva [35], relies on the hypergeometric distribution (HGD). HGD is computationally intensive and in similar orders as Paillier. Talos utilizes mOPE which is about two

Communication	Engine	Input Size [Byte]		
		16	64	128
AES-CCM	Hardware	8.7 μJ	9.4 μJ	10 μJ
	Relic	19.7 μJ	38.8 μJ	64 μJ

Table 2.6: Energy consumption of AES-CCM. The comparison to software implementation shows the energy gain of hardware accelerator.

orders of magnitude more efficient than OPE. This is because that mOPE employs deterministic encryption (in our case Blowfish). Since mOPE requires interaction with the Cloud, we elaborate on its performance aspects in the following section.

HOM. Additive homomorphic encryption is the most expensive cipher in Talos. Paillier encryption starts with 1,619 ms for 4 Byte plaintext and increases linearly to 3,142 ms for 128 Byte plaintext. Decryption time is constant at 1,593 ms. Note that decryption is typically performed on more powerful devices. Paillier's performance in software is by factor 6 slower.

With EC-ElGamal we explored an alternative additive homomorphic encryption to the Paillier cryptosystem. EC-ElGamal's performance, since based on EC-points, is input-length independent. EC-ElGamal encryption takes 210 ms, whereas the decryption with 95 ms is by factor 2 faster. The 210 ms encryption time already includes a maximum upper bound of 20 ms for mapping a 32-bit value into the EC space (8-bit values require only 9 ms).

The homomorphic encryption with both Paillier and EC-ElGamal is energy intensive (see Table 2.5). For 4 Byte plaintext, this amounts to 88 mJ and 11.42 mJ, respectively. Note that this is equivalent to 76 and 9.8 *kB-energy*, respectively. Paillier's energy consumption in software is by factor 7.5 higher, which renders it significantly costly and not suitable for IoT.

2.5.2.2 System Performance

We now assess the overall performance of Talos with focus on the two schemes of order-persevering and additive homomorphic encryptions. Moreover, we discuss the impact of Talos on the lifetime of a battery-based IoT device.

mOPE. With mOPE we trade computation for communication. In Section 2.3.2, we discussed how to reduce the number of interactions in mOPE by tuning the k-ary tree to hold up to 10 values (k=10). Figure 2.9 depicts the total time of interactions in mOPE, based on the number

of interaction rounds for k=4 and k=10. The interaction time is impacted by the transmission delay of our setup, which consists of one wireless hop.

The average roundtrip time (RTT) per interaction is higher for k=10 (68 ms) than k=4 (53 ms) because packets carry in average more elements. Similarly, the per interaction CPU time is for k=10 (314 μs) 19% higher than k=4 (264 μs). This is because on average more item decryptions are needed for the comparisons. However, in total is k=10 several times more efficient than k=4. As depicted in Figure 2.9, with 10^3 items in the database, mOPE with k=4 requires 9 interactions, whereas k=10 needs only 4. This is 200 ms faster than k=4. This trend continues and we experience with 10^5 items in the database 12 interactions for k=4 and only 6 interaction for k=10 (250 ms faster).

The IoT device can ideally utilize radio duty cycle techniques to reduce its energy consumption to an optimal of transmission and reception of the query and response packets of the interaction process. Assuming more than 10^5 items in the database, adding new items requires 6 interaction rounds which involves transmission and reception of 12 packets. This results into an energy consumption of 1.3 mJ. Assuming an optimistic lower bound of 1,600 ms computation time for the traditional OPE, shows that mOPE is 2 orders of magnitude more efficient than OPE.

HOM. Paillier, as discussed earlier, is computationally very expensive (encryption time is between 1.6 to 3.1 s). Its cost is prohibitive, especially when compared to the encryption time of EC-ElGamal (210 ms), an alternative HOM. To render Paillier more efficient, we optimized it to pack several values into a single plaintext. This amortizes the expensive Paillier encryption among several values. As depicted in Figure 2.10, the energy per item decreases as the number of packed items increases.

EC-ElGamal, which operates on 32-bit plaintext values, consumes 11.42 mJ, and thus depicts the lower energy budget for HOM encryptions. In order to reach the same per item energy efficiency as EC-ElGamal, our optimized Paillier must pack fourteen 32-bit plaintext values. Working with 16-bit plaintext values allows packing 32 items in one plaintext. This results in optimal energy efficiency, by a factor 2 better than by EC-ElGamal. The possibility of packing values, however, is application scenario-dependent. To remain as generic as possible, we use EC-ElGamal as the default HOM for Talos.

Energy. Talos significantly impacts the lifetime of a battery-based IoT device, specifically due to the involved crypto operations. This is an inevitable tradeoff that comes with higher security. We assume two AAA alkaline batteries with a typical capacity of 3 Wh (10.8 kJ) and a targeted lifespan of one year. This results in a daily energy budget of 29.6 J.

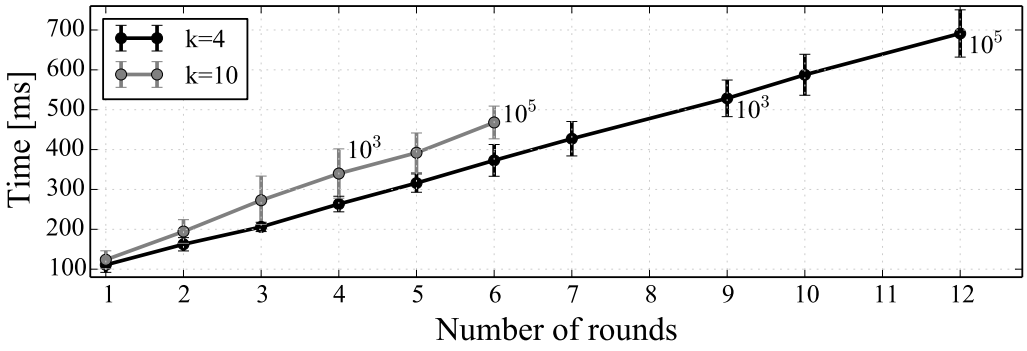


Figure 2.9: mOPE. Time of an insert operation based on the number of interactions for up to 10^5 items. Client is one wireless hop away from the gateway in Flocklab. Average RTT per interaction is 53 ms for k=4 and 68 ms for k=10. The per interaction CPU time is 264 μs for k=4 and 314 μs for k=10.

To put this into perspective, with 20% of the daily budget (5.92 J), Talos is capable of performing: 90 DTLS handshakes, 518 EC-ElGamal based homomorphic encryptions, 5.92×10^6 random or deterministic encryptions of 32-bit values, or 5381 mOPEs (5 interactions).

Case Study. To have a better understanding about the applicability of Talos, let us again consider the application scenario of the health monitoring device which logs heart rate, location, and timestamps. As we described in Section 4.1, the logged items have different sensitivities. For instance, the heart rate measurements have the highest sensitivity, since they can be used to infer health-related information (e.g., stress, depression, or diseases). Hence, Talos protects heart rate with additive homomorphic encryption to provide semantic security and allow average and summation computations. Health monitoring devices[9] typically sample at 1 Hz (every 1 s) during sports activities and 6 times per hour otherwise. For a person with 1 h sports activity per day, this would result in a daily 3,738 data items encrypted with additive homomorphic encryption. The location, logged every 15 min, is maybe less sensitive to this person and could be encrypted with deterministic encryption, to allow encrypted queries correlating heart rate with a given location. Hereby, the timestamp for each heart rate record could be encrypted with the order-preserving encryption, to allow ordering, but not revealing the actual time.

Assuming the same underlying platform we used in our evaluation, the total daily energy cost of additive homomorphic, deterministic, and

[9]Microsoft Band: `http://www.windowscentral.com/how-often-microsoft-band-checks-your-heart-rate`

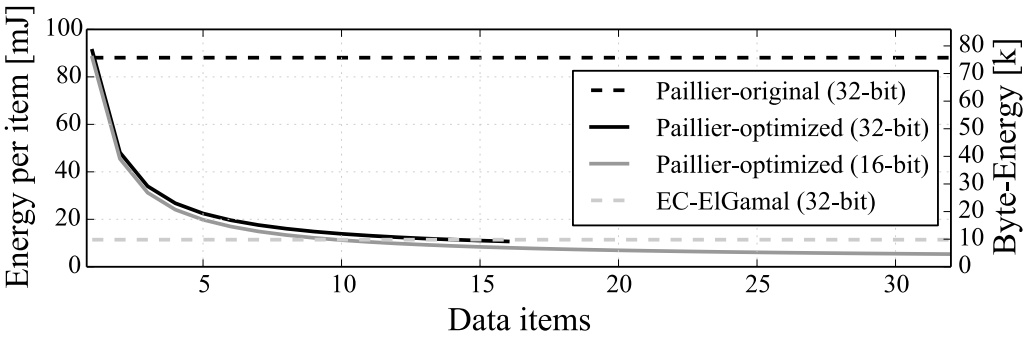


Figure 2.10: Additive homomorphic encryption. Our optimized Paillier is more efficient than original Paillier and can reach the efficiency of EC-ElGamal.

order-preserving encryptions in this scenario accounts for a total of 42.67 J per day. This would contribute to a modest 5.4% of the daily energy budget of a Fitbit device with five days lifetime and a lithium-polymer battery capacity of 1.2 Wh (4.32 kJ).

2.6 Discussion

This work provides a proof-of-concept of the potential and feasibility of building secure systems that address data privacy concerns in the IoT ecosystem. However, more research is needed to realize the full potential of Talos. Here we address some practical challenges and research points that assist in evolving Talos further.

Energy. The feasibility of Talos for IoT devices is driven by its energy efficiency. Although security comes at a price, the price should not hinder the usability of the system. We show that with a modest energy budget, Talos can be integrated on IoT devices providing strong network and data security guarantees. We achieve this by making the employed HOM and OPE schemes one order of magnitude more efficient. However, there is still potential for further optimizations, specifically for HOM, which with EC-ElGamal still accounts for a considerable amount of the energy consumption.

Hardware Accelerator. We explore the impact of hardware crypto accelerator on the feasibility of Talos. The hardware accelerator, which utilizes a separated core (running at 250 MHz), drains with 26 mA a higher current than the main core at 32 MHz (20 mA). The higher frequency and consequently lower computation time result in significant energy savings (by a factor of 2 to 20), as compared to pure software operations.

More importantly, a separate core for crypto operations allows utilizing the main core for other tasks. This results in an active main core, instead of idle, as considered in our evaluation (draining a total of 33 mA for both cores). Hence, an optimal task scheduling would increase the energy efficiency of our system several times higher.

Security Analysis. Talos meets the security goals and efficiency requirements outlined in Section 2.2.2. With our efficient public-key-based E2E secure channel (DTLS), Talos defends against network-based threats. The encryption applied by Talos protects the data against curious database administrators and database compromises. Since we utilize order-preserving encryption and deterministic encryption, we allow leakage of order and equality information for less sensitive data. Sophisticated attacks could potentially misuse the gained information from this leakage. Hence, future steps should address this shortcoming of Talos.

Alternative Crypto Primitives. Recent advances in theoretical cryptography and careful optimization of crypto mechanisms allow us to build practical secure systems, such as Talos, in a novel fashion that inherently address data privacy issues through facilitating computation on encrypted data. We are currently witnessing improvements in the computational efficiency of fully homomorphic encryption [83, 85]. The already achieved improvements of 6 orders of magnitude in the past decade could be further enhanced in the near future. Talos is yet bound to schemes with reasonable efficiency, however, our general system design is not bound to particular schemes.

Fully Homomorphic Encryption (FHE). In Talos, we rely on additive (rather than fully) homomorphic encryption. We measured additive homomorphic encryption to be 5 orders of magnitude slower than AES, which makes it the bottleneck of our system. Hence, despite recent efforts in rendering FHE more efficient, we do not foresee FHE to be soon feasible for IoT devices. However, with GHS [85], an approach of homomorphic evaluation of AES circuit, there is hope that the benefits of FHE can find their way into IoT. GHS transforms AES ciphertexts, without access to the plaintext, into an FHE-compatible form where arbitrary computations over the hidden plaintext are possible.

2.7 Related Work

We now discuss work related to Talos grouped in the following four categories:

E2E Security. Early efforts to bring security to WSN explored low-power cryptographic approaches [152, 228], which facilitated research on secure communication protocols for IoT. Hummen et al. [118, 119, 203] introduce a handshake delegation scheme which allows highly constrained devices without the capability of performing public-key-based operations to still benefit from the scalability and security of public-key-based handshakes. Hu et al. [114] investigated the feasibility and advantages of hardware accelerators on low-power devices. TLS–Rotate and Release [237] allows the auditing of secure communication channels, such that device owners can decrypt and verify recent TLS traffic of their devices. This allows owners (e.g., security experts and consumer watchdogs) to audit the communication regarding the type and extent of transmitted data.

Privacy-Preserving Cryptography. There has been significant amount of work on cryptographic schemes [154, 91, 37, 215, 90, 26, 38] that could be utilized in privacy-preserving computations. Gentry's work [83, 82] marks a breakthrough in cryptography showing an implementable fully homomorphic encryption (FHE) scheme. Since then, his work has been incrementally enhanced up to 6 orders of magnitudes by the research community [85]. Prior to Gentry's work, the focus was on partial homomorphic encryption, where only one type of computation, such as multiplication or addition is supported [116, 39, 42].

Although FHE provides semantic security and supports at the same time arbitrary computations over encrypted data, it is not yet best suited for encrypted query processing. This is due to both its prohibitive cost and the fact that the Cloud must process all existing data in the database for queries such as equality check or comparison. This is the main reason for using weaker encryption schemes such as OPE and DET to allow the database to reduce the scope of computation.

Secure multi-party computation approaches [106, 238] are efficient for simple functions, however become computationally expensive for general functions. Moreover, MPC involves high interactions, large bandwidth, and coordination among the involved parties. Secure in-network processing of aggregate queries was introduced for WSNs [190, 52]. This would increase the security of approaches providing a distributed database interface for WSNs, such as TinyDB [155].

Differential privacy [62] assumes a trusted server, which obfuscates answers to avoid leaking information about data items and the query patterns.

Computation on Encrypted Data. Perrig et al. [223] introduced an efficient search over encrypted text files. This is achieved by deterministically encrypting metadata of files which are protected with strong encryption, i.e., probabilistic. Perrig et al.'s efforts paved the way

for more advanced systems enabling encrypted query processing [120, 184, 185], among them CryptDB [184] which we discussed in depth in Section 2.2. Mylar [185] introduces a multi-key searchable encryption scheme, exemplified for smartphone applications. Mylar protects the content of documents and the searched words from the untrusted server.

Goldwasser et al. [44] introduce an innovative and sophisticated approach for machine learning classification over encrypted data. This approach is complementary to ours and would allow support for a wider range of data types.

Cloud Security. Commercial Cloud database services, such as Google [95], encrypt data before storage (i.e., encryption at rest). However, the queries are still processed over plaintext data. Secure data storage is an essential measure and complementary to our approach. Utilizing a local *trusted machine* [17, 11] at the database is an alternative approach to encrypted query processing. This, however, implies that the user considers the *trusted hardware* trustworthy.

Secure deletion [194] of remotely stored data is a relevant area of research concerned with the ability to delete data irrecoverably. One approach is client-side encryption, such that to securely delete data, it is enough to delete the encryption key. In this case, encryption is not intended to provide confidentiality, but rather as a problem reduction where arbitrarily large data is securely deleted by deleting a fix sized key.

2.8 Conclusion

We presented Talos, a practical secure system that provides strong communication and data security features for privacy-preserving IoT applications. Talos leverages and tailors cryptographic primitives that allow computation on encrypted data without disclosing decryption keys to the Cloud. To achieve this, we utilize optimized encryption schemes, specifically for the expensive additive homomorphic and order-preserving encryptions, accelerating them by 1 to 2 orders of magnitude.

We show the practicality and feasibility of Talos through an implementation and experimental evaluation considering both micro-benchmarking and system performance. Talos copes with the limited energy budget of constrained devices and requires a modest energy budget to provide a higher security level. Advancements in *Computing on Encrypted Data* is increasingly significant to the progression of data privacy and security. Talos is the first system to address energy and computation concerns for integrating encrypted query processing into IoT systems. We hope that Talos facilitates further research in this direction.

3

Pilatus

IoT applications often utilize the cloud to store and provide ubiquitous access to collected data. This naturally facilitates data sharing with third-party services and other users, but bears privacy risks, due to data breaches or unauthorized trades with user data. To address these concerns, we present Pilatus, a data protection platform where the cloud stores only encrypted data, yet is still able to process certain queries (e.g., range, sum). More importantly, Pilatus features a novel encrypted data sharing scheme based on re-encryption, with revocation capabilities and in situ key-update. Our solution includes a suite of novel techniques that enable efficient partially homomorphic encryption, decryption, and sharing. We present performance optimizations that render these cryptographic tools practical for mobile platforms. We implement a prototype of Pilatus and evaluate it thoroughly. Our optimizations achieve a performance gain within one order of magnitude compared to state-of-the-art realizations; mobile devices can decrypt hundreds of data points in a few hundred milliseconds. Moreover, we discuss practical considerations through two example mobile applications (Fitbit and Ava) that run Pilatus on real-world data.

3.1 Introduction

The Internet of Things (IoT), through embedded devices and wearables, is enabling a whole new spectrum of applications. One of the prominent offerings in this domain is the emerging field of health and fitness tracking with over 150k [64] applications listed in the two major smartphone app

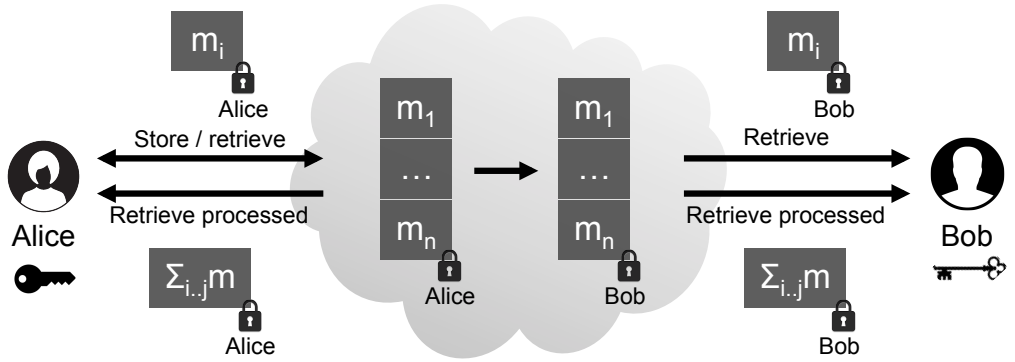


Figure 3.1: Pilatus can both query and share encrypted data. The cloud has access to no keys nor any plaintext. It is able to process encrypted data (e.g., range, sum queries), as well as to re-encrypt it, enabling crypto-protected sharing. We also address revocation, in situ re-keying, and group sharing.

stores. Examples of such applications include wristbands that can infer stress level from skin conductance, sport trackers that can log physical activities, and fertility apps.

The collected data typically consists of sensor readings (e.g., body temperature, conductance response), activity meta-data (e.g., duration, type), or health-related symptoms (e.g., migraine headaches, pain). For scalability, ubiquitous access, and sharing possibilities, the data is most often stored in the cloud. Transparent and secure data sharing (e.g., sharing with friends or domain experts) is considered a key requirement for the practicality and success of typical IoT systems [126, 210]. Moreover, securing the cloud storage is of utmost importance, as the data can be used to infer privacy-sensitive information, such as heart diseases, personal well-being, and fertility-related data [14, 48]. The privacy risks of today's data collection model are many, including systematic unauthorized disclosure of personalized data on clouds [18], for personal advertising [176], trading with insurances [59], and due to external [71] or internal data breaches (e.g., curious cloud employees [71]).

Challenges. *How to benefit from cloud computing (i.e., storage and query processing) without compromising data control and security?* Storing encrypted data with traditional symmetric encryption schemes, such as AES, would offer protection but render the data unsearchable and unshareable. Alternatively, homomorphic encryption schemes enable arbitrary computation on encrypted data but are presently impractical [184]. In this chapter, we focus on Partially Homomorphic Encryption (PHE) and in particular additive homomorphic schemes.

These are practical solutions that enable an important set of queries [231], such as the `sum` query on encrypted data – a common operation in IoT applications when history data needs summing or averaging. Also note that with limited involvement of the client side, more complex computations (e.g., linear regression) can also be achieved [231].

How to bring cryptographically guaranteed sharing to the IoT ecosystem? The current PHE approaches are either targeted at single-key encrypted data [184, 231, 213] (no support for sharing) or consider only text-based data [185, 105] (of limited use in an IoT context). Talos [213, 206] is specifically tailored for IoT scenarios and has demonstrated PHE on embedded devices, but it does not offer any sharing features. Existing protocols for sharing, such as OAuth [153], fall short in providing strong assurances about the policy enforcements. Crypto-based sharing approaches [234], on the other hand, support no query processing over encrypted data.

In short, existing solutions either support encrypted query processing or secure sharing but not both. Moreover, due to their heavy computational overhead, PHE schemes have been considered unsuitable for low-power mobile and IoT devices. In this work, we tackle the challenges of cryptographically-protected and efficient sharing of PHE data, as illustrated in Figure 3.1. Our system is the first to combine encrypted sharing with encrypted query processing. Furthermore, our system is tailored towards mobile platforms, improving the state-of-the-art performance by more than one order of magnitude.

Approach. We introduce Pilatus, which extends Talos [213] with sharing capabilities. We enable efficient sharing of PHE data based on a re-encryption scheme [13]. This means that data is (PHE) encrypted at the client (IoT device/gateway) and uploaded to the cloud. When data owner Alice intends to share her data with Bob, she computes a token that enables the cloud to re-encrypt her data for Bob (without decrypting it first). The same process is used for Alice to share her data with a group. With only public keys and tokens (no secret keys nor plaintext information), the cloud is able to perform query and sharing operations directly on ciphertexts. Users can decide between sharing their individual data points (necessary for complex analytics) or aggregated results, both in encrypted form.

Further, we design a key revocation mechanism that allows users to terminate their data sharing at any time. We also propose an in situ key-update at the cloud, such that old data becomes protected with the owner's new key, without trusting the cloud with any private keys.

Our solutions build on the Elliptic Curve (EC)-based partially homomorphic encryption. Hence, a major challenge is to minimize

the decryption time of our EC-based cryptosystems, for both sharing and encrypted processing. We address the performance optimization with the Chinese Remainder Theorem, which enables smaller, faster, and parallel computations. We keep the induced overheads low enough to preserve the user experience (i.e., below 1 s response time [174], including network latency) such that users can interact with their remotely stored data seamlessly.

Contributions. In summary, this chapter makes the following contributions:

- We introduce a practical construction for sharing of PHE IoT data, with a sharing revocation mechanism that also allows in situ re-keying of the ciphertexts in the cloud;
- We improve the underlying cryptosystems' decryption time by one order of magnitude compared to state-of-the-art realizations such as Talos [213] and CryptDB [184];
- We design and implement Pilatus, a system that extends Talos with the above sharing schemes and with the necessary features for a practical sharing-enabled cloud storage;
- We assess the efficiency of Pilatus through end-to-end and micro benchmarks, on cloud services, on mobile devices, and to a lesser extent on low-power sensor devices. We also present two case study apps running Pilatus: the Fitbit fitness tracker and the Ava fertility tracker [14]. We make our prototype implementation of Pilatus publicly available[1].

This chapter is based on the contributions made in [210, 211].

3.2 Threat Model

We focus on IoT applications where data from wearables/smartphones are stored on third party clouds. We consider the following parties: the cloud (consisting of a front-end server and the database), clients (apps), and an Identity Provider (IDP) to certify the public key of each user.

Threats. Pilatus considers the cloud service to be honest-but-curious, such that it will follow the protocol correctly, but tries to learn as much as possible from the stored data. This is a valid model, as protocol violations, once detected, could penalize the service provider. At the same time,

[1] Pilatus is available at https://github.com/Talos-crypto/Pilatus

the adversary might be eager to learn more about user data without being noticed (i.e., passive). External adversaries can also gain access to the encrypted data as a consequence of system compromise. In summary, we consider cloud-side threats which are due to system compromise (e.g., data theft), financial incentives (e.g., unauthorized trades with users data), or malicious insiders (e.g., curious admins).

Assumptions. In addition to the honest-but-curious cloud assumption, Pilatus assumes that the IDP correctly verifies users' identity-key pairs. The IDP is a standard requirement in multi-user systems and can be a known and trustworthy external entity or an internal unit. Group members of shared data are semi-trusted, in that they do not collude with the cloud to leak the group data or key. This is a reasonable assumption for small groups where members are acquainted with each other. Moreover, Pilatus assumes that the applications behave correctly and do not hand out user keys to malicious parties. Finally, we assume state-of-the-art security mechanisms to be in place for device security [149, 138, 150] and that all parties communicate over secure channels.

Guarantees. Pilatus protects the confidentiality of users data stored in the cloud in the presence of passive adversaries (e.g., compromised servers). In case the user device is compromised and group keys are disclosed, only data of the compromised user and the affected group are disclosed. Pilatus cryptographically prevents unauthorized data access. Pilatus provides user authentication mechanisms but does not guarantee freshness or correctness of the retrieved data. In order to provide such guarantees, one could complement our system with integrity protection frameworks such as Verena [129]. Pilatus does not hide access patterns, which can potentially reveal sensitive information [124].

3.3 Pilatus Overview

We briefly introduce the requirements of the applications we target and then present the architecture of Pilatus.

3.3.1 Applications

Pilatus focuses on applications collecting sensitive data that require processing and sharing. Examples of such applications include fitness or health trackers. These applications store private data in the cloud that can reveal privacy-sensitive information, e.g., illness, lifestyle, or location. For instance, Fitbit wristbands collect a user's heart rate, step counts, and location data. Similarly, certain health tracking applications

and wearables, such as Ava [14], allow women to track their menstrual cycle, predict (in)fertile phases, and detect potential health issues. Ava bracelets rely mainly on body temperature and various other sensors (e.g., heart rate variability, perfusion, breathing rate, bioimpedance). Though insightful, logging such private information raises serious privacy concerns.

Secure and transparent sharing plays an essential role in bringing such fitness- and health-related apps to their full potential. When users are willing to share data with experts (e.g., medical practitioners), analytical services, or just casually with friends, they must be in full control over who can access and what can be accessed. Moreover, there is a need for query processing capabilities directly in the cloud, since downloading the entire data volume for on-device processing becomes impractical as the data grows. For instance, in Fitbit and Ava, the mobile apps typically display total sums or averaged values (e.g., heart rate) over a given period of time.

To satisfy the above requirements, Pilatus facilitates storing encrypted IoT data in the cloud, while enabling processing and cryptographically-protected sharing of encrypted data. In Section 3.7.1, we show that our Pilatus-based app prototypes Ava and Fitbit induce only a modest overhead (i.e., a maximum of 130 ms) while interacting with encrypted data. Note that while the focus of this work is on health and fitness applications, our system design and application discussions apply as well to other IoT applications (smart homes, connected cars, etc.).

3.3.2 Architecture

Pilatus, as an extension of Talos, uses Partially Homomorphic Encryption (PHE) schemes to enable query processing over encrypted data in the cloud. It offers a new scheme for secure sharing of PHE data among users and groups, based on re-encryption techniques. Our sharing feature includes access revocation with in situ re-keying. We optimize the performance of the underlying cryptographic schemes by one order of magnitude to make them practical on mobile devices. Note that Pilatus inherits the order-preserving encryption from Talos for range queries. Together with PHE, this enables querying a sum/average over a range of timestamps or any other relevant combination of metadata[2].

Pilatus consists of three main components: the client engine, the cloud engine, and an identity provider, as depicted in Figure 3.2.

Client Engine. The client engine runs on the user side and is the only

[2]We apply range queries to data types that are of high entropy from a sparse domain to avoid any data leakage due to inference attacks [171].

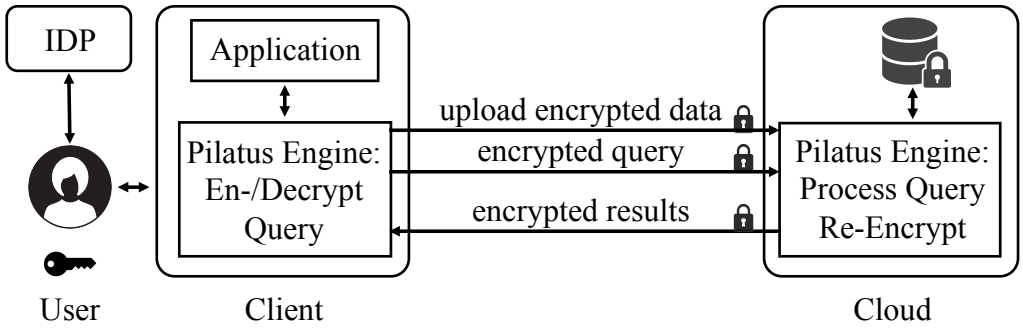


Figure 3.2: Pilatus architecture. The client engine performs en-/decryptions, such that the cloud only stores encrypted data. The cloud engine performs encrypted query processing and necessary re-encryptions for data sharing.

component with access to keying material. It is primarily designed for users' personal mobile platforms, such as smartphones. It interacts with the IDP to verify the ID-to-key bindings and with the cloud engine for secure storage, sharing, and retrieving of data. More specifically, it encrypts, decrypts, handles the keys, and sharing-related activities such as joining new groups, issuing delegated access rights (i.e., re-encryption tokens), triggering revocation, and re-keying. For constrained IoT devices (sensors), we have a stripped-down client engine with limited functionality, such as encryption to store PHE encrypted data in the cloud.

Cloud Engine. The cloud engine is application-agnostic and provides the basic database interface and features. It stores data and processes queries from the client engine. The cloud engine has only access to data in encrypted form. It supports the algorithms required for processing encrypted data, i.e., homomorphic addition, re-encryption, and in-situ re-keying. Our design is currently targeted for structural databases (i.e., MySQL) and uses User-Defined Functions to replace the default routines with crypto-enabled ones (see Section 4.4 for details).

Identity Provider. The IDP is an independent party responsible for verifying the user identity to public key bindings. The IDP is used by the client engine to search for the public key of another user or group. Pilatus is independent of the IDP and outsources this role to systems such as Keybase [133], that provide provable identity-key bindings, by utilizing prevalent social media channels and online accounts (i.e., users prove their identity by posting an individual token on Twitter, Facebook, Github, etc.).

3.4 Cryptographic Background

We present here the necessary cryptographic background to help understand the Pilatus design.

3.4.1 Partial Homomorphic Encryption

Partial Homomorphic Encryption (PHE) schemes allow the computation of certain mathematical operations over encrypted data. For instance, additive homomorphic schemes, such as the Paillier cryptosystem [180], support the addition of ciphertexts, such that the result is equal to the addition of the plaintext values (i.e., $ENC(m_1)\ o\ ENC(m_2) = ENC(m_1+m_2)$).

The Elliptic Curve (EC) version of the ElGamal cryptosystem is an alternative additive homomorphic scheme, used in Talos and Pilatus. EC-ElGamal's security is based on the EC Discrete Logarithm Problem (ECDLP) [222]. It provides semantic security (i.e., IND-CPA under the assumption of decisional Diffie-Hellman). A challenge in making practical use of EC-ElGamal is that it operates over EC points rather than arbitrary messages. Hence, one needs a scheme that maps an integer to an EC point (and back), while preserving the homomorphic property of EC-ElGamal.

Talos, which focuses on IoT data, uses a theoretical method [222] that becomes practical for small integer data, e.g., 32-bit (frequent in IoT scenarios, to represent an integer or fixed point number) [213]. The process is as follows: to map an integer m to an EC point M, m is multiplied by a publicly known point G on the curve: $M = mG$. After decryption, M must, however, be mapped back to m. This requires solving an ECDLP. Although this is computationally infeasible for large numbers, solving it for smaller than 32-bit integers can be realized in a reasonable time with, e.g., the Baby-Step-Giant-Step (BSGS) algorithm (this is equivalent to breaking 32-bit security). BSGS is based on a memory-computing tradeoff and requires an efficient lookup table. Note that this mapping procedure, as depicted in Figure 3.3, does not affect the overall security: the ECDLP is solved to obtain m from M, but M itself is protected with strong cryptography, in this case, 80-bit or 128-bit security.

Pilatus inherits this scheme from Talos, but presents additional optimizations to speed up the process by one order of magnitude, as discussed in Section 3.5.2.

3.4.2 Re-Encryption

Re-Encryption (RE) enables a proxy to convert ciphertexts under Alice's key to ciphertexts under Bob's key, without disclosing the plaintext.

Figure 3.3: Plaintext to EC point mapping before encryption and after decryption.

Hence Alice can share data with Bob, without sharing her private key nor performing any encryption for Bob on her personal device.

The AFGH [13] RE scheme relies on pairing-based cryptography. AFGH defines next to the standard functions *key generation*, *encryption*, and *decryption*, two additional functions: *re-encryption-token generation* and *re-encryption*. The former is used by Alice to generate the re-encryption token for Bob, based on her own private key and Bob's public key. The latter performs the re-encryption from Alice to Bob given the ciphertext is encrypted under Alice's public key.

At a higher abstraction, pairings (or bilinear maps) establish a relationship between two cryptographic groups. In AFGH, re-encryption consists in transforming a ciphertext from the first group to the second group. The underlying pairing in AFGH is essentially a bilinear map [37] which, given a cyclic group $\mathbb{G}$ of prime order q, has the following property for $a, b \in \mathbb{Z}q$ and $g, h \in \mathbb{G}$: $e(g^a, h^b) = e(g, h)^{ab}$. Such maps can be realized with the Weil and Tate pairings, which are efficiently computable with Miller's algorithm [165]. However, designing efficient pairings is an ongoing research topic [10].

More formally, AFGH defines the system parameters as $(g, e, Z, \mathbb{G}, \mathbb{G}_t)$, where $g \in \mathbb{G}$, $Z = e(g, g) \in \mathbb{G}_t$ and e as the map: $\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_t$. $\mathbb{G}$ and $\mathbb{G}_t$ are both cyclic groups of the same prime order. The user Alice computes her public key as $pk_a = g^a$ and her private key as $sk_a = a$. Alice can issue Bob the re-encryption token based on his public key pk_b as the $Token_{a \rightarrow b} = pk_b^{1/a} = g^{b/a} \in \mathbb{G}$. The encryption of m is performed as:

$$C_a = (mZ^r, g^{ar}) \tag{3.1}$$

where r is a random number. Note that the ciphertext C_a is composed of two components, similar to the EC-ElGamal ciphertext. A proxy with access to $Token_{a \rightarrow b}$ performs the re-encryption by transforming the second component of C_a:

$$C_b = (mZ^r, Z^{br}), \text{ with } Z^{br} = e(g^{ar}, g^{b/a}) \tag{3.2}$$

Bob can now decrypt the ciphertext C_b with his private secret $sk_b = b$ and the pairing Z as:

$$m = \frac{mZ^r}{(Z^{br})^{1/b}} \tag{3.3}$$

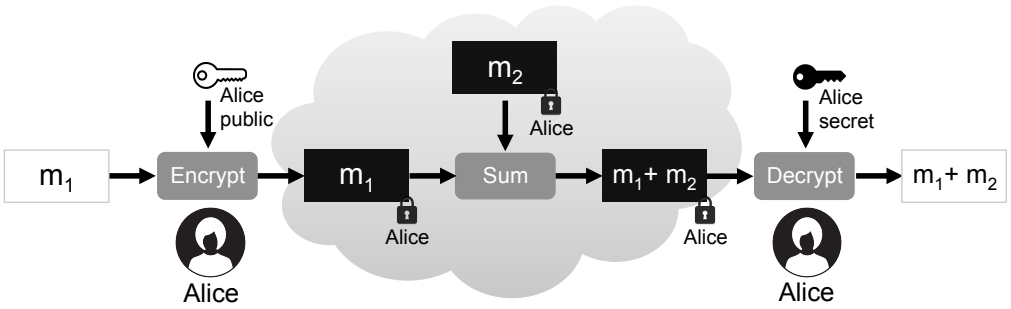


Figure 3.4: Encrypted query processing. The data is encrypted with Alice's public key. Computations take place in the cloud, on ciphertexts. The result is decrypted with Alice's private key.

In Section 3.5.1.2, we show how to employ AFGH in an efficient additive homomorphic context. The cloud serves as the proxy, in charge of re-encrypting data of a user to another user or group.

3.5 Pilatus Design

This section presents our EC-based encryption for sharing of PHE data and discusses aspects such as performance optimization, revocation, and authorization.

3.5.1 Processing and Sharing Encrypted Data

We present Pilatus's two modes of operation: Standard Mode and Data Sharing. The former covers the single-key case, while the latter enables cryptographically-protected sharing. Both modes exhibit additive homomorphic properties, as illustrated in Figure 3.4.

3.5.1.1 Standard Mode

When uploading data that is not intended for sharing, the Pilatus client engine selects the standard mode. The standard mode is mostly inherited from Talos and uses EC-Elgamal as an additive homomorphic encryption scheme (Section 3.4.1). However, the decryption in Talos suffers from an exponential increase of computational costs for larger integer values, as shown in Figure 3.6. In Section 3.5.2, we introduce our optimizations to overcome this shortcoming and accelerate decryption and enable the use of integers larger than 32-bit as plaintext.

3.5.1.2 Data Sharing

In Pilatus, we construct an efficient additive homomorphic interpretation of AFGH (presented in Section 3.4.2). This enables cryptographically-protected (as opposed to policy-based) sharing of PHE data, without the need to disclose any private keys to the cloud. All the cloud needs is a token generated by the owner, from the owner's private key and the target user/group's public key, as illustrated in Figure 3.5.

To realize the homomorphic additive property, we use the algebraic structure of elliptic curves over finite fields, similar to [233]. Note that bilinear-map-based cryptosystems, such as AFGH, leave the selection of the underlying pairing-friendly elliptic curve to implementation. In the recent years, research on optimal paring curves [76] has further progressed. In Pilatus, we rely on the optimal Ate pairing [232][3].

To enable the additive homomorphic property, we represent message m as $M = Z^m$, with Z as the pairing. Given the public key $pk_a = g^a$ of Alice with g as a generator point in $\mathbb{G}$ and the random r, we encrypt as:

$$C_a = (Z^m Z^r, pk_a^r) = (Z^{m+r}, g^{ar}) \tag{3.4}$$

and re-encrypt for Bob as:

$$C_b = (Z^{m+r}, Z^{br}), \text{ with } Z^{br} = e(g^{ar}, g^{b/a}) \tag{3.5}$$

With access to the private key b, Bob can decrypt as:

$$M = \frac{Z^{m+r}}{(Z^{br})^{1/b}} = \frac{Z^{m+r}}{Z^r} = Z^m \tag{3.6}$$

Note that in the final step of decryption, we still need to map back M to m (i.e., solving a discrete log problem), which similar to the standard mode benefits from our performance optimization, presented in the next section.

The homomorphic addition of two ciphertexts C_{b1} and C_{b2} (encrypted under Bob's key) is performed as follows:

$$\begin{aligned} C_{b1} + C_{b2} &= (Z^{m1+r1}, Z^{br1}) \odot (Z^{m2+r2}, Z^{br2}) \\ &= (Z^{m1+r1} Z^{m2+r2}, Z^{br1} Z^{br2}) = (Z^{m1+r1+m2+r2}, Z^{br1+br2}) \\ &= (Z^{m1+m2+r1+r2}, Z^{b(r1+r2)}) \end{aligned} \tag{3.7}$$

Because AFGH is key-optimal, Pilatus's storage size for a user remains constant regardless of the number of users/groups it shares the data to. Moreover, the re-encryption tokens are unidirectional and non-transitive. This implies, given $Token_{a \to b}$ it is only possible to re-encrypt

[3]The optimal Ate pairing is over Barreto-Naehrigopera curves [19, 10], which are pairing-friendly elliptic curves of prime order, with embedding degree 12.

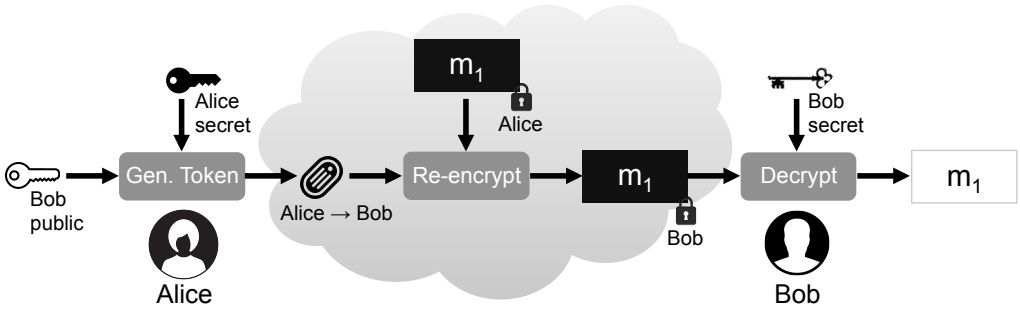


Figure 3.5: Data sharing (re-encryption). Alice generates a token, from her private key and Bob's public key. The cloud uses the token to re-encrypt Alice's data as Bob's data. Bob can in turn decrypt with his private key. The same mechanism is used for group sharing. Note that re-encryption is non-transitive, i.e., the result can not be re-encrypted again.

Alice's ciphertexts C_a to Bob's ciphertexts C_b. The opposite direction is cryptographically not feasible. Additionally, given $Token_{a\to b}$ and $Token_{b\to m}$ it is cryptographically not feasible to transform Alice's ciphertext to Mallory's ciphertext.

3.5.2 Performance Optimization

At decryption, the EC mapping proposed by Talos (see Section 3.4.1) requires solving an ECDLP problem to convert a plaintext EC point to the original plaintext integer. As discussed earlier, using the Baby-Step-Giant-Step (BSGS) algorithm we can solve the ECDLP for small integer values (i.e., ≤32-bit) within a few milliseconds. However, the performance of this algorithm decreases exponentially with larger integers, as depicted in Figure 3.6 (e.g., already 15 s for decryption of 38-bit integers). This is because with each additional bit the search space is doubled until it becomes too large to efficiently find the solution (i.e., computing the discrete logarithm).

This technique has two major shortcomings with regards to our design: *(i)* batch decryption can harm user experience, exceeding 1 s with as few as 25 decryptions (32-bit values); *(ii)* with larger numbers, e.g., 64-bit integers, this approach becomes impractical. This demands an optimization that also maintains the homomorphic property of these schemes.

Approach. Our optimization is based on combining the Chinese Remainder Theorem (CRT) [116] with the BSGS algorithm. It is applicable for decryption in both the standard and sharing modes. With CRT,

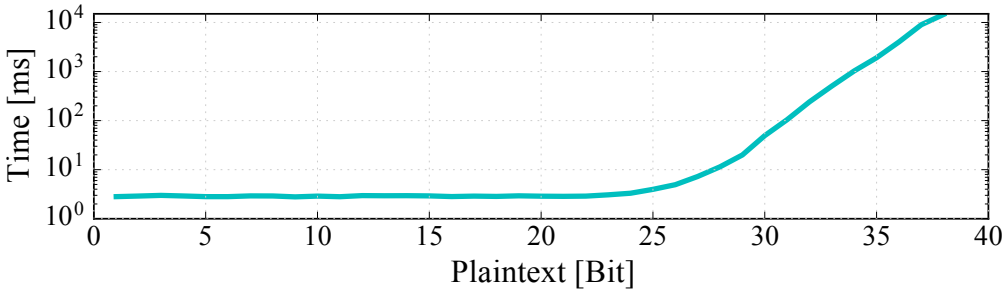


Figure 3.6: Decryption in EC-ElGamal with the Baby-Step-Giant-Step algorithm, 80-bit security and 4 threads on Nexus 5. Smaller plaintexts (23-bit) are decrypted efficiently (<3 ms), larger plaintexts cause an exponential slowdown.

we reduce solving one difficult ECDLP problem (i.e., a large integer) to solving several smaller ECDLP problems, for which the BSGS algorithm performs efficiently, as illustrated in Figure 3.7. As BSGS exhibits an exponential cost, our approach can provide drastic performance improvements.

CRT is used in many cryptographic constructions [58, 222] and Hu et al. [116] present a formal treatment on how to leverage CRT for homomorphic encryption schemes. We are the first to utilize the general CRT optimization in *combination* with the BSGS algorithm for an efficient computation of discrete logarithms in a pairing-based re-encryption.

The CRT technique is based on the simple idea of representing a number X uniquely by its remainders a_i from the following congruence equations, where n_i are co-primes (i.e., $\gcd(n_i, n_j) = 1, \forall\ i, j$):

$$X \cong a_i \ mod \ n_i \tag{3.8}$$

Hereby, N which is the product of all co-primes n_i (i.e., $N = \prod n_i$) should be larger than X. With regards to our encryption schemes (i.e., EC-ElGamal and pairing-based re-encryption), X corresponds to the plaintext value which can now be represented with the remainders a_i. Since the remainders are significantly smaller than X, the decryption is performed more efficiently. Given the co-primes n_i and the remainders a_i, X can be efficiently computed, as:

$$X = \sum_{i=1}^{r} a_i N_i y_i (mod\ N) \tag{3.9}$$

where $N_i = N/n_i$ and $y_i = N_i^{-1} mod\ n_i$. Note that y_i and N_i remain unchanged for a given set of co-prime values n_i and hence can be pre-computed in advance.

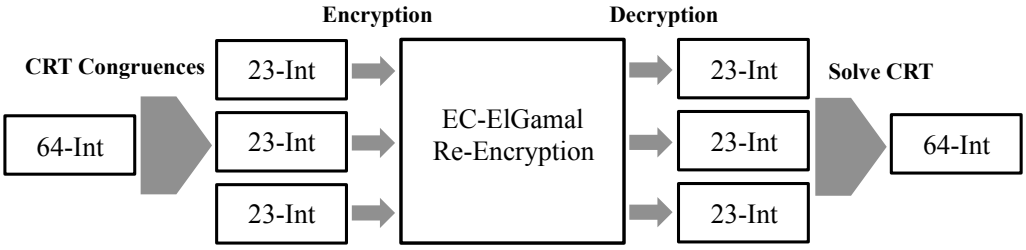


Figure 3.7: We optimize the decryption in both standard and sharing modes, with a technique based on the Chinese Remainder Theorem (CRT) where a larger value is represented by several smaller ones.

Realization. To utilize CRT to our benefit, we first compute the remainders a_1 and a_2 (assuming two congruences) for a given value X (i.e., as defined in Equation 3.8). Now instead of encrypting X, we encrypt a_1 and a_2. Note that the co-prime values n_1 and n_2 are public information.

Utilizing CRT has the side effect of increased encryption cost and ciphertext-size (linearly by the number of congruences). For instance, with 160-bit ECC (80-bit security), the ciphertext-size of EC-ElGamal increases for 32-bit values from 42 bytes (2 compressed EC-points) to 84 bytes (4 compressed EC-points), which is still lower than the ciphertext-size in the Paillier cryptosystem (256 bytes with 80-bit security).

The advantage of CRT becomes apparent while performing decryptions. Instead of solving ECDLP for a large X (which can take seconds to hours for larger values), we now solve it efficiently for the remainders (i.e., a_1, a_2). The larger the plaintext value, the higher the performance gain due to CRT (i.e., several orders of magnitude for large values). For instance, the 15 s decryption time for a 38-bit value is reduced to less than 10 ms for EC-ElGamal (see Section 3.7.0.1).

Homomorphism. We discuss here how we keep the additive homomorphic property with our CRT extension. To add two large integers X_a and X_b, we add their remainders (a_i and b_i respectively) in the corresponding congruences, as follows:

$$X_a + X_b = \sum_{i=1}^{r} (a_i + b_i) N_i y_i \ (mod\ N) \tag{3.10}$$

This is possible due to modular arithmetic ($X_a + X_b = a_i + b_i \ mod\ n_i$). Since our underlying encryption schemes are additive homomorphic, we can compute the addition of the corresponding encrypted remainders,

as follows:

$$\begin{aligned} ENC(X_1) + ENC(X_2) = \\ (ENC(a_1), ENC(a_2)) + (ENC(b_1), ENC(b_2)) = \\ ENC(a_1 + b_1), ENC(a_2 + b_2) \end{aligned} \tag{3.11}$$

Hence, EC-ElGamal and our pairing-based re-encryption remain additive homomorphic.

Note while configuring the CRT in EC-ElGamal, we make a space-computation trade-off, such that we access a larger ciphertext size due to the remainders in favor of efficient decryption. One essential configuration parameter to consider is the number of homomorphic additions to be expected. In the plaintext domain, a 32-bit integer addition can result in overflow. However, in the ciphertext domain, the homomorphically added final value can grow over 32-bits, rending the decryption very expensive. The lack of module or overflow in the ciphertext domain is a limitation. With CRT, the remainders represent a 32-bit integer value within two modulo 17-bit prime values. The encrypted congruences can similarly grow with homomorphic additions to exceed the corresponding prime modulus. For instance, with a high number of homomorphic additions, an encrypted remainder can grow significantly larger than 17-bit, becoming too expensive to decrypt (see Figure 3.6). One possible way to support a higher number of homomorphic additions is to select more congruences. For instance for 32-bit integers, instead of the default two 17-bit congruences, we could use four 9-bit congruences, leaving enough space for the individual congruences to grow with each homomorphic addition.

3.5.3 Key Revocation

To authorize data sharing, a data owner issues a cryptographic token, used by the cloud to re-encrypt users data towards the destination user. We address here the challenge of terminating such a data sharing, cryptographically.

Key Update. In Pilatus, when users decide to revoke a data sharing, they simply begin using new keys for new data. This renders previously issued tokens obsolete and prevents new ciphertexts cryptographically from being re-encrypted with the old token. Once new keys are in place, valid sharing relationships are updated with new tokens such that the sharing flow can be maintained. We discuss in Section 3.5.4 in more details how the data sharing authorization works with regards to joining and leaving groups. Note that a key revocation event can as well occur, when the encryption key of the user is compromised.

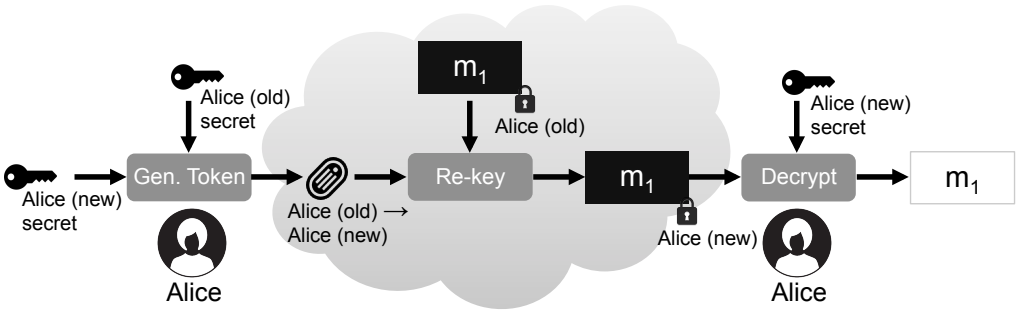


Figure 3.8: Data revocation (in situ re-keying). Alice builds a token from both her old and new private keys (none of which is leaked). The cloud uses the token to re-key Alice's data, such that they can now be decrypted with the new key. Alice can re-key multiple times and apply re-encryption to re-keyed data.

We consider two cases for the key update: malicious cloud and semi-honest cloud. In the former case, we leave old data protected with old keys. We consider such old data to be in the wild, since already shared and possibly cached at sharing parties. However, in the latter case, it is desirable to update the encryption keys of the old data to the latest keys, for consistent access. Hence, it is important to devise a secure solution that allows the semi-honest cloud to perform the re-keying without access to any private key.

In Situ Re-keying. To construct our re-keying mechanism, we leverage the fact that user Alice has access to both old and new private keys (a and a', respectively), neither of which is disclosed to any party[4]. Hence, at the time of re-keying, the private keys of Alice are not compromised.

The re-keying (see Figure 3.8) is carried out on data, at the cloud, prior to sharing (i.e., ciphertexts in level-1, as represented in Equation 3.4). To this end, Alice issues a key-update token $\delta = a'/a$ and the cloud performs the re-keying as follows (only the second component of the ciphertext is adjusted):

$$C_{a'} = (C_1, \delta C_2) = (C_1, ra'G) \tag{3.12}$$

After re-keying, all ciphertexts on the cloud are encrypted with the latest key a'. Note that re-keying, unlike re-encryption, is transitive, i.e., the re-keying can be applied multiple times to the same item. The scheme would, however, be poorly suited for sharing, as it requires both the source and target private keys.

Note that since both old and new keys are only known to Alice,

[4]Note that in contrast to key homomorphic PRFs (i.e., Pseudo Random Functions) [234], where a symmetric key is shared between parties, our re-keying scheme is key-private.

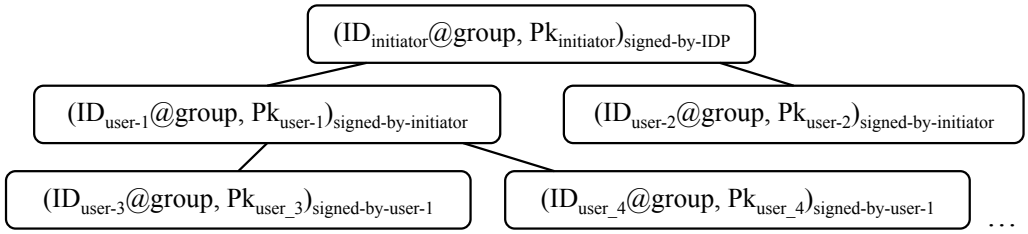


Figure 3.9: Access graph for group membership.

a curious cloud cannot learn anything about the private keys from δ. However, a malicious cloud could use the reverse of the key-update token (i.e., a/a') and downgrade ciphertexts encrypted under newest keys to old keys. This is why we only enable re-keying in case a semi-honest cloud or a trusted proxy is present.

3.5.4 Group Sharing Authorization

Data in Pilatus can be shared either directly with a user or a group. We describe a simple sharing authorization mechanism for group-related operations. The construction of the authorization is crucial as it ensures that a joining group member *(i)* issues the re-encryption token for the authentic group; and *(ii)* retrieves the correct group key. These two aspects are related, since the correct group key is necessary for the process of generating the re-encryption token.

Access Graphs. In our construction, we leverage access graphs, which are similar to certification paths in the public key infrastructure, in combination with our re-encryption scheme. We form an access graph for each group. The root node holds the initiator's ID concatenated with the group identity (e.g., "runners") and the public key of the initiator, as depicted in Figure 3.9. The joining members of the group compose the access graph nodes. Each node, except for the root, is signed by the immediate parent node. The root is signed by the IDP. The access graph allows group members to vouch for the membership of other members.

Joining. Group membership is authorized by a group member in two steps: *(i)* signing the extended identity (i.e., ID@group name) and the public key of the new member, e.g., Alice, (after verifying the key correctness over the IDP); and *(ii)* issuing a re-encryption $Token_{g \to a}$ which is then stored in the cloud. After joining the group, Alice provides the re-encryption $Token_{a \to g}$, required for sharing with the group. To issue this re-encryption token and later be able to access (i.e., decrypt) group data, Alice needs the public (PK_g) and private (SK_g) keys of the target group.

This information is stored signed and partly encrypted on the cloud:

$$(ID_g, PK_g, ENC_g(SK_g))_{signed-by-initiator}$$

Note that the initiator's signature on the key information prevents a malicious entity in deceiving Alice into issuing a token for a fake group with the same name. Currently, group members are authorized to add new members. We can restrict this authority to the initiator only by encrypting the group's private key with the initiator's key. A new member would require the initiator's authorization which is expressed in the $Token_{initiator \rightarrow a}$ to access the group key.

Leaving. To leave a group, Alice initiates our revocation procedure (c.f. Section 3.5.3). After revocation, new coming data is encrypted with new keys and can no longer be transformed with the expired $Token_{a \rightarrow g}$. Her previous data, however, remains in the wild and can still be accessed by the group members, unless Alice decides to trigger our in-situ re-keying feature. Note that disclosure of the group's secret key SK_g and Alice's $Token_{a \rightarrow g}$ does not expose Alice's private key.

3.5.5 Security Analysis

Our main security goals are to defy passive attacks targeted at data on the cloud as well as to prevent access of unauthorized users (see Section 3.2). Pilatus achieves these goals such that data on the cloud remains strongly encrypted (i.e., semantic security) at all times. The cloud never gains access to any decryption keys. We rely on the IDP to prevent fake user creations.

To protect the data from unauthorized access, we cryptographically restrict data access to users with decryption keys (i.e., either individual or group keys). With the re-encryption token, the cloud can only re-encrypt the stored data towards the authorized group/service. Moreover, we prevent a malicious cloud from performing unauthorized re-encryptions towards a malicious user (thanks to the one-hop property of the re-encryption scheme).

The disclosure of group keys does not affect the security of the corresponding private keys of the group members. This is because our underlying re-encryption scheme is key-private. After such an incident, members can perform a revocation to terminate the data transformation into group data. Our in situ key-update technique assumes a semi-honest cloud. In other cases, re-keying can either be disabled or delegated to a trusted proxy.

Cryptosystem	Setup time [ms]	
	80-bit sec.	128-bit sec.
Paillier	1623	42190
Standard	0.28	0.61
Sharing: Key setup	4.15	6.72
Sharing: Token gen.	2.5	4.7

Table 3.1: Average setup time on Nexus 5 for 80-bit and 128-bit security levels.

3.6 Implementation

We implemented a prototype of Pilatus for mobile platforms (user/client devices) and the cloud (structured data storage). The client engine (Android) consists of a REST client for interactions with the cloud. Application developers can use the Pilatus API which internally calls their previously defined SQL procedures. The client engine handles data encryption before performing requests and decryption after retrieving the data from an API call. The client engine parses the query (i.e., JSQL parser), checks if the operations are valid, encrypts the query, and sends it to the cloud.

For the EC-ElGamal encryption, we utilize the ECC module of the OpenSSL library (v1.1.0). We implemented the data sharing components (i.e., re-encryption) based on the RELIC toolkit [9, 10]. We support 80-bit and 128-bit security levels. Our BSGS algorithm implementation relies on hash-map (i.e., klib library) for the look-up table.

The cloud engine supports a MySQL database, for which we implemented the corresponding User Defined Functions (UDF). We use UDFs to replace the default routines with crypto-enabled ones, without the need of recompiling the database. Incoming queries indicate the UDF to be used, e.g., `SELECT SUM_EC_ELGAMAL(column-x) FROM table-y`, where the standard `SUM` is replaced with the dedicated homomorphic addition sum for EC-ElGamal. Moreover, the cloud engine is equipped with a REST engine (i.e., Restlet library).

The implementation of Pilatus consists of 2000 sloc of C/C++, 10000 sloc of Java, and another 4000 sloc for testing, setup scripts, and benchmarking. Our prototype Android applications Fitbit and Ava consist of 2400 and 2500 sloc, respectively.

Example Applications. To show the feasibility of Pilatus and evaluate its end-to-end performance, we developed two example mobile applications that integrate Pilatus, where the cloud components are hosted on

Mode	Security	Smartphone			Constrained IoT	Cloud		
		ENC [ms]	DEC-1 [ms]	DEC-2 [ms]	ENC [ms]	ADD-1 [μs]	ADD-2 [μs]	Share [ms]
Standard	80	2.4	2	-	252	54	-	-
	128	4.7	3.9	-	530	91.8	-	-
Sharing	80	9.8	11.9	9	not available	62	30.6	1.7
	128	15.2	17.4	13.3	not available	75	73.7	2.3

Table 3.2: Complete overview of our evaluation results for 32-bit integers (i.e., 2 CRTs) with 80-bit security. DEC-1/2 and ADD-1/2 refer to operations (1) before and (2) after sharing. Note that on smartphones, batch encryption/decryption with multithreading (i.e., 4 threads) yields us a 3× performance improvement.

Amazon's cloud services. Our Android activity tracking app operates on data collected by our personal Fitbit device. It fetches the data from Fitbit servers and stores it encrypted in our cloud instance. For our Ava fertility tracking app, we received anonymized data from the Avawomen startup [14]. In both apps, the users can interact with the data similar to the original apps. Our applications do not cache any data locally which allows us to study the worst-case performance while interacting with remote data. The cloud and the client communicate over HTTPS and data is encoded in JSON format, as a compact data representation form. For authentication, we rely on the OpenID Connect [153], where we currently support Google accounts [94] as a proof-of-concept.

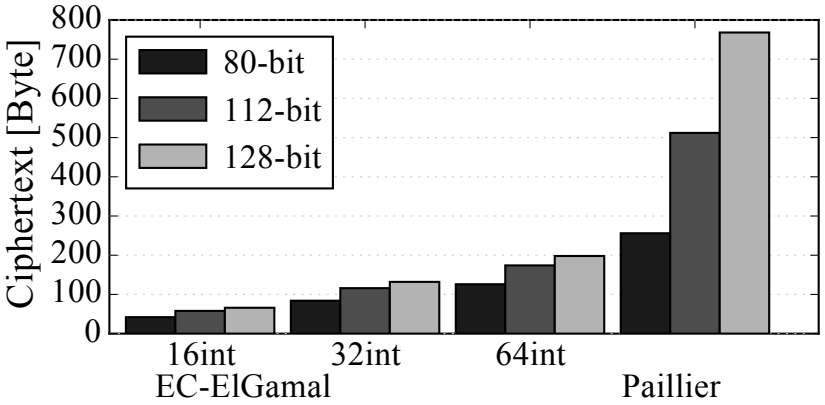
Constrained IoT Devices. We implemented a prototype of Pilatus for more constrained IoT devices, where the client-engine only accommodates the encryption logic in the standard mode. We based our implementation on Contiki [61], an open-source low-power operating system for IoT devices. For cryptographic processing, we utilize both software libraries (i.e., RELIC toolkit [9]) and the hardware crypto accelerator.

3.7 Evaluation

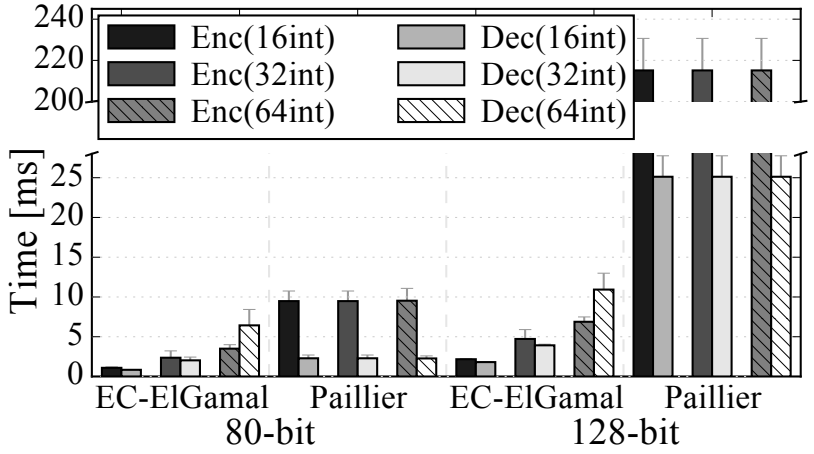
This section presents a thorough evaluation of Pilatus, both on the cloud and on client sides (mobile device and, to a lesser extent, constrained IoT device).

Evaluation Setup. Our evaluation setup consists of the client engine running on a smartphone and the cloud engine running at Amazon cloud services (AWS). We use an LG Nexus 5, equipped with a 2.3 GHz quad-core 64-bit processor and 2 GB RAM, running Android 5.1.1. Our AWS account provides 25 GB of storage and one instance of Intel Xeon 3.3 GHz CPUs with 1 GB RAM. For micro-benchmarks of homomorphic addition, we additionally use a MacBook Pro equipped with 2.2 GHz Intel Core i7 and 8 GB of RAM.

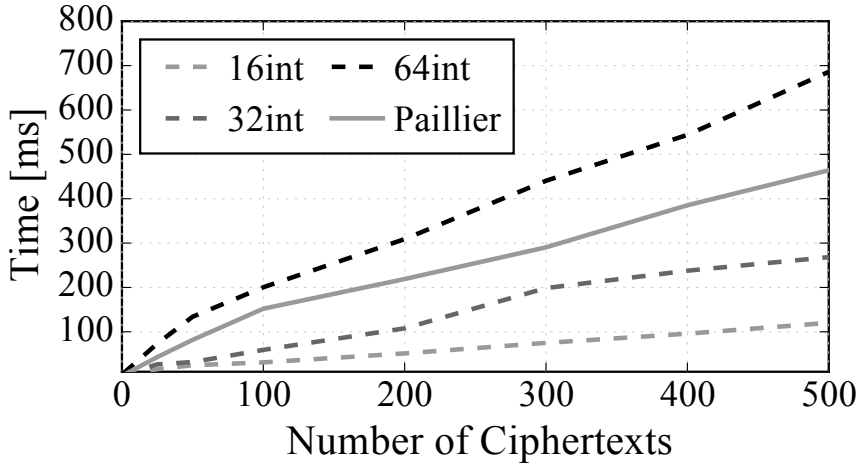
For the client engine, we also present results on constrained IoT devices with our Contiki implementation. We select OpenMotes as the hardware platform, which utilize the same class of MCU as popular activity trackers such as Fitbit. OpenMotes are based on the TI CC2538 microcontroller [229], i.e., 32-bit ARM Cortex-M3 SoC at 32 MHz, with a public-key crypto accelerator running up to 250 MHz. They are equipped with IEEE 802.15.4-compliant RF transceivers, 32 kB of RAM and 512 kB of ROM.



(a) Ciphertext size (bandwidth/memory).



(b) Paillier vs. optimized EC-ElGamal, 80 and 128-bit security levels.



(c) Batch decryptions, 80-bit security and 4x threads on Nexus 5.

Figure 3.10: On-device Standard Mode evaluation. Compares Paillier (utilized in CryptDB) with our optimized EC-ElGamal. With the CRT technique, our standard mode can differentiate between different input lengths.

Metrics. We rely on the following metrics to report on the overheads and performance of Pilatus:

- **Computation time** indicates the CPU time required to perform a certain operation. The computation time has a direct impact on the application delay and energy consumption of mobile platforms.
- **Ciphertext size** is an important metric considering network bandwidth. It measures the impact of Pilatus on the communication requirements.
- **System throughput** is the rate of operations on encrypted data performed in the cloud.
- **Application latency** reflects the time from a client initiating a query to the client receiving and decrypting the results. It accounts for query processing at the client side, plus network latency and cloud processing time.

In the next sections, we continue with discussing the results of the system components benchmark and then elaborate on the end-to-end evaluation results. Table 3.2 summarizes the evaluation results.

3.7.0.1 Standard Mode

In the standard mode, we utilize EC-ElGamal to process data encrypted under a single key (no data sharing). While the focus of our benchmark is on EC-ElGamal's performance, we also compare it to the more conventional Paillier cryptosystem, used for instance in CryptDB [184].

Key Generation. Table 3.1 shows the average setup time on the Nexus 5 device. EC-ElGamal requires less than 0.3 ms to generate keying material with 80-bit security. This is significantly lower than the 1623 ms required for Paillier, even though our key generation additionally includes finding the corresponding primes for the congruences in the CRT. Anyhow, key generation does not frequently occur, compared to encryption and decryption, detailed next.

Ciphertext Size. Figure 3.10(a) shows the ciphertext sizes for different integer sizes and security levels. In the standard mode, we support 16, 32 and 64-bit integers. Each integer size requires a different number of congruences, leading to a ciphertext size between 42 and 126 bytes in the 80-bit security case. In contrast, Paillier requires 256 bytes, regardless of the integer size. The difference is even more pronounced as the security level increases, negatively impacting network bandwidth and cloud storage.

Integer size [bits]	Encryption time [ms]					Decryption time [ms]					Ciphertext size [bytes]				
	Talos	Pilatus, #congruences				Talos	Pilatus, #congruences				Talos	Pilatus, #congruences			
		2	3	4	5		2	3	4	5		2	3	4	5
32	1.1	**2.4**	3.3	4.4	5.4	42*	**2.0**	2.9	4.0	5.0	42	84	126	164	210
64	1.2	2.5	**3.5**	4.7	5.7	infeasible	139	**6.4**	4.2	5.0					

Table 3.3: CRT optimization. Performance of Pilatus (EC-ElGamal with CRT) on Nexus 5 with 80-bit security, compared to Talos (i.e., no CRT). *The reported 190 ms in Talos [213] is reduced here to 42 ms with multithreading.

Encryption/Decryption. Figure 3.10(b) shows the encryption and decryption time with EC-ElGamal vs. Pailler, for different integer sizes, with both 80-bit and 128-bit security. Paillier has constant computation times due to its large padding, while EC-ElGamal sees its performance increase for smaller plaintext. EC-ElGamal outperforms Paillier for encryption with a factor of 3 and more. For decryption, EC-ElGamal is the fastest in all settings but the case of 80-bit security and 64-bit integers. Paillier's sharp decrease in performance with 128-bit security is due to larger key sizes (from 1024 to 3072-bit) and the resulting big number operations.

Figure 3.10(c) shows the batch decryption time at the mobile device. Even for 64-bit integers, hundreds of items can be decrypted in some hundred milliseconds. This is an important factor for the responsiveness of smartphone applications.

Note that the good performance of EC-ElGamal is to a large part due to our CRT optimization, as discussed in Section 3.5.2. Moreover, the benefits of an efficient encryption are particularly important in an IoT context, since IoT applications tend to encrypt more data items than they decrypt (all measurements are stored in the cloud, only a subset or aggregates are accessed for display).

Homomorphic Addition. We measure the homomorphic addition in isolation on a MacBook Pro, thus exclude the bootstrapping overhead of the database's UDF. The homomorphic addition in the standard mode requires between 27 and 82 μs for 16 and 64-bit integers, respectively (see Table 3.2). This is higher than Paillier (8 μs), which is mainly due to the underlying structure of EC-ElGamal where the ciphertext consists of two EC points. Note that parallelizing the homomorphic addition on the cloud would potentially result in considerable performance gain. In Section 3.7.1, we discuss the impact of this overhead on `sum` queries.

3.7.0.2 Data Sharing Mode

We evaluate the data sharing mode, where the client encrypts data and issues a token such that the cloud can re-encrypt a ciphertext to the target user or group. The key setup is with 4 ms similarly efficient to EC-ElGamal (see Table 3.1).

Ciphertext Size. Note that we use the same number of congruences as in standard mode. Since our re-encryption scheme is based on bilinear maps, we have to select the parameters such that we achieve at least 80-bit (i.e., subgroup size 160 and extension field size 1024) or 128-bit security. This results in larger ciphertext sizes compared to standard mode (i.e., between 186 and 558 bytes as depicted in Figure 3.11(a)).

This is four times larger than in standard mode, but still comparable with Paillier. After sharing (i.e., re-encryption) the ciphertext sizes expand by a factor of 1.7 due to pairing.

Encryption/Decryption. Figure 3.11(b) shows the performance of encryption and decryption in the sharing mode, for different integer sizes and security levels. Overall, the sharing mode is slower than standard mode by a factor of 2.2 to 3.3, but remains within acceptable bounds, that is, below 30 ms.

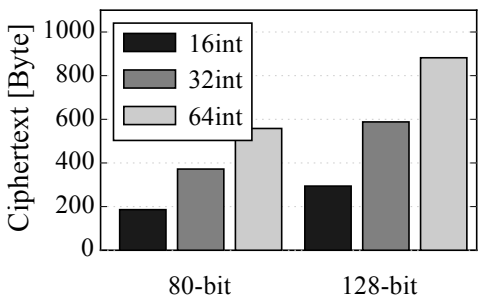
Note that this overhead is, to some extent, offset by the performance gains of offloading the re-encryption operation to the cloud. To enable sharing, the client only needs to generate a token (takes 2.5 ms) and then encrypt the data in sharing mode. As depicted in Figure 3.11(c), a client can issue a few hundred re-encryption tokens within 500 ms. Issuing a large number of re-encryption tokens becomes relevant after access revocation, as discussed in Section 3.5.4. The performance of re-encryption at the cloud is evaluated in Section 3.7.1.

Homomorphic Addition. The homomorphic addition with the data sharing mode is more efficient than the standard mode. Note that with data sharing, we have two types of additions: prior share and post share, which amount to 31 and 15 μs per CRT, respectively (see Table 3.2).

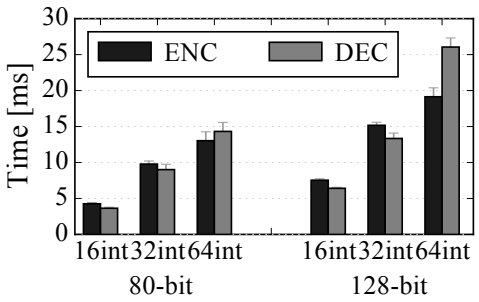
3.7.0.3 Constrained IoT Devices.

We now turn our attention to constrained IoT devices and evaluate encryption performance with the OpenMote's hardware accelerator. For the time and energy measurements, we rely on a dedicated hardware timer with an accuracy of 1 μs and a mixed signal oscilloscope, respectively. Paillier in 80-bit security requires 1.8 s to encrypt a 16-bit value. The same encryption with EC-ElGamal is significantly more efficient with 126 ms (corresponds to 6.85 mJ). Table 3.2 compares the results of constrained devices to smartphones. Although the encryption time on constrained devices is more than one order of magnitude higher than encryption on the more powerful IoT gateways, it is still feasible with only 10% of the daily energy budget of a typical Fitbit (400 mAh lithium-polymer battery) to encrypt at a rate of 0.24 Hz. To put this number into context, assuming heart rate tracking at 1 Hz during sport and six times per hour otherwise[5], we can encrypt the heart-rate of a person with 6 hours of sports activity per day.

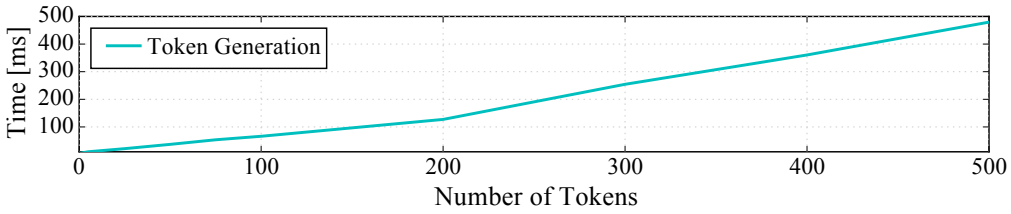
[5]Microsoft Band: `http://www.windowscentral.com/how-often-microsoft-band-checks-your-heart-rate`



(a) Ciphertext size (bandwidth/ memory).



(b) Enc-/decryption in sharing mode.



(c) Batch token generation (for re-encryption) with 80-bit security and 4x threads.

Figure 3.11: On-device Data Sharing evaluation. The Client engine performs encryptions and decryptions of data. After revocation, new re-encryption tokens are issued for valid sharing relationships. Note batch encryption/decryption with multithreading (e.g., 4 threads) yields a factor of 3 performance improvement, e.g., decryption of 64-bit integers is reduced to 5.5 and 10 ms, for 80-bit and 128-bit security, respectively.

3.7.1 System Benchmark

This section evaluates Pilatus as a full system: processing throughput, end-to-end latency, and our two case study applications: Fitbit and Ava.

Methodology. We utilize our client and cloud engines for the system benchmark. The client engine has an average ping time to the Amazon cloud of 22 ms, or it can be co-located at the cloud, neglecting network latency in favor of more isolated throughput measurements. Similar to micro-benchmarks, we rely on the *Stopwatch* class for the time measurements, instrumented at the client engine. To compute the system's throughput at the cloud engine, we initiate SQL `sum` queries over a varying number of values from our client module. For the end-to-end evaluation, the ciphertexts are additionally decrypted and provided to the corresponding application.

Encrypted Query Processing. Figure 3.12(a) and Figure 3.12(b) depict the performance of the cloud engine on AWS when performing `sum` SQL queries, over either plaintext or encrypted data. We create queries with

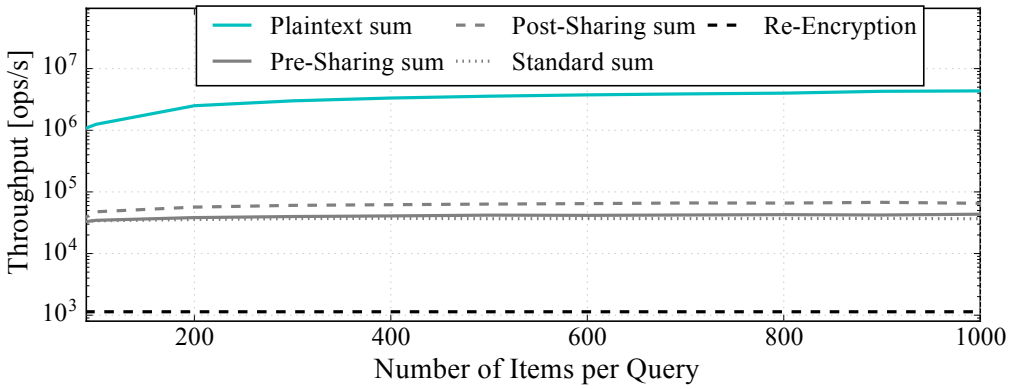
variable lengths (i.e., summands) and measure the time to process each, and compute the average system throughput of the cloud engine for a single connection. In Figure 3.12(a), we initiate the requests locally from the cloud (no network latency) and in Figure 3.12(b) from the client engine over the Internet.

Without consideration of network communication, plaintext sum operations are about two orders of magnitude faster than the homomorphically encrypted sums. With network communication, the relative performance loss is lower, in particular when only a few items are added and network delay is the bottleneck. However, the larger the number of items, the more the overhead of homomorphic additions impacts the performance, e.g., with 1000 values to be added, the performance loss reaches a factor 2.7. Note that such queries are highly parallelizable, allowing for better performance; plaintext operations already benefit from parallelization. Our current evaluation results show the lower bound performance and in future work, we plan to parallelize our routines at the database.

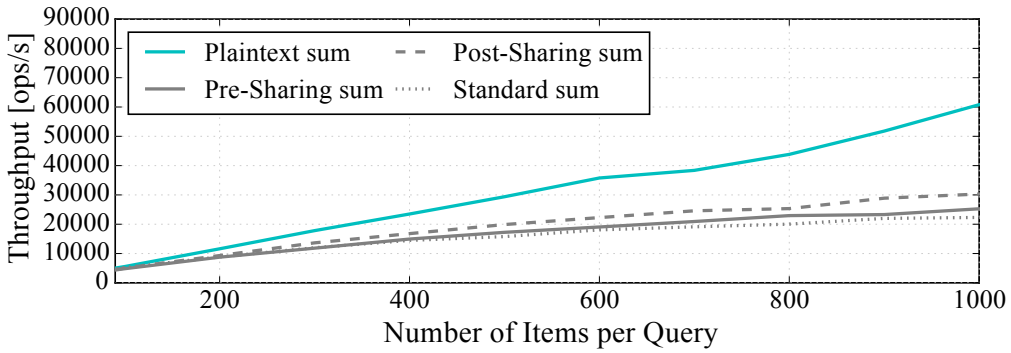
For data sharing, the computations take place offline in the cloud, without user interaction. The throughput of re-encryptions per data item amounts to 1136 re-encryptions per second, with a single thread (see Figure 3.12(a)). Note that re-encryption is the most expensive operation as it involves expensive pairing. The re-encryptions should also be parallelized in the cloud to reach the best performance.

End-to-end Latency. Figure 3.12(c) depicts the end-to-end latency for varying `sum` queries. The latency values follow a similar trend as the throughput values, as depicted in Figure 3.12(b). For lower range `sum` queries, the average performance of data sharing and standard mode are close to queries over plaintext. For larger ranges, the average latency increases by a factor of 2 and 2.7, respectively. To guarantee a smooth user interaction with encrypted data, the latency should be below 1 s, which is the case even for larger ranges (i.e., below 50 ms for 1000 items).

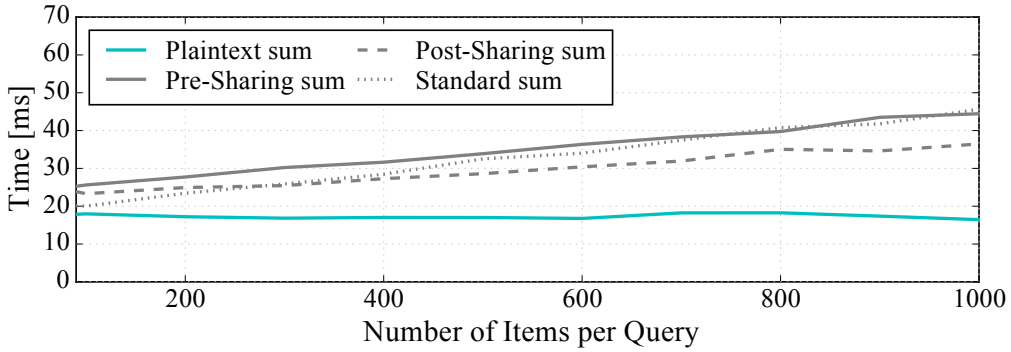
Applications. Our two Android applicationsrun Pilatus on real-world collected data and allow the user to upload encrypted data and interact with them similarly to the original apps (see Table 3.4). Our FitBit app adds an overhead of 1.3 s for uploading data of one day – an operation that takes place in the background. While rendering different views of the app (e.g., daily, weekly, detailed graphs), we measure a maximum overhead of 32 ms due to decryptions. Note that we use no local caching to emulate worst case scenarios. Our Ava fertility tracking app collects data from a more diverse set of sensors at a higher granularity during sleep and hence produces more data points. Additionally, we discuss the numbers in the context of the more expensive data sharing mode. Our Ava app induces



(a) Operations in the cloud, initiated locally at the cloud.



(b) Operations in the cloud, initiated by the client over the network (22 ms ping).



(c) End-to-end latency including decryption.

Figure 3.12: Cloud evaluation of Standard Mode and Data Sharing, with our cloud engine running on Amazon, with 80-bit security. Pre- and post-sharing sum refer to `sum` queries before and after sharing.

an overhead of 31 s for uploading one day worth of data. For rendering different views after sharing, the maximum overhead is 127 ms. For both apps, the overhead due to decryption is well below the 1 s requirement. Hence, the user experience with Pilatus remains unaltered.

App (mode)	Upload (encrypt)		View (decrypt)	
	[s]	[# items]	[ms]	[# items]
Fitbit (Standard)	1.3	1500	30	50
Ava (Sharing)	31	9000	127	50

Table 3.4: Fitbit and Ava enhanced with Pilatus. Encryption overhead when uploading one day worth of data and decryption overhead at visualization for the most costly view. Security is set to 80-bit, all data items are 32-bit values, and multithreading enabled.

Conclusion. Our evaluation shows the practicability of Pilatus, especially for mobile platforms which play a vital role in the IoT ecosystem (i.e., as the gateways). The end-to-end latency results show that Pilatus succeeds in preserving the user experience while interacting with encrypted data. Pilatus induces a moderate overhead for an increased level of data privacy and security.

3.8 Related Work

In the following, we discuss important research directions in relation to Pilatus.

Encrypted Search. Recent advancements of fully homomorphic encryption [83] have resulted into implementable schemes [84, 45, 204], which are however presently too slow for real world applications. Searchable encryption schemes support only a limited set of operations, but can be efficiently used in specialized domains. Song et al. [223] introduced the first encrypted search scheme for text files, where the metadata is encrypted deterministically and hence searchable. Their idea is based on deterministically encrypting the meta information of files, and hence being able to search over them. Follow-up schemes address other problems such as encrypted data de-duplication [132], deep packet inspection [218], and private network function virtualization [12]. More capable search schemes [184, 213, 219, 38, 199, 204, 198], targeted for structured databases, employ additional techniques such as partially homomorphic and order-preserving encryptions. Among these, CryptDB has early adopters in industry [162, 93]. Monomi [231] improves the performance of CryptDB and extends supported queries. In CryptDB, the application server has access to keys and carries out en-/decryptions. Hence, it can leak information if compromised. Talos [213, 206] tailors

CryptDB for IoT devices and eliminates the need to trust the application server. Mylar [185] introduces encrypted text file search with multiple keys. Shi et al. [220] propose private aggregation for time series data, which blends secret sharing with homomorphic encryption. Access patterns to encrypted data still leak sensitive information about the plaintext data. This shortcoming can be addressed with Oblivious RAM approaches [195, 227]. Pilatus is the first practical system to support processing of multi-key encrypted data and is tailored for constrained devices. This opens practical encrypted data processing to a new space of applications that existing systems do not support.

MPC. In traditional Secure Multi-Party Computation (MPC) [238], private functions are computed among a set of users without a trusted party. Hereby individual values from participating users are kept confidential, while the outcome can be public. This requires high interaction between users, which would drain the limited resources of mobile platforms. With the rise of cloud computing, server-aided/outsourced MPC approaches have emerged. However, these schemes are either only of theoretical interest [154] or require at least two non-colluding servers, where for instance one server has only access to encrypted data and the other server has access to the keys [233, 182, 175].

Trusted Computing. An orthogonal approach to encrypted computing assumes a trusted computing module on an untrusted cloud environment [21, 160, 17]. The data remains encrypted at rest and is decrypted in the trusted module for computations. This approach is appealing to data center operators, due to control over hardware. However, it implies that users consider the trusted computing module trustworthy. Autocrypt [230] combines PHE and trusted computing to enable encrypted computing on sensitive Web data on virtual machines.

Re-Encryption. The idea of Re-Encryption (RE) has been initially proposed for email forwarding. The initial schemes [32, 125] have the bi-directional property and are not resistant against collusion. Moreover, the parties need to exchange their private keys. The later schemes [13, 98] address these weaknesses and are uni-directional and non-interactive. Pilatus utilizes the RE scheme by Ateniese et al. [13] to allow transformation of encrypted data for data sharing. More importantly, we extend this scheme with the CRT technique, to render it efficient. The symmetric-key RE based on the key-homomorphic PRF scheme [40] lacks our required homomorphic property and master-secret secrecy. Sieve [234] utilizes this key-homomorphic scheme to provide cryptographically enforced access control for cloud data. Sieve's key revocation assumes that the cloud does not yield access to compromised

shared keys. Otherwise, the new key will be automatically compromised as the cloud can use the old key to compute the new key from the re-keying delta. In Pilatus, even with access to the revoked key of Bob and re-keying delta, the cloud cannot learn the new key of Alice.

3.9 Conclusion

We presented Pilatus, a new practical system tailored for the IoT ecosystem. We empower the user with full control over their data, despite it being stored in third-party clouds. In Pilatus, the cloud does not have access to any secret keys and stores only encrypted data. It can though process queries on encrypted data and re-encrypt it for sharing. Our sharing scheme comes with cryptographic guarantees and the possibility of revocation. We have optimized the underlying cryptographic operations towards mobile platforms. Our implementation and case studies on Fitbit and Ava show that Pilatus has reasonable overhead in processing time and end-to-end latency. We anticipate the presented cryptosystem and open-source platform to be helpful for the design of secure mobile applications and to enable further research in this field.

4

Droplet

This chapter presents *Droplet*, a decentralized data access control service, which operates without intermediate trust entities. Droplet enables data owners to securely and selectively share their data, while guaranteeing data confidentiality against unauthorized parties. Droplet handles time-series data, and features a cryptographically-enforced fine-grained and scalable access control for encrypted data streams. In this chapter, we present Droplet's design, the reference implementation of Droplet, and experimental results of three case-study applications on top of Droplet: Fitbit activity tracker, Ava health tracker, and ECOviz smart meter dashboard.

4.1 Introduction

Billions of Internet-enabled things contribute relentlessly to a new data deluge that needs to be managed, processed, and adequately secured to realize the full potential of the *Internet of Things* (IoT). IoT systems are commonly comprised of decentralized and collaborative entities. These include heterogeneous IoT devices (i.e., data sources), infrastructure entities (e.g., storage), and various services that exploit the sensor data. Often, these entities span across different protection domains, execution environments, and communication channels. Hence, security and access control are of critical concern, particularly as data in this space is inherently privacy-sensitive. In today's IoT development, user data is often scattered across multiple storage silos whose access control is governed by corresponding applications. This leaves data owners with

little control over their data; users basically lack the ultimate authority to determine how and with whom their data is shared. This chapter focuses on providing users with control over their data in decentralized settings.

State-of-the-art distributed access control mechanisms, such as OAuth2 [153] and Macaroons [31], require central trusted parties to issue and verify access tokens. Users have no choice but to rely on a third party's promises of security and privacy protection. Trusted third parties are, however, inherently prone to compromise [30], misconduct [59, 29, 23], collusion/corruption [241], and coercion (e.g., susceptible to pressure by autocratic institutions to violate user's data privacy rights [164]). Further, these schemes cannot enforce any cryptographic constraints for data access, as they are decoupled from the underlying data protection. Consequently, the user has no cryptographic guarantees that their data will not be shared against their will, nor that the sharing relationship will remain private. In this work, we realize a secure decentralized access control scheme that diminishes the need for trusted third parties, supports auditability, maintains the privacy of the sharing relationships, and more importantly is cryptographically enforced.

To address these challenges, we introduce Droplet, a new secure data access management system that we design for IoT time-series data. In Droplet, we realize a cryptographically-enforced decentralized access control service to empower the user with data ownership. Droplet leverages the blockchain technology to bootstrap trust, for a decentralized, secure, and resilient access control management. Blockchains like Bitcoin [167] and Ethereum [70] and their respective peer-to-peer networks have seen notable adoption in the last few years, allowing the realization of powerful decentralized systems without trusted intermediaries [5, 73, 188, 225, 178, 177], beyond cryptocurrencies. Realizing Droplet requires overcoming the challenges of: *(a)* the narrow bandwidth and privacy-transparency tension in blockchains, and *(b)* designing a private access control service unified with crypto-enforced data access to enable fine-grained sharing of data streams.

In Droplet, IoT devices produce streams of encrypted data without directly handling the access permissions. Instead, data owners register data streams and securely associate privacy-preserving access permissions in the blockchain. Only the data owner can subsequently adjust access permissions, i.e., grant access to new principals or revoke existing access. Additionally, the integrity of data can be protected via the blockchain (i.e., secure time-stamping), such that even the data owner cannot later alter data (e.g., necessary in contractual arbitration). Moreover, the cryptocurrency feature of blockchains enables a self-sustained ecosystem with economic incentives

(e.g., payment for storage [189]).

Public blockchains inherently exhibit a high overhead and low capacity due to their consensus protocols. To overcome this limitation, we place only the minimum necessary logic of our system in the blockchain, via an indirection. Though chain writes are inherently slow, they are not bound to time critical operations in Droplet. Only granting/revoking access permissions and data timestamping require writing to the chain. All other operations, such as data storage and retrieval, (i.e., write/read operations), use only the read operation which is not subject to this limitation. Note that in Droplet, data streams are stored off-chain, similarly via an indirection. We construct our access control state machine on top of a running blockchain, such that any node can independently bootstrap the state in a decentralized manner and check the access permissions.

Realizing privacy-preserving access permissions that protect the privacy of sharing relations (i.e., who is granted access to what) is an essential property of our access control system. This is, however, particularly challenging in public blockchains, because of the transparency of transactions. We overcome this challenge by employing the cryptographic technique of dual-key stealth addresses [57], ensuring unlinkability between the set of all principals.

Our system is designed from the ground up to support continuous data streams. We encrypt chunked data streams at the application layer on the client, and only authorized principals are cryptographically able to access/decrypt the intended data chunks. We design a novel key distribution and management scheme to enable efficient key updates and fine-grained yet scalable sharing of both arbitrary ranges and open-ended streams. Our design builds on key regression and hash trees via a layered encryption technique.

The underlying physical storage is oblivious to our design and we support various storage modalities. Droplet storage nodes can consist of on-premise storage, locality-aware decentralized storage (e.g., IPFS [128]), or the cloud. Storage nodes in Droplet act as primitive nodes, decoupled from the data access control logic. Recent efforts explored the benefits of this decoupling paradigm [234, 50, 217], which enables seamless interplay of heterogeneous applications. This is particularly relevant for the IoT, as heterogeneous services will integrate data from various sources (e.g., smart thermostats that sense thermal comfort from wearables [20]).

In Droplet, we build on research in time-series databases [147, 101, 75, 7], key management [78, 46], and recent blockchain-powered efforts, e.g., in archiving files [236, 189] and domain name registries [5]. The contribution of this work is the design of Droplet, a novel

decentralized access control service for time-series data. More specifically, our **contribution** consists of *(i)* a decentralized access control system with privacy-preserving permissions, *(ii)* cryptographically secure data sharing with fine-grained yet scalable access to encrypted data and the possibility of access revocation, *(iii)* a flexible storage layer tailored for time-series data with optional blockchain-based integrity protection. Our work is the first to enable a (trustless) decentralized access control service that works with encrypted data streams.

With a prototype implementation[1] of Droplet on a public blockchain, we quantify Droplet's overhead and compare it to the state-of-the-art systems. When deploying Droplet with Amazon's S3 as a storage layer, we experience a slowdown of only 3% in request throughput. Moreover, we show the potential of Droplet as authorization service for the serverless computing domain with an AWS Lambda-based prototype. We show Droplet's performance is within the range of the industry-standard protocol for authorization (OAuth2). Also, we deploy Droplet with a decentralized storage layer (similar to the peer-to-peer storage service IPFS [128]). With our example apps on top of Droplet, we show that real-world applications with unaltered user-experience can be developed.

Droplet comes with certain limitations that we hope to address collectively within the research community and future work. For instance, the time until access permissions come into effect is bound by the transaction time of the underlying blockchain. We give an overview of various state-of-the-art blockchain solutions that Droplet can utilize. However, we acknowledge that addressing the scalability issues of blockchains is a prominent challenge that has a direct impact on systems like Droplet, requiring further research.

This chapter is based on the contributions made in [207].

4.2 Overview

We discuss the design goals of our system, review relevant aspects of the IoT, and give a primer on the blockchain technology. We conclude with a discussion on the security guarantees and assumptions of Droplet.

4.2.1 Design Goals

Droplet's primary goals are to retain users ownership and control over their data in a trustless setting, as a means to achieve a higher level of privacy and security compared to the state-of-the-art

[1] Droplet is available under `https://github.com/dropletchain`

	Identity-based	No Trusted Intermediary	User-Centric	Privacy-Preserving Access Permissions	Crypto-enforced Data Access	Revocation	Computation Cost
OAuth2 [153]	✗	✗	✗	✗	✗	token lifetime	Implementation dependent
Macaroons [31]	✗	✗	✗	✗	✗	token lifetime	Hash function
ABE [97, 234]	✗	✓	✓	(✓)	✓	re-encryption [234]	Pairing-based crypto
SPKI/SDSI [66]	✓	(✓)	(✓)	✗	✗	ACL	Asymmetric crypto: signature
Droplet	✓	✓	✓	✓	✓	ACL + enc	Asymmetric crypto: signature

Table 4.1: Overview of state-of-the-art distributed access control schemes in comparison to Droplet.

access control schemes. We define data ownership as having the right and control over data, wherein the owner can define/restrict access to their data, restrict the scope of data utility (e.g., sharing aggregated/transformed/homomorphically-encrypted data, instead of raw data), delegate these privileges, or give up ownership entirely. A true realization of this definition requires work on two fronts: *(i)* privacy-preserving computation (i.e., differential privacy and secure computation) and *(ii)* decentralized access control without centralized trust entities, to ensure private access to externally hosted data, with strong confidentiality guarantees. In this work, we focus on the latter, specifically in the context of time-series data. In the following, we briefly discuss limitations of current solutions in facilitating ownership and make a case for Droplet. Table 4.1 provides a qualitative overview of current approaches compared to Droplet.

Distributed Access Control. Research in distributed authorization has matured in the last decade with important prior work. The decentralized authorization model of Droplet is a combination of these techniques with a trust-less realization. Current distributed access control schemes, such as OAuth2 [153], and Macaroons [31], rely on tokens issued by a trusted intermediary serving as an identity provider. Users present these tokens (i.e., credentials) to a gatekeeper of a resource, e.g., data. The gatekeeper forwards the token to the issuer who in return confirms the validity of the token or rejects it, if invalid. These schemes are reliant on trusted authorities for token issuance and validation. Today only a handful of centralized authorities control this space, e.g., Google, Facebook, and Amazon, who as well learn about all the services a user calls on. This is, however, undesirable and not aligned with our goal of diminishing the role of trusted intermediaries. Moreover, these schemes come without any cryptographic guarantees and lack built-in audibility features. These systems, despite being architecturally decentralized, they are logically centralized (in contrast to Droplet).

Though signature-based schemes (e.g., public-key based certificates [33, 66]) do not suffer from these limitations, they require a centralized, hierarchical network of certification authorities (CA) to issue certificates, which come with their weaknesses, as extensively studied by prior work [158]. Alternative decentralized public-key based approaches, e.g., SPKI/SDSI [66] and follow-up schemes [67], eliminate the need for complex X.509 public key infrastructure and CAs. However, these schemes are either based on the idea of local names and suitable for deployments under a single administrative domain (e.g., smart home) or build upon an organically growing trust model (i.e., Web of Trust [243]). While the key idea of public-key based schemes underpins Droplet,

we leverage a public blockchain to overcome the challenge of certificate-chain discovery (Section 4.3.1.1). In Droplet, data owners directly issue access permission rights (i.e., similar to signed certificates) for their data streams, where the blockchain allows us to ensure the global ordering and protect the integrity and authenticity of access permissions.

Crypto-enforced Data Access. To meet our design goals, our system should ensure strong data protection using client-side encryption, to avoid privacy risks due to: *(i)* Unauthorized trade with users data (e.g., targeted advertising). *(ii)* Access by rogue employees to users data (i.e., insider attacks). *(iii)* Data leakage due to system compromise. *(iv)* Collaboration/coercion with government agencies without user's consent. While cloud providers support various options of encryption for data protection [100], they have access to the encryption keys, specifically in encryption at rest.

Though client-side encryption provides a stronger level of protection, it adds constraints on sharing. Data in this setting can only be shared at a coarse-grained level. Various cryptographic schemes [36, 13] have been introduced to overcome this limitation, among which attribute-based encryption (ABE) [97, 234, 197, 96] offers the best expressiveness in this space. At a high level, in ABE data is encrypted towards a policy (i.e., associated with a set of attributes), and only those with the secret keys satisfying the policy can decrypt the data. Several ABE-based systems [234, 239] introduce crypto-based access control for remote storage services. However, ABE suffers from expensive crypto operations (due to the underlying pairing crypto), and the costs grow linearly with the number of attributes, limiting the granularity of access due to computational burdens [80, 3]. The overhead dominates even with a hybrid encryption technique [234, 239], where large amounts of data is encrypted with fast encryption and the rather small encryption keys are encrypted with the expensive ABE, e.g., only two attributes result in 100 ms for enc/decryption on desktops and few seconds on low-power IoT devices [235]. Hence, in Droplet, we opt to design a new crypto-enforced data access mechanism that is tailored for the velocity of data streams and supports scalable fine-grained sharing (Section 4.3.1.3).

4.2.2 Background

Internet of Things Data. The pre-dominant IoT system design consists of the stove-piped architecture [240], where IoT devices are tightly coupled with a specific application and stream their data directly to the cloud. resulting in users data being scattered across multiple isolated storage silos. Realizing the limitations of this design, i.e., latency, privacy,

durability, interoperability, has led to the emergence of new design paradigms, e.g., IoT edge computing [242, 55, 144] and designs that decouple data storage and applications [234, 50, 217]. The collected data typically consists of immutable time-series, i.e., append-only data records, with a single writer and multiple readers. Time-series data is deemed as the most pervasive type of data in the IoT space [7, 51, 122]. It is characterized as a sequence of data points (i.e., time-stamped records), where time is a primary axis. Data in this space is inherently privacy-sensitive, since it embodies multi-dimensional representations of our immediate environment. Today, IoT services collect and control sensitive data (e.g., health, home, car, agriculture, etc.) with little or no transparency. Privacy and security concerns have been steadily rising, notably for the IoT, due to the surge in data breaches [134, 18], misconduct of service providers [59, 29], and surveillance [186, 23]. Enabling secure and transparent sharing of the data is crucial to the success of the IoT. This is specifically relevant for bilateral data sharing, which in contrast to multilateral sharing (e.g., crowd-sourcing, model training), cannot benefit from group privacy (e.g., differential privacy [68]).

Applications. In the following, we discuss two classes of IoT applications to understand their requirements better. In Section 4.5, we evaluate Droplet on top of these apps.

Quantified Self. Health and fitness applications are one of the prominent domains in the IoT space. For example, Fitbit wristbands collect users' heart rate, step counts, and location data. Similarly, certain health tracking applications and wearables, such as Ava [14], allow women to track their menstrual cycle, predict (in)fertile phases, and detect potential health issues. Since privacy-sensitive information can be inferred from this data (e.g., illness, lifestyle, or location), data security is of utmost importance for these applications. Moreover, secure and transparent sharing plays an essential role in unlocking the full potential of these applications. Users should be in full control of their data and be able to securely share it with experts (e.g., medical practitioners), analytical services, or just casually with friends. We extend example Fitbit and Ava apps with Droplet and show that the user experience remains unaltered.

Smart e-Metering Smart electricity meters are being deployed in millions of households worldwide to collect fine-grained electricity consumption data. Electricity providers utilize this data for load management and billing purposes. Beyond enabling dynamic pricing policies, it enables other interesting applications [140, 25, 221, 145]. In our analysis, we consider a scenario where users utilize Droplet for storing fine-grained electricity data generated by smart meters. Users can authorize sharing of, e.g., daily aggregated energy consumption values with utility providers

for billing. More specifically, our case study is on ECOviz, an interactive dashboard app [139] to monitor and analyze home energy consumption.

Blockchain. Conceptually, blockchain is a technology that facilitates the realization of distributed applications without the need for a centralized trust, by distributing trust in the peer-to-peer network that spans it (i.e., democratize trust). Cryptocurrencies [169], such as Bitcoin [167] and Ethereum [70], leverage blockchains to enable mutually mistrusting parties to trade, without requiring traditional trusted centralized intermediaries. A blockchain is essentially a distributed ledger that consists of a continuously growing set of records. Tampering with past records is prevented with a high computational barrier. The distributed nature of blockchains implies no single entity controls the ledger (i.e., censorship/coercion resistant), but rather the participating peers together validate the authenticity and ordering of records. These records are organized in blocks which are linked together using cryptographic hashes, ensuring the ordering of the transactions. The blockchain incentivizes the network of peers to carry out computations towards a network-wide consensus, i.e., solving a computational intensive mathematical puzzle referred to as proof-of-work (PoW).

Permissioned (closed) blockchains have a designated set of authorized validators and use a variant of the practical Byzantine fault tolerance (PBFT) [49] consensus, which tolerates a malicious behavior by f validators among $3f + 1$. Since, the set of validator nodes is known, PBFT can handle a higher transaction throughput compared to PoW blockchains, i.e., 10 vs. 10^4 transactions per second. However, PBFT has its limitations. Most importantly, it requires a trusted third party to initially authorize the set of validators. Moreover, due to the high communication overhead of the consensus protocol (in $O(n^2)$), only deployments with up to a few tens of validators are practical. Note that given a modified trust assumption, permissioned blockchains can also be leveraged in Droplet.

Blockchain Evolution. *Permissionless* (public) blockchains have to cope with a dynamic set of membership, where anyone can join and leave at any time. Hence, they leverage the expensive PoW to mitigate sybil attacks which induces high overhead regarding throughput, latency, and energy footprint. To understand the extent of this overhead, consider Bitcoin as an example; which has currently a throughput of 7 transactions per second with an average latency of 10 min and finality after 6 blocks [43]. Academia and industry are persistently working on designing next-generation blockchains [87, 47, 72, 163, 143] to achieve higher throughputs and lower latencies, which is crucial for the adoption of cryptocurrencies in retail payments and financial sector, and for realizing practical large-

scale decentralized applications. Recent works [142, 143] introduce a hybrid consensus by combining the slow PoW to bootstrap the faster PBFT algorithm, where for each epoch a random set of validators is selected. Hence, they bring the best of both worlds: secure open enrollment and high throughput and low latency. Other works [87] depart from PoW completely and alternatively employ a publicly verifiable, unpredictable, and deterministic source of randomness to select a dynamic set of validators. These scalable blockchain protocols, e.g., OmniLedger [143], lay the groundwork enabling practical advanced decentralized services, such as Droplet.

4.2.3 Security Model

Threat model. The threat model addressed by Droplet consists of a passive honest-but-curious adversary, who is interested in learning about users data without necessarily being noticed (i.e., it follows the protocol correctly without deviating from it). Our threat model covers malicious storage nodes, potential real-world security vulnerabilities leading to data leakages, and as well external adversaries who gain access to data as a result of system compromise. Moreover, an adversary can launch a data scraping attack against storage nodes.

Guarantees. Droplet provides users with autonomy and control over their data. As depicted in Figure 4.1, after an initial pairing of the IoT device's public-key-based identity to that of the data owner via the blockchain, the IoT device is authorized to store its data on the storage provider directly, which can consist of on-premise, decentralized, or cloud storage. Data is encrypted at the client-side and keys are never revealed to the storage provider, guaranteeing confidentiality. Decryption keys are only shared with authorized parties via a blockchain-based indirection. Data chunks are digitally signed, allowing parties without decryption keys to verify data ownership and integrity. Droplet enables checking the freshness of data and it provides data immutability optionally via an authenticated data structure anchored in the blockchain, such that even the data owner can no longer modify past data. Droplet cryptographically prevents evicted users from accessing future data. However, evicted users may have cached old data locally. Moreover, a new device owner will have no rights over past data. Droplet encodes user-defined access permissions on the blockchain, eliminating trusted intermediaries and assuring collusion-resistance and auditability. Even malicious institutions cannot illegitimately modify access permissions. Moreover, we employ privacy-preserving access permissions, preventing an observer from learning the identities of the sharing parties.

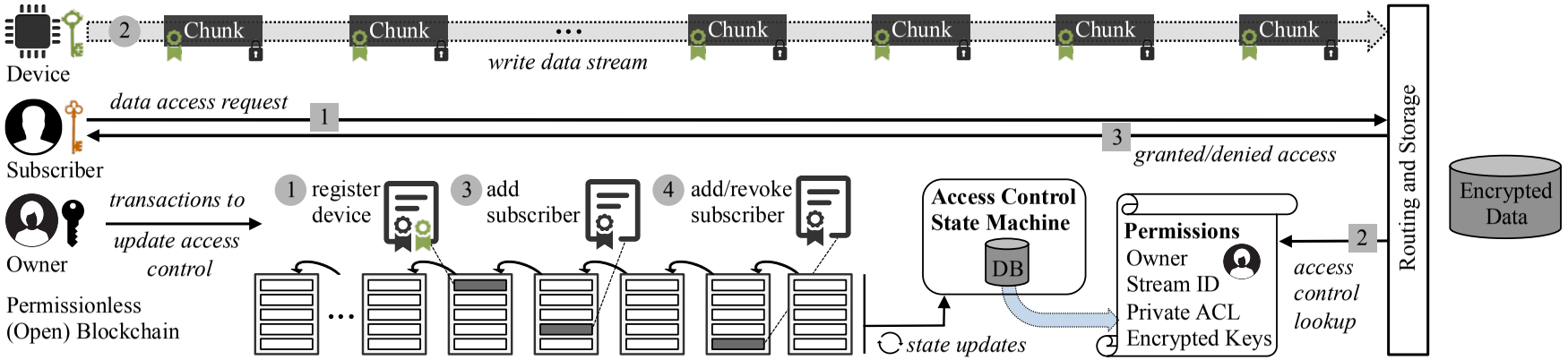


Figure 4.1: System overview of Droplet. Separation of storage and control layers. Data owner defines the access permission policies. The storage layer validates access requests based on the user-defined access permissions.

The combination of encryption-based and decentralized access control mechanisms enable Droplet to address the above threat model. Droplet does not protect against denial-of-service attacks nor does it hide access patterns. It could be extended with ORAM techniques to hide access patterns [127, 226]. Cryptographic techniques alone are not sufficient to prevent a malicious storage provider from denial-of-service or deconstruction of data. Hence, adequate replication strategies on multiple providers are necessary to ensure preservation and availability of data.

Assumptions. In Droplet, we make the following assumptions to provide the above guarantees. We assume the storage nodes to be honest-but-curious, such that they would follow the protocol correctly. This is a valid assumption, since the storage node could face financial (and potentially legal) consequences upon detection of misbehavior. We assume the adversaries to be subject to the standard cryptographic hardness and the underlying blockchain to be secure. We assume users store their private keys securely and that multi-device key recovery techniques are deployed [214]. We assume the correctness of data generated by IoT devices and more importantly that only the owner has physical access to the device. We assume that there exists a financial agreement between the storage provider and data owner (i.e., to provide persistent storage) which can be facilitated through the cryptocurrency feature of the underlying blockchain.

4.3 Droplet Design

Our design considers the following main parties: **Data owner** is someone who owns a set of **IoT devices** (e.g., wearables, appliances, or apps) which produce time-series data. In an industrial setting, the data owner can be an organization that owns a swarm of IoT devices. Data owners store data generated by their devices online and can decide to selectively expose their data to **principals** who can produce an added value from their data. The **storage provider** is in charge of storing data and providing access to principals as defined by the data owner. The storage node can take various forms, such as edge (e.g., gateway), decentralized (e.g., a node in a peer-to-peer storage service [128]), or cloud storage (e.g., Amazon's S3). The owners claim their ownership by binding their IoT devices to their identity. The identity of a principal is established by verifying the validity of the corresponding public-private key pair. A principal can be granted access to arbitrary intervals of past data or a data stream subscription, which is valid either temporarily or until revoked. Note that an owner

can be represented by a single or subset of personal and trusted devices, e.g., smartphone and tablet.

Droplet in a Nutshell. In Droplet, we decouple the control and data planes, enabling data owners to express fine-grained access permissions independent of a trusted intermediary (Section 4.3.1.1). During the initial setup, the owner and the device share a master key out-of-band, which is later used to generate data encryption keys (Section 4.3.1.3). In this setup, the public key of the device is bound to the owner's public key. Thereafter, the device independently stores encrypted data in the data plane. Droplet's data plane is tailored for time-series data and provides an adequate query interface (Section 4.3.2.1). We realize the control plane with an access control state machine on top of a public and deployed blockchain (Section 4.3.1.2). The access control state machine assembles the current global state (i.e., access permissions and data ownership) through embedded state transitions. The control plane serves as well as a decentralized entry point to the storage layer.

Challenges. To realize Droplet, we have to overcome several technical challenges; *(i)* Blockchain write operations are inherently slow and on-blockchain memory is scarce and expensive. *(ii)* Transparency and privacy are at odds in public blockchains, as any blockchain data is accessible by the public, impacting the privacy of access permissions in Droplet. *(iii)* A crypto-enforced data access implies access to data authorized by encryption keys. For time-series data with continuous data coming in, we have to design a low overhead and efficient key management to enable a fine-grained and expressive access control management. We now introduce and elaborate on different aspects of Droplet and describe how we overcome the above challenges.

Without loss of generality, we begin with simplified descriptions of our system components and gradually converge to the full system design.

4.3.1 Control Plane

In Droplet, the control plane is logically separated and agnostic of the data plane. This separation is a fundamental property of our design. In the following, we elaborate on the control plane components. We discuss how we manage identities and access permissions. As the backbone of our encryption-based data access, we present the design of an efficient key-management scheme.

4.3.1.1 Blockchain

We employ a publicly verifiable blockchain to maintain an accountable distributed access control system and bootstrap trust in a network *without a central trust entity*. The reasons why we opt to discard trusted intermediaries and alternatively utilize blockchain for access control are manifold: *(i)* resilience against centralized corruption/coercion, *(ii)* identity management, specifically of relevance to the IoT, *(iii)* audibility of access permissions by authorized parties, *(iv)* immutability of data streams after a defined interval, *(v)* potential of nano-payments for storage services and data market. Droplet embeds ownership of data streams and corresponding access permissions in the blockchain transactions. We now describe the owner-device pairing, blockchain encoded access permissions, and how we protect the privacy of principals. We conclude with our approach to overcome the narrow bandwidth of blockchains.

Owner-Device Pairing. The blockchain ecosystem relies on public key cryptography for identification and authentication of the involved principals. The hash digest of the public key serves as a unique pseudo-identity in the network. We leverage this feature to allow IoT devices to securely and autonomously interact with the access control and storage layers. This way we overcome the hurdle of passwords and rely on public-key crypto for authentication and authorization. During the bootstrap phase of a new device, it creates locally a new pair of public-private keys, where the private key is stored securely (e.g., in the trusted hardware) and never leaves the device. Through an initial multisignature registration transaction on the blockchain, Droplet allows the binding of the IoT device (PK_{IoT}, SK_{IoT}) to the owner (PK_{Ow}, SK_{Ow}). Hereafter, the owner can set access permissions (via the signing key SK_{Ow}) and the IoT device is permitted to securely store data (via the signing key SK_{IoT}).

In the event of device decommissioning, the new owner must issue a new multisignature device-binding transaction, to gain ownership rights of future data. Note that there is no need for the IoT device to interact with the blockchain network directly. The owner creates the raw multisignature registration transaction and uses an out-of-band channel (e.g., BLE) to get the device's signature. After adding her signature, she broadcasts the register transaction to the blockchain network. During this process, neither the owner's nor the device's private keys leave the secure local memory area.

Access Permissions. We utilize the blockchain to store access permissions in a secure, tamperproof, and time-ordered manner. Access permissions are granted per data stream and the data owner can revoke the sharing of a data stream. Initially, the data owner issues a transaction including the

stream ID which creates the initial state. To change this state, e.g., grant read access permissions to a principal, the data owner issues a subsequent transaction which holds, among others, *(i)* the stream ID, *(ii)* the public key of the principal they want to share their data with, *(iii)* the temporal scope of access (e.g., intervals of past or open-end subscription), and *(iv)* encrypted keying material for data decryption (Section 4.3.1.3).

For any request to store or retrieve data, the storage node retrieves the access permissions from the blockchain in a decentralized manner. The requesting party proves (i.e., in our case via a digital signature with a freshness guarantee) that it is in possession of the correct private key. The blockchain-based access permissions provide auditable information about when access was granted for a stream and to whom. In the next section, we discuss how we allow only authorized entities to perform the audit. For the storage nodes, the access permissions allow them to protect network resources (i.e., bandwidth/memory) from unauthorized users. For instance, this mitigates an attack, where malicious nodes flood the network with download/storage requests of large files. The storage node can terminate malicious sessions (e.g., data scraping and storage spamming attacks) after checking the access permissions (Section 4.3.1.2). The impact of a malicious node handing out data without permission is low, since data is additionally protected via encryption (Section 4.3.1.3).

Privacy-Preserving Sharing. In blockchain, users are represented through virtual addresses, providing pseudonymity. However, advanced clustering heuristics can potentially lead to the de-anonymization of users [161, 8]. Access permissions in Droplet should be enforceable by storage nodes and be auditable by authorized parties. However, we want to protect the privacy of sharing relationships from the network. To realize this, we leverage the technique of dual-key stealth addresses [57] which builds on known crypto primitives (a modification of the Diffie-Hellman protocol). Dual-key Stealth addresses do not require off-blockchain communication (i.e., no out-of-band channel) and provide strong anonymity for the principals that are granted access permissions. Moreover, different streams shared with the same principal are unlinkable. Conceptually, in dual-key stealth addresses, each user is represented by two public keys (main and viewer keys), which are used by other parties to generate unlinkable new addresses. The viewer key can be shared with an auditor to audit the permissions. Note that we are not protecting the identity of a principal from a storage provider.

More specifically, let us consider the case of a data owner Alice giving access permission to a subscriber Bob. Bob has initially constructed and published his dual public keys (B, V): $B = bG$ and $V = vG$, with G as the elliptic curve group generator and the private keys b and v. Alice

constructs a new address P using Bob's stealth addresses by using a hashing function H, and generating a random salt r:

$$P = H(rV)G + B \tag{4.1}$$

Alice embeds the tuple (P, R) in the access permissions, with $R = rG$ (r is protected and not recoverable from R). Only Bob can claim the address P, as he is the only one capable of recovering the private key x, such that $P = xG$, as follows:

$$x := H(vR) + b \tag{4.2}$$

Hence, he can prove (e.g., with a signature) to the storage node that he is the rightful principal. Note that guessing x, given G and P, is equivalent to solving the elliptic curve discrete log problem, which is computationally intractable for large integers. The correctness of x from Equation 4.2 can be shown as:

$$\begin{aligned} xG = (H(vR) + b)G = H(vR)G + bG = \\ H(vrG)G + B = H(rvG)G + B = H(rV)G + B = P \quad \square \end{aligned} \tag{4.3}$$

Except Alice and Bob no other party can learn that P is associated with Bob's stealth addresses. Moreover, the randomness r in the address generation ensures the uniqueness and unlinkability of new addresses. To enable an authorized auditor to audit the sharing, Bob discloses the private viewer key v to the auditor. The auditor can verify the mapping of the tuple (P, R) to Bob's main key address B as:

$$\begin{aligned} P - H(vR)G = P - H(vrG)G = \\ H(rV)G + B - H(rV)G = B \quad \square \end{aligned} \tag{4.4}$$

Note that the auditor is cryptographically prevented from using v to compute Bob's private key x. The auditor breaks the unlinkability of the addresses linked to Bob for the auditing purpose.

ACL Indirections. Each change to the access permissions of a data stream requires a new transaction. The time until a change comes into effect is tied to the transaction waiting time of the underlying blockchain, ranging from few seconds to minutes depending on the blockchain. Moreover, transactions require a per-byte fee to ensure that they are added to the blockchain. To keep the number/size of transactions as low as possible, our design includes the off-chain storage of the access control list (ACL), as illustrated in Figure 4.2. The transaction, instead of holding the address information of all services, just includes an indirection to the ACL via the hash digest of it. This allows managing access permissions with an unlimited number of services in a single transaction. Similar to before, any

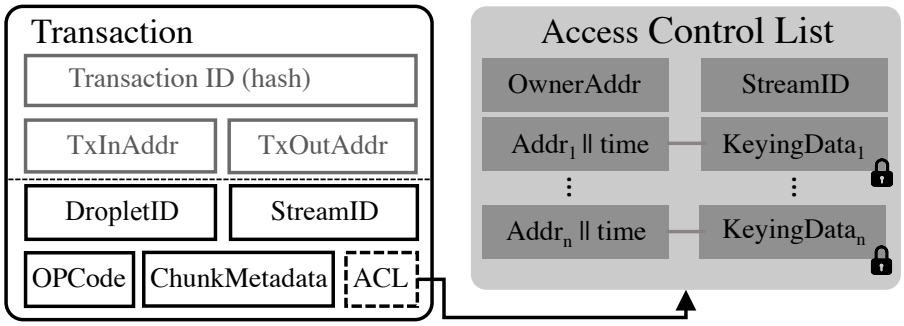


Figure 4.2: Overview of access control transactions. Transactions embed transitions to the global access control state via an indirection (i.e., hash to the ACL). `OPCode` takes values for issuing a new stream, granting/revoking access, changing chunk metadata, and terminating the stream sharing.

change to the ACL requires a new transaction. The hash digest serves as a data pointer and more importantly ensures integrity protection of the ACL. The ACL is stored off-chain in the storage network of Droplet (Section 4.3.1.2). Note that the ACL includes as well the encrypted key information for each service, as explained in Section 4.3.1.3.

4.3.1.2 Access Control State Machine

Today, there are three main options developers can take for realizing decentralized applications that employ a blockchain as a ubiquitous trust network (i.e., a shared ground truth): *(i)* operating a new blockchain, *(ii)* piggybacking on an existing cryptocurrency blockchain, *(iii)* using a blockchain with built-in scripting, such as Ethereum. We opt for the second approach where we piggyback our logic without alternation of the underlying blockchain. This allows us to benefit from the security properties of an existing production blockchain and make our design generic. We briefly discuss the reasons why we opt for this choice and detail on how we realize this efficiently.

Integrating a new application logic into a running production blockchain typically results in consensus-breaking changes and hard forks, i.e., a new blockchain with a subset of peers enforcing the new application logic. While necessary for specific applications, this results in parallel blockchains which may not exhibit strong security properties due to a smaller network of peers. To benefit from security properties of a strong and robust production blockchain, new applications include their log of state changes in transactions. This is in turn used to bootstrap the global state in a secure and decentralized manner. While offering the highest level of security, this approach requires addressing consistency and efficiency challenges.

We employ the approach of virtualchain [172] which addresses these challenges adequately. A virtualchain is a fork*-consistent replicated state machine, allowing different application logic to run on top of any production blockchain, without breaking the consensus. One of the most prominent systems utilizing virtualchains is Blockstack [5, 4], a decentralized domain registration service. While the combination of virtualchain and blockchain is comparable to Ethereum with its built-in scripting language, we explicitly decide to make our design independent of a specific blockchain, such that Droplet can be deployed on the most secure and efficient blockchain of choice. The virtualchain instance in Droplet creates a global state of access permissions based on the blockchain's totally-ordered and tamper-resistant transaction logs and updates the state according to new state transitions. A virtualchain instance essentially scans the blockchain for the corresponding access permission transactions and maintains the global state in a database that can be queried for access permissions of a given data stream and principal. The virtualchain approach [172] introduces several improvements that make maintaining the global state efficient and fast, such as automatic fork-resolution, cross-chain migration, and fast bootstrap. Anyone can run a virtualchain instance[2], either as a storage node to lookup access permissions or just to provide it as a service. The instances span among each other a peer-to-peer storage network, which we employ to store part of access control metadata, e.g., ACL and keying material. Due to the public nature of blockchain, third party apps can be developed around Droplet, e.g., allowing data owners to view/modify access permissions.

4.3.1.3 Key Management

In the following, we discuss how we realize an expressive and fine-grained sharing. In Droplet we distinguish between three types of sharing. *(i)* subscription, where the subscriber is granted continuous access to the data stream until revoked (e.g., sharing of fitness tracker data with an analytical service or friends), *(ii)* sharing arbitrary intervals of past data (e.g., sharing the fitness data of only past marathons or smart meter readings of past winters), and *(iii)* a combination of *i* and *ii*. To meet our design objective of *a fine-grained yet scalable encryption-based access control*, we design a low-overhead and efficient key management scheme, with the constraints of IoT devices in mind.

Our design basically combines hash trees [46] and key regression [78] and employs an efficient symmetric key encryption. While a hash tree gives us the expressiveness of defining arbitrary intervals of keys,

[2] A virtualchain node can run either as a full node or in lightweight mode.

it falls short meeting our low overhead requirement with regards to subscriptions. Key regression [78] provides an efficient and low overhead key management scheme for subscriptions, however, it does not support sharing of arbitrary intervals. In the following sections, we describe the components of our key management individually and gradually put them together as a whole system. We first describe the basic key-regression scheme combined with scalable key distribution. We then detail on how we extend the basic key-regression to support bounded interval sharing. Finally, we describe the integration of a hash tree to realize the maximum level of expressiveness for sharing arbitrary intervals of past data (e.g., weekends in 2018). As our key management heavily relies on hash chains, we later discuss how to construct compact chains for an efficient and fast key rotation.

Scalable Subscriber Sharing In key regression, initially a hash chain is created and later a secret key is derived from each hash token. Given a hash token, one can derive all previous keys by applying the cryptographic hash function successively. The pre-image resistance property of hash functions ensures forward-secrecy, such that given the current key (i.e., hash token) no future keys can be computed. However, given key K_t in time t one can compute all keys until the initial key K_0, i.e., $\forall_{i \in [0..t]} K_i$. This is, however, not always desirable. Before we discuss how we enable bounded interval sharing, let us discuss the key distribution aspect. For now, we assume our encryption keys are generated by key regression and that the encryption key is rotated at a predefined time epoch. The interval defines the granularity at which we can give access to data. After each key update, only the new K_{t+1} is shared with the subscribers, allowing us to keep the number of keys to be managed at the minimum. However, given s subscribers, this still incurs communication and computation overheads in $O(s)$: at each key update, the new key must be shared s times after encrypting it with each subscriber's public key (PK): $ENC_{PK_i}(K_{t+1}), i \in s$.

To reduce this overhead, we distribute the latest *data encryption key* K_t within a digitally signed and encrypted lockbox. Authorized subscribers obtain a long-term *distribution key KD* to open the lockbox. When sharing access to a data stream, we share the distribution key *KD* encrypted for the new subscriber within the ACL (as depicted in Figure 4.2): $ENC_{PK_i}(KD||\text{metadata}), i \in s$. Afterwards, each subscriber can use the distribution key to get the latest encryption key: $ENC_{KD}(K_t||t)$, where the counter t serves to identify the key. While the *data encryption key* is frequently updated at a defined interval, the distribution key KD is updated only in case of access revocations, as detailed later in this section. Note that for performance optimization, the latest lockbox can

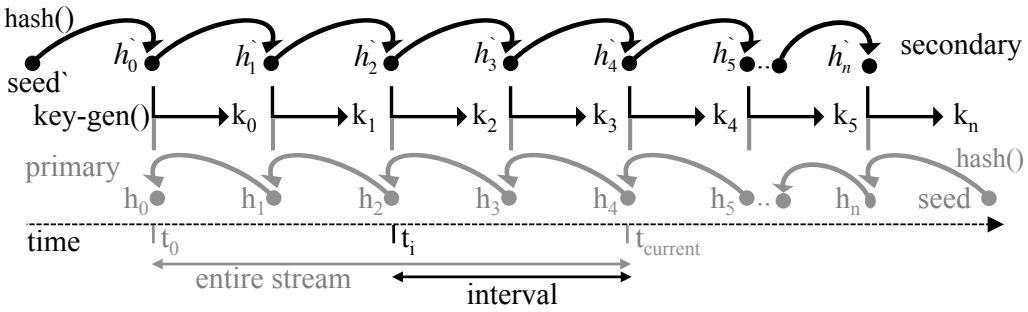


Figure 4.3: Our dual key regression supports time-bounded sharing, via a secondary hash chain. The gray elements depict the standard key regression mechanism. Given current k_c, one can compute all keys up to k_0. Our scheme allows the sharing of keys for a time-bounded interval via a secondary hash chain.

by piggybacked to each data retrieval.

Revocation. Revocation can either take place implicitly, as the permissions expire or explicitly, by the data owner issuing a new transaction overwriting previous permissions. To revoke access to a data stream, the data owner updates the distribution key and issues a new transaction with a state transition that evicts the revoked service. The transaction includes the new distribution key KD' contained in the encrypted key information per subscriber. Hereafter, the new data encryption key K_{t+1} is only available to the remaining authorized subscribers, protected with the new distribution key: $ENC_{KD'}(K_{t+1}||t+1)$.

With the newly issued blockchain transaction, the global access permission state is updated. In our access control model, Droplet cryptographically prevents any future access to new data by the evicted subscriber. Revoking access to old data is however difficult, as the user might have cached the data anyway. Nevertheless, in this case, the storage nodes decline future access requests by the corresponding subscriber.

Time-bounded Subscription. A limitation of key-regression-based sharing is that it enables sharing from the beginning until *current time i* (i.e., all-or-nothing principle). We design a key management mechanism that enables sharing in subscription mode with a defined lower time bound, e.g., access to data of a particular stream from *Jan'18* till revoked. To realize this, we extend key regression with an additional hash chain in the reverse order, to cryptographically enforce both boundaries of the shared interval, as depicted in Figure 4.3. We refer to this scheme as *dual key regression*.

In the standard key-regression, hash tokens are consumed in the reverse order of chain generation as input to a key derivation function to derive the current key. Due to the pre-image resistance property of hash functions, it is computationally hard to compute future tokens and hence future keys (i.e., given token h_i it is intractable to find token h_{i+1}). However, the reverse is performed efficiently, allowing the derivation of past keys. We leverage this property of hash chains for defining the beginning of an interval through a secondary hash chain in the reverse order, as depicted in Figure 4.3.

In dual key regression, the key derivation function (KDF) takes an additional hash token h'_i to derive a key: $KDF(h_i||h'_i) = K_i$, where h'_i comes from a secondary hash chain as depicted in Figure 4.3. For instance, to share a data stream from time t_i to t_j, the user provides the tokens h'_i and h_j. Since it is infeasible to compute h_{j+1}, no key posterior to k_j can be computed. Conversely, since it is infeasible to compute h'_{i-1}, no key prior to k_i can be computed. With access to the two hash tokens (h_j, h'_i), indicating the beginning and end of the shared interval, one can compute all the encryption keys within this interval. Note that the time epoch of key update defines the granularity of the interval sharing (e.g., hourly, daily, or weekly).

For upcoming key updates, the lockbox still contains only the updated token h_{j+1} from the main chain, while the individual secondary token h'_i for the start of the interval remains unchanged. A subscriber computes the current key K_{j+1} given the current token h_{j+1} and the start token h'_i as follows:

$$KDF(h_{j+1}||h'_{j+1}) = K_{j+1}, \text{with } \mathrm{H}^{(j-i+1)}(h'_i) = h'_{j+1} \tag{4.5}$$

with H as a hash function. The secondary token, indicating the start of the interval is hence stored along the long-term key information per reader in the ACL, as shown in Figure 4.2.

Compound Key Management. Our construction so far, while consistent with our low overhead requirements, lacks in expressiveness for sharing past data. For instance, dual key regression does not allow sharing of multiple disjoint intervals within the same stream. We now explain the role of binary hash trees (BHT) and describe our key management in its entirety. A BHT is constructed top-down with an initial secret seed at the root with a hash function $H_l()$ and $H_r()$, for the left and right child nodes, respectively. Each parent serves as input to the corresponding hash function to compute the child node. BHTs are similar to hash chains in that due to the preimage resistance of crypto hash functions, it is computationally intractable to find the parent of a given child node, while the reverse is efficiently computable.

In Droplet, we first construct a binary hash tree of depth d given a

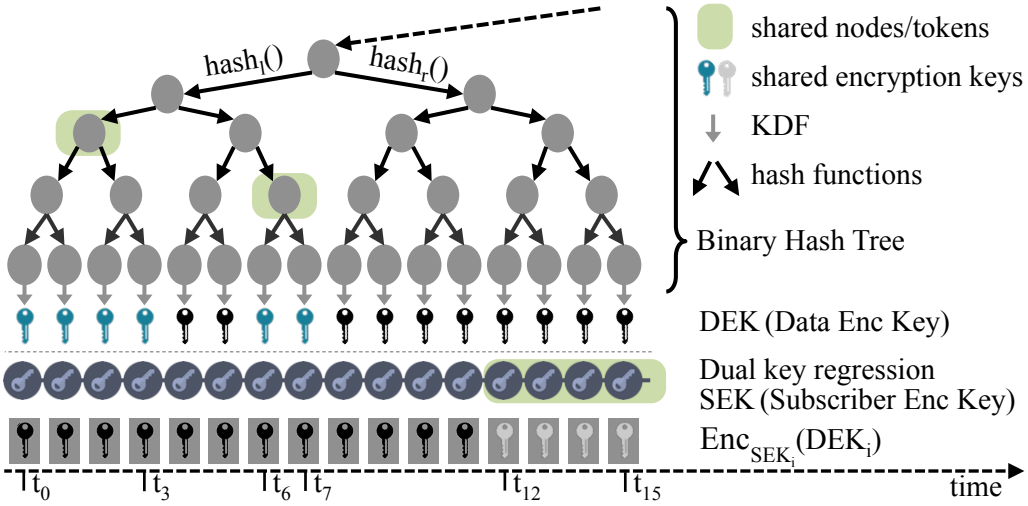


Figure 4.4: Droplet's hybrid key management system supports sharing of both arbitrary intervals and subscriptions. While DEKs are managed through the binary hash tree structure, SEKs are managed through dual key regression.

seed as the root. We realize the two hash functions by concatenating 0 or 1 to the input for the left and right hash functions, respectively. The leaf nodes deliver the encryption keys (via KDF), as depicted in the upper part of Figure 4.4. To share any arbitrary interval, the data owner includes the nodes necessary to compute the corresponding keys within the encrypted key information of the ACL. Note that a hash-tree-based key management is not suitable for subscriptions, as it requires managing per subscriber state which is not inline with our low-overhead requirement concerning distributing keys.

To overcome this challenge, we combine dual key regression with a hash tree construction via a layered encryption technique. Conceptually, we exploit the hash tree to allow arbitrary sharing of intervals and the dual key regression to support sharing in subscription mode. The layered encryption consists of three steps: *(i)* the hash tree delivers data encryption keys DEK_i which we use to encrypt data during the time epoch i. *(ii)* the dual key regression delivers subscriber encryption keys SEK_i of the same epoch i. We use SEK_i to encrypt the corresponding data encryption key: $ENC_{SEK_i}(DEK_i)$. *(iii)* each encrypted data chunk holds the encrypted *DEK*.

Note that we can give access to data encryption keys either via the hash chain (arbitrary intervals) or dual key regression (subscription), as depicted in Figure 4.4. To a subscriber, *DEK*s appear as random encryption keys per epoch. For principals with access to past data, *DEK*s

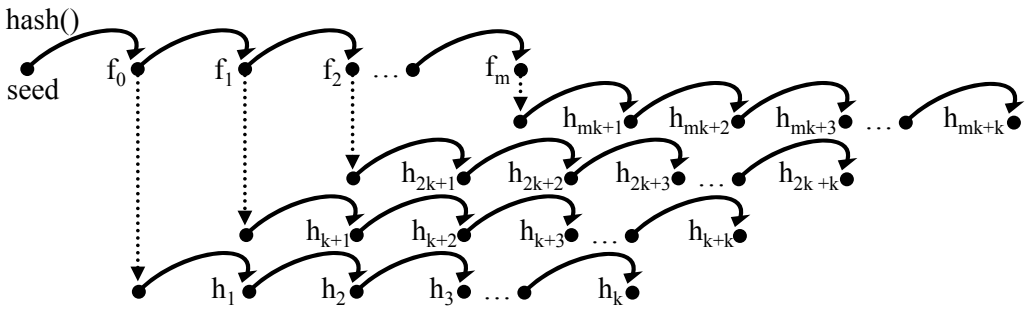


Figure 4.5: Our hash chain construction requires lower memory and computation time compared to standard hash chains. m and k are equal to $\sqrt{n}-1$ and $\sqrt{n}$ of the total n hash tokens, respectively. The worst case computation time is reduced from $O(n)$ to $O(\sqrt{n})$.

are the leaf nodes of the hash tree which they locally compute based on the shared parent nodes of the corresponding subtrees. Note that a principal can assume both roles, as illustrated in the example of Figure 4.4. In this example, the data owner has granted the principal access to the intervals $t_{[0-3]}$ and $t_{[6-7]}$, where access to the corresponding data encryption keys is realized through the hash tree. Additionally, the principal is granted access to a subscription from t_{12} which is realized over dual key regression. We describe next how to handle long key chains efficiently.

Compact Hash Chains. Our key management scheme, specifically dual key regression, relies heavily on hash chains. The underlying chains can grow quickly due to frequent key updates. Hash chains are computed based on a random secret seed. They can be either stored locally or re-computed on demand for each key update. Due to memory-constraints of IoT devices, a combination of re-computing on demand and storing a segment of the hash chain is desirable, to achieve fast and efficient key rotations. We leverage hierarchical hash chains [115] which maintain the same security features as traditional hash chains but reduces the worst case compute time from $O(n)$ to $O(\sqrt{n})$ computation.

In a nutshell, we divide the hash chain into segments, enabling efficient access (i.e., computation-wise) to individual segments. In this construction, a main hash chain with $\sqrt{n}$ nodes is first computed, as illustrated in Figure 4.5. Each token of the main hash chain, marked as f_i, serves as a seed of a side chain. Each side chain is $\sqrt{n}$ long. The tokens of the side chain are used as input to the key regression. The advantage of our construction is that the current side chain in use can be re-computed in $O(\sqrt{n})$. For key regression, this comes at the cost of additionally sharing

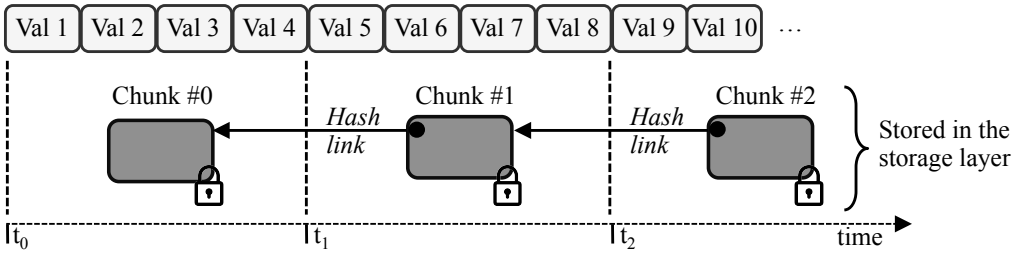


Figure 4.6: Data streams are chunked, compressed, and encrypted at defined intervals. To lookup a record, the timestamp of the record is mapped to the chunk identifier.

the seed of the last used side channel (the token from the main chain). For instance, in the given example of Figure 4.5, the client starts with the first side chain: tokens h_{mk+k} up to h_{mk+1}. Note that, as the case with hash chains, we use them in the reverse order, such that given h_i it is intractable to find h_{i-1}. After consuming all k tokens from the first side chain, we switch to the next side chain $h_{(m-1)k+k}$ to $h_{(m-1)k+1}$. From this point onwards, the client shares f_m from the main chain in addition to the current side chain token. This allows the computation of already disclosed main chain tokens, and hence all previously disclosed side chains (this corresponds to computing old keys in key regression). In our evaluation discussions in Section 4.5.1, we show how compact chains allow for a two-orders of magnitude key rotation speed-up.

4.3.2 Data Plane

Droplet's design is agnostic of the physical storage utilized in the data plane. Data is however protected from storage services with client-side encryption. The key management and data sharing model in Droplet are tailored for time-series data, specifically in the IoT space. We now discuss data serialization, namely, chunking and encrypting of data records. Later, we detail our search mechanism for encrypted data chunks, our optional data immutability feature, and storage nodes role in Droplet.

4.3.2.1 Data Serialization

Droplet focuses on time-series data with a stream nature, where data records are generated continuously, as depicted in Figure 4.6. The key element of Droplet's data model is data chunking which is a common technique for time-series data [101, 75, 7]. Instead of storing individual data records, we store data chunks, which are an ordered batch of data

records of an arbitrary type (i.e., pairs of timestamp/value). Each chunk contains as well a hash link to the previous chunk, enabling absolute time-ordering. Although chunking prevents random access at the record level, there is a positive gain on the performance of data retrieval since in time-series data most queries involve data that are co-located in time [101]. E.g., data analytic apps work with temporal data records (e.g., all records of a day).

Encryption. Each data chunk is initially compressed and then encrypted at the source with an efficient symmetric cipher. We rely on AES-GCM, as an authenticated encryption scheme. Note that NIST bounds the use of AES-GCM to 2^{32} encryptions for a given key/nonce pair. Due to our frequent key rotations, we stay far below this threshold. The chunks have a metadata segment containing, among others, the chunk identifier, the owner's address, the key version, the encrypted key (Section 4.3.1.3), hashes to previous chunks (Section 4.3.2.3), and the stream identifier. The data field contains the encrypted and compressed data records. We employ authenticated encryption to protect the integrity of both the plaintext metadata and ciphertext, and to authenticate them. Services with access to the encryption key can verify the integrity of the chunk and perform an authenticated decryption. To ensure data ownership, for instance towards the storage layer, each chunk is also digitally signed. This allows parties without access to the encryption key to still be able to verify the owner of the data stream, albeit at a higher computation cost. In general, digital signature operations are three orders of magnitude slower than symmetric key operations, as discussed in Section 4.5.1. Hence, authenticated encryption in our design allows a higher read throughput for authorized subscribers with access to the symmetric encryption key.

Compression. IoT data is highly compressible, as it exhibits a low entropy from a limited range. However, individual record compression has a low yield. Hence, we compress data chunks before encryption. This reduces bandwidth and storage requirements significantly. In Droplet, we employ `zlib` as the compression algorithm, except for images, where we utilize the state-of-the-art compression algorithm Lepton [113].

4.3.2.2 Search

The storage layer in Droplet resembles a key-value store (in our case UUID-chunk pairs). We define the universally unique identifier (UUID) as the cryptographic hash of the tuple: `<owner address, streamID, #counter>`, where `streamID` is a unique identifier of an owner's various data streams. Since the values are comprised of encrypted data chunks and will not allow any indexing, we need to devise a mechanism to

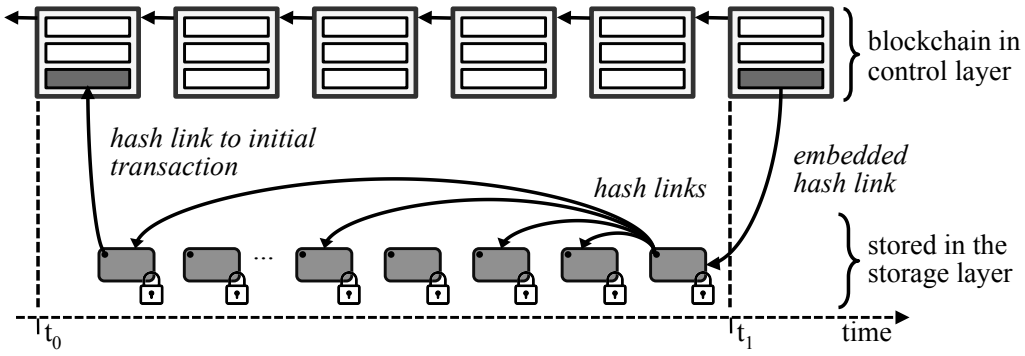


Figure 4.7: Immutable chunks after a defined grace period. After the grace period, the hash of the latest chunk is included in the blockchain. Each chunk contains hash links to previous data chunks, forming a geometric series. Hence, one can verify the integrity of all data chunks within the grace period.

perform temporal range queries efficiently. To avoid consistency issues of a shared index, we exploit a simple local lookup mechanism to enable temporal range queries. For a constant lookup time of a record with timestamp t_i, we compute the counter of the chunk holding it based on the known time interval Δ of the chunks: $\lfloor (t_i - t_0)/\Delta \rfloor$. For instance, Droplet maps the lookup of value 7 in Figure 4.6 to the identifier of chunk #1. The chunk metadata is included in the stream registering transaction, as depicted in Figure 4.2. Note that the chunk metadata additionally enables freshness checks for chunks, since the chunk interval indicates the frequency and time at which new data chunks are generated. The data owner can additionally include a dedicated freshness window, to tolerate for common delays.

4.3.2.3 Strong Immutability.

While Droplet provides integrity protection via authenticated encryption and digital signatures, the data owner can still modify old data. Specific applications might require a stronger notion of immutability such that even the data owner can no longer modify the data (e.g., contractual agreements in logistics). Droplet enables such a notion of immutability through blockchain's append-only property [43]. The application developer can define a grace period until data chunks become immutable. For sensitive applications, this can be per chunk. Otherwise, a longer period can be selected. To accommodate for the narrow bandwidth of blockchains, we leverage an anchoring technique, where data immutability transactions are reduced to the level of the grace

period. To realize this, the first data chunk holds a pointer to the registration transaction and after the grace period another transaction with a pointer to the latest chunk is issued, as depicted in Figure 4.7. Since all data chunks are cryptographically linked via hashes, all data chunks in the grace period become immutable at once, forming a chain of data chunks. To avoid a linear verification time, chunks hold hashes to several previous chunks, forming a geometric series. This enables a logarithmic verification time.

4.3.2.4 Storage Interface.

Droplet's design supports various storage modalities, e.g., edge, decentralized, or cloud storages. The storage node exposes a simple *store/get* interface with various flavors of *get*, such as *getAll* or *getRange*. For each request, the storage node verifies the identity of the client and looks up the corresponding access permissions regarding the client's identity, i.e., public key.

4.3.3 Privacy and Security Analysis

Control Plane. For an adversary to alter access permissions in the blockchain, it requires forging a digital signature (i.e., breaking public key cryptography with 128-bit security level) or gaining control over the majority of the computing power in the blockchain network (i.e., 51% attack). Moreover, an adversary is not capable of learning sensitive information from the public blockchain, since only unlinkable pseudo-identities and stream identifiers are stored there. In profiling attacks, the adversary creates profiles of all user identifiers and the network of users [161]. This would allow the adversary to break the pseudonymity of specific users. To protect Droplet against profiling attacks, we employ dual-key stealth addresses, where the anonymity set is equal to the set of users with stealth addresses. Note that if dual-key stealth addresses are used for signing transactions, additional measures are required to to maintain the anonymity. A malicious storage node could hand out data without permission. However, the impact of this action is limited since data is encrypted. The blockchain provides auditable information about when a stream was shared with whom; a crucial piece of information to prove or disprove access right violations should the need arise.

Data Plane. Data chunks are encrypted, integrity protected, and authenticated. Any data chunk manipulations are detectable with the digital signature and authenticated encryption. The optional data immutability is based on the security of blockchain. The secure channel

(i.e., TLS) for storing and fetching data prevents replay attacks, in addition to ensuring an authenticated and confidential channel. An adversary with access to encryption keys cannot alter old chunks, as it requires access to the signing private key. Our *store* and *get* are susceptible to a denial of service attack.

4.4 Implementation

Our reference implementation of Droplet is composed of three entities implemented in Python: the client engine, the storage node engine, and the virtualchain. Droplet's client engine is in charge of composing a data stream and data serialization, i.e., chunking, compression, and encryption. It handles viewing, setting, and modifying access permissions via the virtualchain. It provides the interface to interact with the storage layer via TLS. The client engine is implemented in 1700 sloc. We utilize Pythons's `cryptography` library [191] for our crypto functions. For compression, we use `Lepton` [60] for images and `zlib` [56] for all other value types.

The storage engine can either run on a centralized cloud or the nodes of a p2p storage network. Currently, we have integrated drivers for Amazon's S3 storage service. On individual nodes, we employ LevelDB, an efficient key-value database [148]. We have as well a realization of Droplet with a serverless computing platform with ASW Lambda serving as the interface to the storage (i.e., S3). Emerging serverless platforms such as Lambda [16], require request-level authorization [2]. In this setting, the client directly accesses backend resources (i.e., without a gatekeeper server) where each request needs to be separately authorized, which makes this setting an interesting case study for Droplet. Our serverless setup consists of AWS Lambda serving as the interface to the storage (in our setup S3). Once Lambda is invoked, it performs a lookup in the access control state machine to process the request. For comparison, we implement as well an OAuth2-based authorization based on AWS Cognito [15] as the identity provider. For the distributed storage, we build a DHT-based storage network. We instantiate a Kademlia library [192] and extend it with the security features of S/Kademlia [22]. The Kademlia protocol runs an asynchronous JSON/RPC over UDP. We add support for TCP connections for storing/fetching large chunks. Our extensions amount to 2400 sloc.

The virtualchain is instantiated from Blockstack [34] and extended to implement our access control state machine. The virtualchain scans the blockchain, filters relevant transactions, validates the encoded operations,

	AES Encrypt		SHA Hash		ECDSA Sign	
	[μs]	[op/s]	[μs]	[op/s]	[ms]	[op/s]
IoT SW	298	3.4k	297	3.4k	270	3.7
IoT HW	42	23.8k	17	58.8k	174	5.7
Phone	50	20k	45	22.2k	4.4	227
Laptop	5.4	185k	1.6	623k	1.3	770
Cloud	2.6	384k	1.2	833k	1.1	909

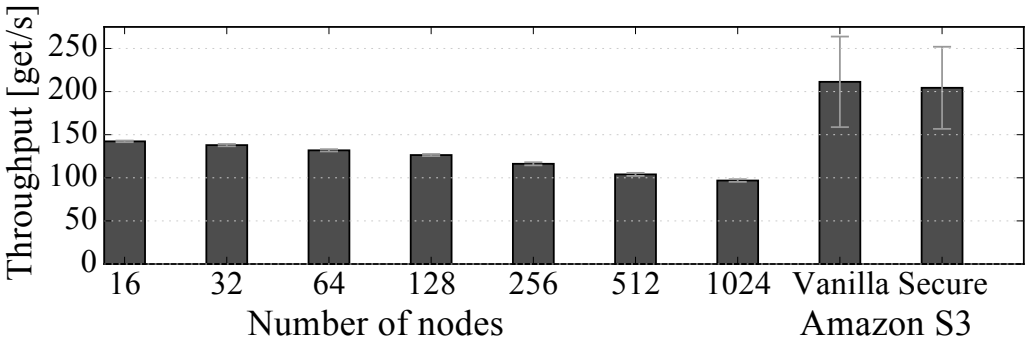
Table 4.2: Performance of security operations with 128-bit security on different platforms (i.e., AES256, SHA256, ECDSA256). The IoT is represented by the OpenMote microcontroller with software (SW) computations or crypto hardware accelerator (HW). For smartphone, we use a Nexus 5 and for Laptop we use Macbook Pro. The cloud is represented by an Amazon t2.micro instance.

and applies the outcome to the global state. The state is persisted in an SQLite database. The global state can either be queried through a REST API or accessed directly through the SQLite database. Our extensions to the virtualchain amount to 1400 sloc. As the underlying blockchain, we employ a Bitcoin test-network with a low block generation time to emulate a hybrid consensus blockchain [143] (i.e., ca. 15 s block confirmation).

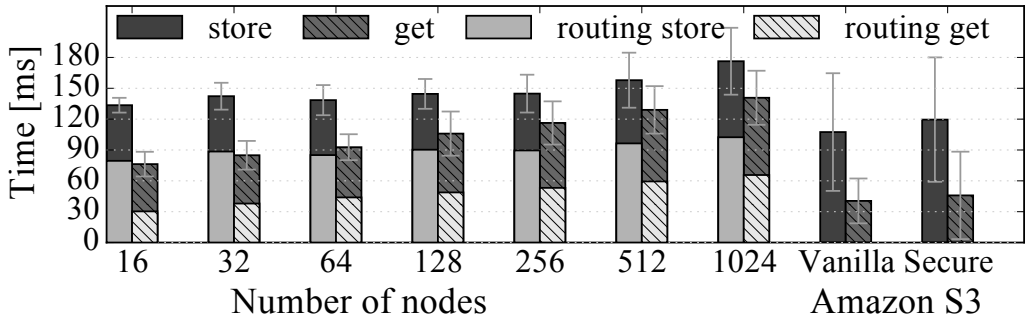
4.5 Evaluation

We now discuss the micro-benchmark evaluation of Droplet functionalities and then present the overall system performance in the end-to-end evaluation. We run and present the performance of Droplet on top of centralized and decentralized storage layers, i.e., Amazon's S3 and 1024 DHT nodes running in real time on an emulated network. Evaluating and prototyping Droplet within a decentralized storage setting is an interesting case, as peer-to-peer storage networks could become a viable solution for the IoT [242]. Additionally, this setup resembles storage-oriented blockchains (e.g., Storj [236], Filecoin [189]). These efforts still lack adequate mechanisms for secure data sharing, where Droplet can be helpful.

We additionally evaluate the performance of Droplet in a serverless setting and compare it to OAuth2 authorization. Emerging serverless platforms such as Lambda [16], require request-level authorization [2]. Hence, this is particularly an insightful setting as Droplet serves as an independent *Authorization as a Service*, which can be particularly useful



(a) Average throughput for *get*.



(b) Latency for single *store* and *get* requests.

Figure 4.8: *store*/*get* performance for centralized (Vanilla: w/o Droplet, Secure: with) and decentralized storage layers (DHT). The distributed storage latency is dominated by network routing. For fairness, all settings, including Vanilla S3 operate on compressed data chunks.

to the *Function as a Service* (FaaS) paradigm. Our control plane runs on a Bitcoin test network which has similar properties to the main Bitcoin network, except we tune the network to emulate the performances of recent scalable blockchains [143] (i.e., ca. 15 s block confirmation).

Methodology. Throughout the evaluation, we are interested in quantifying the overhead of Droplet and measuring its performance. For this, we rely on the following metrics: *(i)* computation: indicates the CPU time required to perform a certain operation. It has a direct impact on the application delay and energy consumption. *(ii)* latency: reflects the time from a client initiating a *store* or *get* to the client receiving the confirmation or result, and is typically dominated by network round trip time. *(iii)* throughput: reflects the number of reads and writes per unit of time. Note that in evaluation, we do not cache any data, to emulate the worst case scenarios. For each experiment, we show the mean and distribution (i.e., standard deviation) or Tukey box plot of 100 iterations.

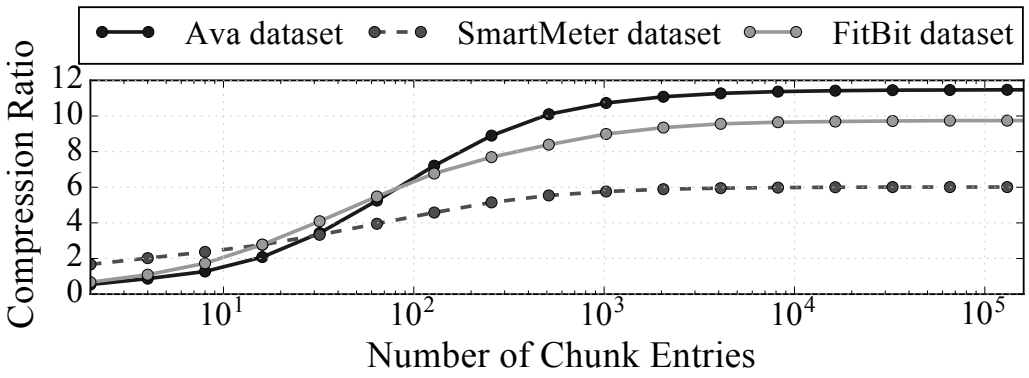


Figure 4.9: Compression ratio of our chunking for Fitbit, Ava, and smart energy dataset.

Setup. The serverless setting is composed of Lambda, S3 storage, and AWS Cognito in case of the OAuth2 baseline benchmark. Our setup for the decentralized storage consists of a memory-optimized instance of Amazon's EC2 (r3.4xlarge, 122 GiB, 16 vCPU), where we run up to 1024 instances of storage nodes. We use `netem` [173] to emulate a network roundtrip time of 20 ms between the instances of storage nodes. We use one instance of Amazon's S3 storage service (a round trip time of 20 ms) for the centralized storage scenario. Both our S3 and EC2 instances are located in central Europe (Frankfurt). For the crypto operations, we use four classes of devices: *(i)* IoT device: OpenMotes microcontroller are equipped with 32-bit ARM Cortex- M3 SoC at 32 MHz, with a public-key crypto accelerator running up to 250 MHz. Fitbit trackers utilize a similar class of microcontroller; *(ii)* smartphone: LG Nexus5 equipped with a 2.3 GHz quad-core 64-bit CPU and 2 GiB RAM; *(iii)* laptop: MacBook Pro equipped with 2.2 GHz Intel Core i7 and 8 GiB RAM; *(iv)* Cloud VM: EC2 t2.micro (1 vCPU, 1 GiB RAM).

Datasets We validate Droplet on three datasets and quantify the end-to-end overhead: *(i)* for the Fitbit activity tracker, we use one of the co-author's data from Fitbit Charge HR for one year (16 data types, 130 MB). *(ii)* for the Ava health tracker, we use an anonymized dataset for Ava (10 s intervals, 13 sensors, 1.3 GB). *(iii)* for the ECOviz smart meter dashboard, we use the ECO dataset (1.85 GB) for 6 Swiss households over a period of 8 months [24] (1 Hz accuracy).

4.5.1 Micro-Benchmark

We instrument the client engine to perform the micro benchmark operations in isolation with up to 1000 repetitions.

Compression Ratio. We study the compression ratio as a function of chunk size, for three different datasets. In all three cases, a data stream includes more than 10 types of sensor data. As depicted in Figure 4.9, almost for all three cases, we reach the optimum compression ratio already with a chunk size of 2000 records. This corresponds to chunk intervals of about one day for Ava and one hour for energy data. However, as our Fitbit data is already aggregated by the service, a chunk interval of one day only corresponds to about 100 records (60% of the optimum compression ratio). The resulting trade-off is between granularity and size-efficiency.

Cryptographic Operations Table 4.2 summarizes the costs of the crypto operations involved in Droplet on four different platforms. All these operations, namely AES encryption, SHA hash, and ECDSA signature are performed once per chunk for store requests. For data retrieval, the client does not perform a signature verification, since AES-GCM has built-in authentication. Running the crypto operations only in software on the IoT devices shows the highest cost, with 3.4k encryptions/hashes per sec and only 3.7 signatures per sec. With the onboard hardware crypto, the cost of AES and SHA is improved by one order of magnitude and approaches that of smartphones. Note that overall signatures are three orders of magnitude slower than symmetric key operations.

Dual Key Regression Hash computations are the basis for dual key regression. The computation takes place at the initial setup and each key update if the client chooses to re-compute keys on-demand rather than store them. This is a practical solution to handle long chains in constrained devices. Assuming a long chain of length 9000 (hourly key updates for one year), it takes 405 ms to compute the entire chain on smartphones and 2.7 s on an IoT device without hardware crypto engine, as depicted in Figure 4.10. With the compact hash chain technique, we reduce this worst case compute time to 4.3 and 28.2 ms, respectively. The savings become pronounced with smaller epoch intervals.

4.5.2 System Performance.

To model the real-world performance of Droplet, we constructed an end-to-end system setup, where we use our three app datasets. Note that we do not cache any data to emulate worst case scenarios. The chunk size of the data stream is set to 8 KiB. We evaluate `get` and `store` requests to the

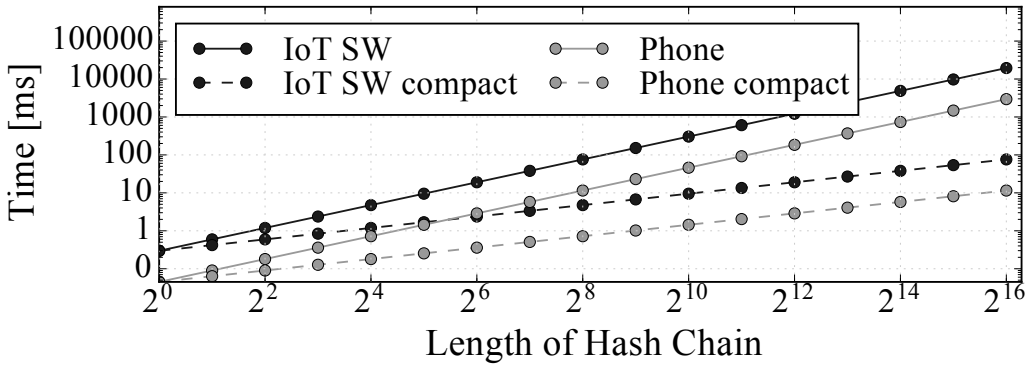


Figure 4.10: Impact of compact chains on the compute time. The $O(\sqrt{(n)})$ optimized computation time of hash chains of length n benefits computationally constraint devices significantly.

storage layer, which include the overhead of Droplet's access control.

Serverless Computing In the serverless setting, Lambda either runs Droplet for the access control or uses the AWS Cognito service, which runs OAuth2, as the baseline. Lambda with both Droplet and Cognito exhibits a latency of around 118 ms (0.4% longer with Droplet). Note that with OAuth2, to reach the same level of access granularity as with Droplet, separate access tokens are required for each data chunk, which is impractical. This is why in practice long-lived and more broadly-scoped access tokens are granted.

Cloud. We extend AWS S3 storage with Droplet and compare its performance against vanilla S3. Figure 4.8(a) shows the throughput for different request types. We follow Amazon's guidelines to maximize throughput: for instance, the chunk names are inherently well distributed allowing the best performance of the underlying hash-table lookup. The vanilla S3 throughput of 211 gets/s is within Amazon's optimal range (100-300). With Droplet, we maintain an average rate of 204 get/s (3 % drop). Figure 4.8(b) shows the latency for individual *store* and *get* operations. In Droplet, the latency overhead is 13% for *get* and 11% for *store* (incl. crypto operations). Part of the overhead is due to the expensive signature operation. Additionally, there is an overhead for a fresh lookup of access permissions at the access control DB of the virtualchain instance. The access permissions are only cached for 1 minute.

Distributed Storage. We measure the performance of *get* and *store* requests on a secure DHT with Droplet, with varying network sizes, from 16 to 1024 nodes. Figure 4.8(a) shows the throughput results.

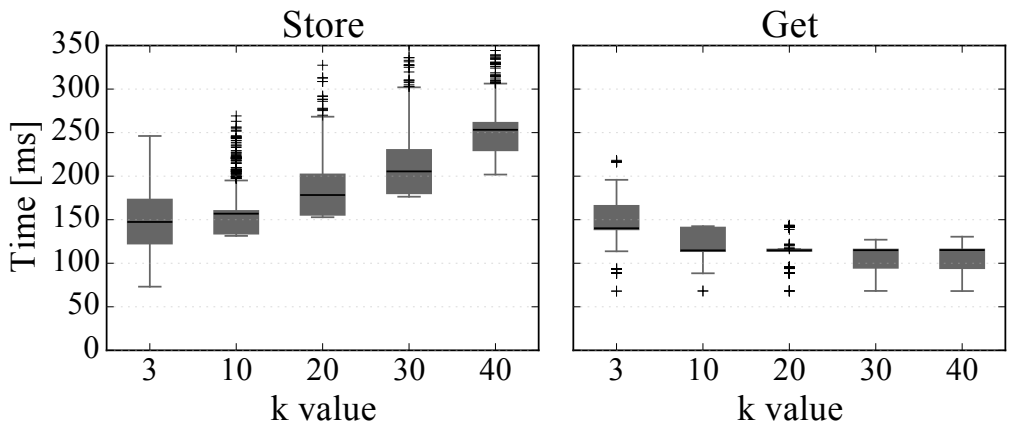


Figure 4.11: The impact of degree of replications (k) with 512 DHT nodes. A higher k results in faster retrieval time, as more nodes are likely to hold the data. The inverse effect is the increased time for storing chunks.

As the number of nodes increases from 16 to 1024, the performance decreases from 142 to 96 get/s. Figure 4.8(b) shows the latency results, divided into routing and retrieval. The total *get* latency increases from 76 to 140 ms as the number of nodes grows. This is about 3 times slower than S3's centralized storage. However, note that this slowdown is dominated by the routing cost. After resolving the address of the storage node with the data chunk, the secure retrieval time is similar to that of S3. Also, note that *get* requests have a lower routing overhead than store requests. This is because for get requests, the routing process is aborted as soon as a node holding the data chunk is found.

An important factor defining the performance of store and get is the initial replication factor k. Figure 4.11 shows the latency of get and store as a function of k. Notice how the store latency increases with k, while the get latency decreases. We set k to 10 in our experiments, as it provides a reasonable trade-off.

Applications. For our three applications, we measure the overhead of *store* and *get* for different views in the app running on top of Droplet. At the storage layer, we discuss here the case of the decentralized storage setting with 1024 nodes. Fitbit and Ava rely on a smartphone to store their data. Due to memory constraints, data synchronization should take place at least weekly for Fitbit and daily for Ava. This results in an average *store* latency of 176 ms and 1.2 s for Fitbit and Ava, respectively. Note that store operations runs in the background. For different views, the maximum *get* latency is below 150 ms. Hence, the user experience remains unaltered.

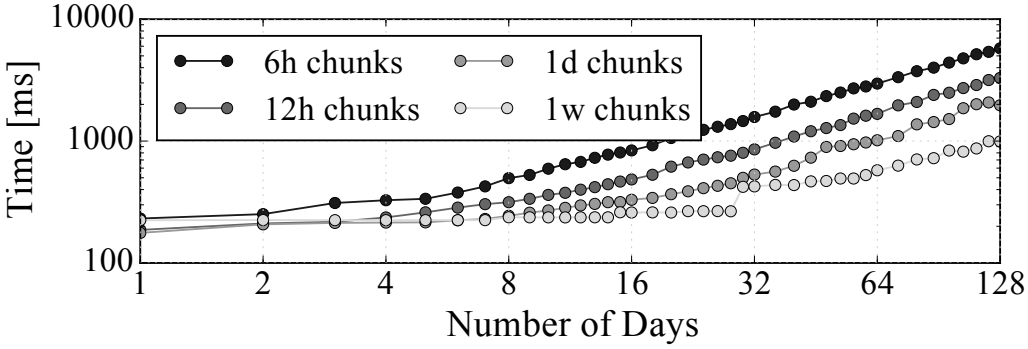


Figure 4.12: EccoViz app results. Retrieving records from the energy dataset in the EccoViz dashboard app (peer-to-peer storage).

In contrast to Fitbit and Ava, the smart meter node has direct Internet connectivity. Instead of synchronizing periodically, it stores chunks immediately after generation. This takes 176 ms per chunk. The most comprehensive view in the ECOViz dashboard can visualize the entire data stream. Figure 4.12 shows the latency to fetch chunks dependent on the number of days requested. Fetching data for 128 days of 6 h chunk size requires about 10 s, whereas the one-week chunk size requires less than 1 s.

Blockchain. In Droplet, we inherit the security properties [86], as well as the limitations of the underlying blockchain. Consequently, the performance of Droplet's control plane is bound to that of the underlying blockchain. In our prototype, the transaction confirmation time is set to 15 s, similar to that of Ethereum. The slow blockchain writes have a direct impact on the time until new access permissions take effect, which is significantly higher compared to OAuth2 protocol. Read throughput is, however, fast and comparable to that of OAuth2. Data stream registrations and access permission adjustments (e.g., grant/revoke access) require transactions writes. To scale Droplet to data streams in the order of billions, a blockchain throughput in the order of thousands of transactions per sec is necessary. While currently deployed blockchains achieve only a fraction of this throughput, next-generation blockchains could soon close this gap [143].

4.6 Related Work

In Section 4.2.1, we discussed relevant work in distributed and crypto-based access control in length. Hence, we keep our review here brief.

Distributed Access Control. In the OAuth2 protocol [153], a trusted entity issues tokens that allow a principal to access data. Macaroons [31] build upon pure bearer tokens, using chained HMACs, enabling delegation and attenuation of capabilities for access scope. Such tokens, however, require a trusted intermediary to verify a principals authorization, whereas Droplet handles the authorization itself. SPKI/SDSI [66] is a certificate-based scheme which requires either certification authorities or works only in an isolated deployment. DelegaTEE [157] introduces the concept of brokered delegation using Trusted Execution Environments (TEE). DelegaTEE leverages TEEs, e.g., Intel SGX, to handle and manage access delegations.

Crypto-enforced Access Control. A large body of cryptographic research [80] focuses on private access control on untrusted cloud providers, such as (hierarchical) identity- based encryption, attribute-based encryption (ABE), predicate encryption, and functional encryption. ABE [197, 179, 96, 28, 97] is the most expressive among these schemes. However, it comes with limitations with respect to revocation, fine-grained access, and dynamic access updates [80] (Section 4.2.1). Sieve [234] combines ABE with key-homomorphic encryption, to enable revocation, where the cloud is trusted not to collude with adversaries to disclose the new key or launch a rollback attack. ABE-based schemes target file-based storage and fall short in providing fine-grained access for time-series data. Moreover, the storage provider has no means to verify the access request, as access can be only verified with a decryption test. Droplet does not suffer from these limitations and additionally enables unique features such as immutability and audibility.

Untrusted Servers. The SFS read-only file system [79, 77] is among the first efforts to introduce efficient secure content distribution using untrusted servers. They introduce the concept of key regression as a means of access control, where the data publisher is contacted for the latest keys. Tutamen [201] proposes a distributed and federated access control system, with a set of trusted servers in charge of key management. GORAM [156] introduces an ORAM primitive to enforces access-control restrictions on encrypted cloud data within a group of users. GORAM while effective in hiding access patterns, it imposes high overhead upon users (in $O(polylog(n))$), requires a-priori knowledge of maximum number of principals, and in general falls short in supporting efficient and concurrent access. Anonymized and controlled sharing of files in peer-to-peer networks [123] overcomes the challenge of untrusted servers, however, is not suited for sharing data streams.

Blockchain. In recent years, a new class of blockchain technologies

have emerged that utilize the accountable computing and auditability of blockchains for other domains. Blockstack [5] introduces the concept of virtualchains and proposes a decentralized server-less domain name registry. Storj [236] and FileCoin [189] introduce distributed object storage. They are both targeted for archiving files and currently lack sharing features. Enigma [245, 246] introduces a similar approach in leveraging the blockchain for access control and sharing of off-chain stored data. However, Enigma does not account for stream data and stores data access logs within the blockchain, without addressing the consequent scalability issues. Our approach is inspired by the above approaches, but our focus on access control and data streams leads to a number of important design differences.

4.7 Conclusion

In this chapter, we introduce the design of Droplet, a decentralized and auditable access control service, tailored for time-series data. We leverage a public blockchain to bootstrap the necessary trust in our system and eliminate the role of trusted intermediaries. Droplet enables fine-grained crypto-based data access and secure sharing of data streams. With our prototype implementation and experimental results, we show the feasibility and applicability of Droplet as a decentralized authorization service that operates without a centralized logic.

5

Conclusions and Outlook

Today's Internet of Things deployments are developed vertically within organizations which are in control of the collected data, with little to no transparency or guarantees on how they ensure the privacy and security of the collected data. The privacy implications of this paradigm are immense, especially as data leakages and misuse of private data plague numerous computing systems and have become commonplace. This can hurdle the adoption of new disruptive technologies (e.g., personalized healthcare) and has unforeseen societal implications (e.g., profiling, discrimination, extortion, misuse). While policymakers efforts to tackle these concerns with regulatory provisions (e.g., European Union's General Data Protection Regulation) are necessary to hold service providers accountable, we believe that designing and developing powerful alternative secure systems and technologies, which ensure the privacy of users data and their data ownership is inevitable. We think that practical secure systems, which do not compromise on user-experience nor functionality, will enable the transition towards a more secure and privacy-preserving Internet ecosystem. The mechanisms and system designs proposed in this dissertation provide essential building blocks contributing to the realization of practical secure systems with high degree of security and privacy guarantees, based on cryptographic techniques. To ultimately conquer all aspects of data privacy and security from a holistic point of view, we need to work beyond technical solutions on several fronts and across disciplines, such as societal awareness, technological awareness, human-computer interaction for usable security, new legislation and legal frameworks, and alternative economic models.

We conclude this dissertation with a summary of our contributions, a short discussion of potential research challenges for future work, and final remarks.

5.1 Contributions

In this thesis, we argued for systems that ensure users data ownership and control over their data. We leveraged cryptographic techniques to achieve our goals and showed that practical, secure systems with reasonable overheads are feasible. To support our argument, we introduced three system designs that contribute to the state of data privacy and security; **Talos**, **Pilatus**, and **Droplet**, where each propose novel design components, which we consider of independent interest.

Talos is our effort in embracing the recent advancements in encrypted data processing, and bringing these to the IoT. One major barrier to employing such cryptographic primitives on IoT devices is their resource constraints. IoT devices are inherently limited with regards to energy, memory, CPU, and bandwidth. This challenge is exacerbated by computationally heavy asymmetric-crypto-based schemes, such as additive homomorphic encryption schemes. With Talos, we focus on designing and developing an encrypted query processing system for the IoT, where we overcome the computational challenges with our proposed optimizations, rendering our system feasible for the IoT. At a higher level, Talos is a system that stores IoT data securely in a cloud database while still allowing query processing over the encrypted data. We enable this by encrypting IoT data with a set of cryptographic schemes such as order-preserving and partially homomorphic encryption. We tailor Talos to accommodate for the resource asymmetry of the IoT, particularly towards constrained IoT devices. Talos leverages a batching technique to utilize the large ciphertext size in the Paillier cryptosystem to boost the performance of crypto operations (e.g., encryption, decryption, homomorphic addition) due to the concept of Single Instruction, Multiple Data (SIMD), where for instance one single homomorphic addition is required for the entire batch. Additionally, Talos proposes EC-ElGamal as a viable and efficient alternative to the Paillier cryptosystem, where we leverage the baby-step-giant-step algorithm for efficient decryptions. With a thorough evaluation of our prototype implementation, we show that Talos is a practical, secure system that can provide a high level of security with reasonable overhead.

Pilatus features a novel encrypted data sharing scheme based on re-encryption, with revocation capabilities and in situ key-update. Our key revocation mechanism allows users to terminate their data sharing

at any time. The in situ key-update at the cloud, protects as well old data with the owner's new key, without trusting the cloud with any private keys. Our solution includes a suite of novel techniques that enable efficient partially homomorphic encryption, decryption, and sharing. We present performance optimizations that render these cryptographic tools practical for mobile platforms. More specifically, we enable the performance optimizations with the Chinese Remainder Theorem, which leads to shorter, faster, and parallel computations. Our optimizations achieve a performance gain within one order of magnitude compared to state-of-the-art realizations.

Droplet enables data owners to securely and selectively share their data, while guaranteeing data confidentiality against unauthorized parties. Droplet leverages the blockchain technology to bootstrap trust, for our decentralized, secure, and resilient access control management. Droplet handles time-series data and features a cryptographically-enforced fine-grained and scalable access control mechanism for encrypted data streams. We design a hybrid low-overhead key management scheme based on hash trees and key regression, to support sharing of arbitrary intervals and open-end subscriptions. With a prototype implementation of Droplet on a public blockchain, we quantify Droplet's overhead and compare it to the state-of-the-art systems. When deploying Droplet with Amazon's S3 as a storage layer (a popular cloud service), we experience a slowdown of only 3% in request throughput. Moreover, we show the potential of Droplet as authorization service for the serverless computing domain, which requires request-level authorization.

5.2 Future Work

We now provide insights into interesting potential future work in the field of data security and privacy related to our work.

Private Statistical Queries on Encrypted Data Streams. With Droplet, we support client-side encrypted data storage on third-party services, where only user's devices and authorized parties have access to encryption keys and can perform encryption/decryption operations. While with our simple temporal queries, we support a wide range of interaction patterns specific to times series data, it is desirable to support more rich queries, such as private statistical queries on encrypted data, without accessing raw data.

This is motivated by the pattern, that is often observed while interacting with time series data; the queries are concerned with statistical values, e.g., the average of a certain metric during a defined time window.

This has inspired recent efforts in times-series databases [7], to construct a time-partitioned tree-based index over large volumes of raw time-series data. The index, since smaller in size compared to the raw data, fits in memory and enables fast processing of statistical queries. A potential research challenge is to bring private statistical analytics for encrypted time-series data, which would render Droplet capable of processing private statistical queries.

Data Quality and Data Authenticity. While in the threat model of our thesis, we assumed data source devices to be trustworthy and produce only correct values, it is an interesting challenge to relax this model and ensure the correctness and quality of data values. Data quality and authenticity are crucial for service correctness. Quality in this setting refers to the data record correctly representing the real-world event it represents, i.e., the sensor is not malfunctioning. Authenticity in this setting refers to the data record being indeed produced by trusted sensor hardware and not artificially crafted. This allows preventing a rough person contributing with incorrect sensor records to a dataset. It would be desirable that certified sensing hardware attest to the quality and authenticity of data records. Such assurances would play an essential role, not only in public datasets, but as well in critical applications, such as private digital healthcare, where devices, e.g., wearables, generate large volumes of data. The trustworthy sensing efforts [89, 88, 200] indicate the key role of trusted execution environments which are becoming pervasive in a wide range of platforms. Integration of these efforts intro encrypted data processing systems is a relevant future research avenue.

Additionally, for datasets it is possible to define serval data validation rules to ensure the data quality, such as data-type, range and constraint, and structured validations. Realizing such validations on encrypted data is a major research challenge.

Decentralized Large-Scale Storage. With the emergence of blockchain-based technologies, realizing decentralized large-scale storage has become one of the core envisioned services. The vision behind this service is allowing users to provide their unused storage to the network and in return be rewarded for the memory and bandwidth they contribute. Data providers can tune their storage strategies based on factors such as redundancy, the retrieval latency, and cost. The decentralized storage complements today's cloud-centric storage and could play a key role in the edge computing paradigm. Moreover, it creates a price and performance competitive alternative. Droplet provides essential insights as of how to enable secure sharing in such a decentralized storage setting. However, Droplet leaves out several challenges yet to be addressed in this space, such as the integration of the financial incentives into the

security model of access control. Moreover, decentralized storage offers many exciting research challenges, such as proof-of-replication to verify storage, locality-aware storage to enable a dynamic and fast retrieval time, and private computations on encrypted data stored in the network, to name only a few.

Practical Secure Multi-Party Computation. Secure Multi-Party Computation (MPC) [238] allows computation of private functions among a set of users without a trusted party. The individual values from participating users are kept confidential, while the outcome can be public, enabling many interesting interaction patterns. Despite MPC being a well-studied cryptographic field, its widespread adoption has been hindered by its current impracticality. With the emergence of egalitarian computing systems via blockchain technology there is a strong demand for MPC-based approaches, where the participants can be financially incentivized to perform the distributed computations. Designing and developing practical MPC solutions for the emerging blockchain technologies is an exiting research avenue.

5.3 Final Remarks

In this thesis, we introduced three system designs that innovate with powerful data security and privacy mechanisms. We leverage advanced cryptographic techniques to empower users with data ownership and control over their data. We show that it is feasible to achieve a higher level of data security and privacy, without sacrificing the user experience nor functionality. Our measures and designs create additional barriers for the adversary and assume other components of the ecosystem to be secure. This leaves many research challenges in this area to be yet addressed, some of which we have identified and discussed in the previous section, as future research directions.

We hope our findings and system design insights facilitate further research and system realizations towards a more secure and privacy-preserving digital world. Moreover, we hope the research outcome of this thesis contributes to shaping a roadmap to transform the way current systems store, process and interact with users data.

Bibliography

[1] Adrian Chen. GCreep: Google Engineer Stalked Teens, Spied on Chats, Gawker. Online: `http://gawker.com/5637234/gcreep-google-engineer-stalked-teens-spied-on-chats`, September 2010.

[2] Gojko Adzic and Robert Chatley. Serverless Computing: Economic and Architectural Impact. In *ACM Foundations of Software Engineering (FSE)*, pages 884–889, 2017.

[3] Shashank Agrawal and Melissa Chase. FAME: Fast Attribute-based Message Encryption. In *ACM Conference on Computer and Communications Security (CCS)*, pages 665–682, 2017.

[4] Muneeb Ali. *Trust-to-Trust Design of a new Internet*. PhD thesis, Princeton, 2017. Advisor: Andrea LaPaugh.

[5] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J. Freedman. Blockstack: A Global Naming and Storage System Secured by Blockchains. In *USENIX Annual Technical Conference (ATC)*, pages 181–194, 2016.

[6] Michael P Andersen and David E Culler. System Design Trade-Offs in a Next-Generation Embedded Wireless Platform. In *Technical Report No. University of California, Berkeley, EECS-2014-162*, 2014.

[7] Michael P. Andersen and David E. Culler. BTrDB: Optimizing Storage System Design for Timeseries Processing. In *USENIX Conference on File and Storage Technologies (FAST)*, pages 39–52, 2016.

[8] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating User Privacy in Bitcoin. In *Financial Cryptography and Data Security*, pages 34–51, 2013.

[9] Diego F. Aranha and Conrado P.L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. Online: `https://github.com/relic-toolkit/relic`, (accessed March 2, 2018).

[10] Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio Lopez. Faster Explicit Formulas for Computing Pairings over Ordinary Curves. In *Springer Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 48–68, 2011.

[11] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, and Ramarathnam Venkatesan. Orthogonal Security with Cipherbase. In *Conference on Innovative Data Systems Research (CIDR)*, 2013.

[12] Hassan Jameel Asghar, Luca Melis, Cyril Soldani, Emiliano De Cristofaro, Mohamed Ali Kaafar, and Laurent Mathy. SplitBox: Toward Efficient Private Network Function Virtualization. In *Workshop on Hot topics in Middleboxes and Network Function Virtualization (HotMiddlebox)*, pages 7–13, 2016.

[13] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30, 2006.

[14] Ava. Fertility Tracking Bracelet. Online: `https://www.avawomen.com/`, (accessed March 2, 2018).

[15] AWS Cognito. Online: `https://aws.amazon.com/cognito/`, (accessed March 2, 2018).

[16] AWS Lambda. Online: `https://aws.amazon.com/lambda/`, (accessed March 2, 2018).

[17] Sumeet Bajaj and Radu Sion. TrustedDB: A Trusted Hardware Based Database with Privacy and Data Confidentiality. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):752–765, 2014.

[18] Mario Ballano Barcena, Candid Wueest, and Hon Lau. How safe is your quantified self? Technical report, Symantec, August 2014.

[19] Paulo Barreto and Michael Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In *Springer Conference on Selected Areas in Cryptography (SAC)*, pages 319–331, 2005.

[20] Liliana Barrios and Wilhelm Kleiminger. The Comfstat - Automatically Sensing Thermal Comfort for Smart Thermostats. In *IEEE Conference on Pervasive Computing and Communications (PerCom)*, pages 257–266, 2017.

[21] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding Applications from an Untrusted Cloud with Haven. *ACM Transactions on Computer Systems (TOCS)*, 33(3):8, 2015.

[22] Ingmar Baumgart and Sebastian Mies. S/Kademlia: A Practicable Approach Towards Secure Key-based Routing. In *IEEE International Conference on Parallel and Distributed Systems*, pages 1–8, 2007.

[23] BBC. Fitness App Strava Lights up Staff at Military Bases. Online: `http://www.bbc.com/news/technology-42853072`, January 2018.

[24] Christian Beckel, Wilhelm Kleiminger, Romano Cicchetti, Thorsten Staake, and Silvia Santini. The ECO Data Set and the Performance of Non-Intrusive Load Monitoring Algorithms. In *ACM Conference on Embedded Systems for Energy-Efficient Buildings (BuildSys)*, pages 80–89, 2014.

[25] Christian Beckel, Leyna Sadamori, and Silvia Santini. Automatic Socio-economic Classification of Households Using Electricity Consumption Data. In *Conference on Future Energy Systems (e-Energy)*, pages 75–86, 2013.

[26] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and Efficiently Searchable Encryption. In *Springer Advances in Cryptology (CRYPTO)*, pages 535–552, 2007.

[27] Josh Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Stanford University, 1988. Yale University, Department of Computer Science.

[28] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[29] Sam Biddle. Stop Using Unroll.me, right now. It sold your data to Uber. Online: `https://theintercept.com/2017/04/24/stop-using-unroll-me-right-now-it-sold-your-data-to-uber/`, April 2017.

[30] John Biggs. It's time to build our own Equifax with blackjack and crypto. `https://techcrunch.com/2017/09/08/its-time-to-build-our-own-equifax-with-blackjack-and-crypto/`, September 2017.

[31] Arnar Birgisson, Joe Gibbs Politz, Ulfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[32] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible Protocols and Atomic Proxy Cryptography. In *Springer Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 127–144, 1998.

[33] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[34] Blockstack Virtualchain. `https://github.com/blockstack/virtualchain`, (accessed March 2, 2018).

[35] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-Preserving Symmetric Encryption. In *Springer Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 224–241, 2009.

[36] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based Encryption with Efficient Revocation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 417–426, 2008.

[37] Dan Boneh and Matt Franklin. Identity-based Encryption from the Weil Pairing. In *Springer Advances in Cryptology (CRYPTO)*, pages 213–229, 2001.

[38] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. Private Database Queries Using Somewhat Homomorphic Encryption. In *Springer Applied Cryptography and Network Security (ACNS)*, pages 102–118, 2013.

[39] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *Springer Theory of Cryptography (TCC)*, pages 325–341, 2005.

[40] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key Homomorphic PRFs and Their Applications. In *Springer Advances in Cryptology (CRYPTO)*, pages 410–428, 2013.

[41] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically Secure Order-Revealing Encryption: Multi-Input Functional Encryption Without Obfuscation. In *Springer Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 563–594, 2015.

[42] Dan Boneh, Gil Segev, and Brent Waters. Targeted Malleability: Homomorphic Encryption for Restricted Computations. In *ACM Innovations in Theoretical Computer Science (ITCS)*, pages 350–366, 2012.

[43] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *IEEE Symposium on Security and Privacy*, pages 104–121, 2015.

[44] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine Learning Classification over Encrypted Data. In *Network and Distributed System Security Symposium (NDSS)*, 2015.

[45] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[46] Bob Briscoe. Marks: Zero side effect multicast key management using arbitrarily revealed key sequences. *Workshop on Networked Group Communication*, 1736:301–320, 1999.

[47] Vitalik Buterin and Virgil Griffith. Casper the Friendly Finality Gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[48] Charles L. Buxton and William B. Atkinson. Hormonal Factors Involved in the Regulation of Basal Body Temperature During the Menstrual Cycle and Pregnancy. *The Journal of Clinical Endocrinology & Metabolism*, 8(7):544–549, 1948.

[49] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[50] Tej Chajed, Jon Gjengset, Jelle Van Den Hooff, M Frans Kaashoek, James Mickens, Robert Morris, and Nickolai Zeldovich. Amber: Decoupling User Data from Web Applications. In *USENIX Workshop on Hot Topics in Operating Systems (HotOS)*, volume 15, pages 1–6, 2015.

[51] Kiran Challapalli. The Internet of Things: A time series data challenge. IBM, Online: `http://www.ibmbigdatahub.com/blog/internet-things-time-series-data-challenge`, October 2014.

[52] Haowen Chan, Adrian Perrig, and Dawn Song. Secure Hierarchical In-network Aggregation in Sensor Networks. In *ACM Computer and Communications Security (CCS)*, pages 278–287, 2006.

[53] Nishanth Chandran, Juan A Garay, Payman Mohassel, and Satyanarayana Vusirikala. Efficient, Constant-round and Actively Secure MPC: Beyond the Three-Party Case. In *ACM Computer and Communications Security (CCS)*, pages 277–294, 2017.

[54] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancréde Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch Fully Homomorphic Encryption Over the Integers. In *Springer Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 315–335, 2013.

[55] Cisco. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. White Paper `https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf`, 2015.

[56] Compression Library zlip. `https://zlib.net/`, (accessed March 2, 2018).

[57] Nicolas T Courtois and Rebekah Mercer. Stealth Address and Key Management Techniques in Blockchain Systems. In *International Conference on Information Systems Security and Privacy (ICISSP)*, pages 559–566, 2017.

[58] Cunsheng Ding, Dingyi Pei, and Arto Salomaa. *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. World Scientific, 1996.

[59] Stuart Dredge. Yes, those Free Health Apps are Sharing your Data with other Companies. Guardian, Online: `https://www.theguardian.com/technology/appsblog/2013/sep/03/fitness-health-apps-sharing-data-insurance`, September 2013.

[60] Dropbox JPEG compression. `https://github.com/dropbox/lepton`, (accessed March 2, 2018).

[61] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *IEEE Conference on Local Computer Networks (LCN)*, pages 455–462, 2004.

[62] Cynthia Dwork. Differential Privacy: A Survey of Results. In *Springer Conference on Theory and Applications of Models of Computation*, pages 1–19, 2008.

[63] Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9:211–407, August 2014.

[64] Economist. Things Are Looking App: Mobile health apps are becoming more capable and potentially rather useful. Online: `https://www.economist.com/news/business/21694523-mobile-health-apps-are-becoming-more-capable-and-potentially-rather-useful-things-are-looking`, March 2016.

[65] Costas Efthymiou and Georgios Kalogridis. Smart grid privacy via anonymization of smart metering data. In *IEEE Conference on Smart Grid Communications (SmartGridComm)*, pages 238–243, 2010.

[66] Carl M Ellison, Bill Frantz, Brian Thomas, Tatu Ylonen, Ron Rivest, and Butler Lampson. SPKI Certificate Theory. RFC 2693, Online: `https://www.ietf.org/rfc/rfc2693.txt`, September 1999.

[67] Andres Erbsen, Asim Shankar, and Ankur Taly. Distributed Authorization in Vanadium. *arXiv preprint arXiv:1607.02192*, 2016.

[68] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1054–1067, 2014.

[69] ETH Zurich. FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems. Online: `https://www.flocklab.ethz.ch`, (accessed March 2, 2018).

[70] Ethereum. White-Paper. Online: `https://github.com/ethereum/wiki/wiki/White-Paper`, (accessed March 2, 2018).

[71] Experian. Data Breach Industry Forecast. *White Paper Online: `https://www.experian.com/assets/data-breach/white-papers/2015-industry-forecast-experian.pdf`*, 2015.

[72] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 45–59, 2016.

[73] Factom. https://www.factom.com/, (accessed March 2, 2018).

[74] Caroline Fontaine and Fabien Galand. A Survey of Homomorphic Encryption for Nonspecialists. *EURASIP Journal on Information Security*, 2007:15, January 2007.

[75] Mike Freedman. Time-series data: Why (and how) to Use a Relational Database Instead of NoSQL. Timescale, Online: https://blog.timescale.com/time-series-data-why-and-how-to-use-a-relational-database-instead-of-nosql-d0cd6975e87c, April 2017.

[76] David Freeman, Michael Scott, and Edlyn Teske. A Taxonomy of Pairing-Friendly Elliptic Curves. *Springer Journal of Cryptology*, 23(2):224–280, 2010.

[77] Kevin Fu, M Frans Kaashoek, and David Mazieres. Fast and Secure Distributed Read-only File System. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 13–29, 2000.

[78] Kevin Fu, Seny Kamara, and Tadayoshi Kohno. Key Regression: Enabling Efficient Key Distribution for Secure Distributed Storage. In *Network and Distributed System Security Symposium (NDSS)*, 2006.

[79] Kevin E Fu. *Integrity and Access Control in Untrusted Content Distribution Networks*. PhD thesis, Massachusetts Institute of Technology, 2005. Advisors: M. Frans Kaashoek and Ronald L. Rivest.

[80] William C. Garrison, Adam Shull, Steven Myers, and Adam J. Lee. On the Practicality of Cryptographically Enforcing Dynamic Access Control Policies in the Cloud. In *IEEE Symposium on Security and Privacy*, pages 819–838, 2016.

[81] Tingjian Ge and Stan Zdonik. Answering Aggregation Queries in a Secure System Model. In *Conference on Very Large Data Bases (VLDB)*, pages 519–530, 2007.

[82] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009. Stanford University: AAI3382729, Advisor: Dan Boneh.

[83] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *ACM Symposium on Theory of Computing (STOC)*, pages 169–178, 2009.

[84] Craig Gentry and Shai Halevi. Implementing Gentry's Fully-Homomorphic Encryption Scheme. In *Springer Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 129–148, 2011.

[85] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic Evaluation of the AES Circuit. In *Springer Advances in Cryptology (CRYPTO)*, pages 850–867, 2012.

[86] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the Security and Performance of Proof of Work Blockchains. In *ACM Conference on Computer and Communications Security (CCS)*, 2016.

[87] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 51–68, 2017.

[88] Peter Gilbert, Landon P Cox, Jaeyeon Jung, and David Wetherall. Toward Trustworthy Mobile Sensing. In *ACM Workshop on Mobile Computing Systems and Applications (HotMobile)*, pages 31–36, 2010.

[89] Peter Gilbert, Jaeyeon Jung, Kyungmin Lee, Henry Qin, Daniel Sharkey, Anmol Sheth, and Landon P Cox. Youprove: Authenticity and Fidelity in Mobile Sensing. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 176–189, 2011.

[90] Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[91] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable Garbled Circuits and Succinct Functional Encryption. In *ACM Symposium on Theory of Computing (STOC)*, pages 555–564, 2013.

[92] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption & How to Play Mental Poker Keeping Secret All Partial Information. In *ACM Symposium on Theory of Computing (STOC)*, pages 365–377, 1982.

[93] Google. Encrypted Bigquery Client. Online: `https://github.com/google/encrypted-bigquery-client`, (accessed March 2, 2018).

[94] Google. Google Sign-In. Online: `https://developers.google.com/identity/sign-in/android/`, (accessed March 2, 2018).

[95] Google Cloud Database. Security and Integration with Google Cloud. Online: `https://cloud.google.com/sql/docs`, (accessed March 2, 2018).

[96] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded Ciphertext Policy Attribute Based Encryption. In *Springer Colloquium on Automata, Languages and Programming (ICALP)*, pages 579–591, 2008.

[97] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based Encryption for Fine-grained Access Control of Encrypted Data. In *ACM Conference on Computer and Communications Security (CCS)*, pages 89–98, 2006.

[98] Matthew Green and Giuseppe Ateniese. Identity-Based Proxy Re-encryption. In *Springer Conference on Applied Cryptography and Network Security (ACNS)*, pages 288–306, 2007.

[99] Jessica Groopman and Susan Etlinger. Consumer Perceptions of Privacy in the Internet of Things. *Altimeter Group, Online: http://www.altimetergroup.com/pdf/reports/Consumer-Perceptions-Privacy-IoT-Altimeter-Group.pdf*, Juune 2015.

[100] AWS guide. Protecting data using encryption. Online: https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingEncryption.html, (accessed March 2, 2018).

[101] Trinabh Gupta, Rayman Preet Singh, Amar Phanishayee, Jaeyeon Jung, and Ratul Mahajan. Bolt: Data Management for Connected Homes. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 243–256, 2014.

[102] Shai Halevi and Phillip Rogaway. A Tweakable Enciphering Mode. In *Springer Advances in Cryptology (CRYPTO)*, pages 482–499, 2003.

[103] Dick Hardt. h. In *RFC 6749*, October 2012.

[104] Haris Z. Bajwa JD and Iltifat Husain MD. Health Apps can sell your data to insurance companies, and there's nothing you can do about it. Online: https://www.imedicalapps.com/2014/08/health-apps-insurance-companies/, August 2014.

[105] Warren He, Devdatta Akhawe, Sumeet Jain, Elaine Shi, and Dawn Song. ShadowCrypt: Encrypted Web Applications for Everyone. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1028–1039, 2014.

[106] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: Tool for Automating Secure Two-party Computations. In *ACM Computer and Communications Security (CCS)*, pages 451–462, 2010.

[107] Anwar Hithnawi, Vaibhav Kulkarni, Su Li, and Hossein Shafagh. Controlled Interference Generation for Wireless Coexistence Research. In *Software Radio Implementation Forum (SRIF) at ACM MobiCom*, pages 19–24, 2015.

[108] Anwar Hithnawi, Su Li, Hossein Shafagh, Simon Duquennoy, and James Gross. Poster Abstract: Cross-Layer Optimization for Low-power Wireless Coexistence. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 443–444, 2015.

[109] Anwar Hithnawi, Su Li, Hossein Shafagh, James Gross, and Simon Duquennoy. CrossZig: Combating Cross-Technology Interference in Low-power Wireless Networks. In *ACM International Conference on Information Processing in Sensor Networks (IPSN)*, pages 1–12, 2016.

[110] Anwar Hithnawi, Hossein Shafagh, and Simon Duquennoy. Poster Abstract: Low-Power Wireless Channel Quality Estimation in the Presence of RF Smog. In *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 137–138, 2014.

[111] Anwar Hithnawi, Hossein Shafagh, and Simon Duquennoy. Understanding the Impact of Cross Technology Interference on IEEE 802.15.4. In *ACM Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH) at ACM MobiCom*, pages 49–56, September 2014.

[112] Anwar Hithnawi, Hossein Shafagh, and Simon Duquennoy. TIIM: Technology-Independent Interference Mitigation for Low-power Wireless Networks. In *ACM Conference on Information Processing in Sensor Networks (IPSN)*, pages 1–12, 2015.

[113] Daniel Reiter Horn, Ken Elkabany, Chris Lesniewski-Lass, and Keith Winstein. The Design, Implementation, and Deployment of a System to Transparently Compress Hundreds of Petabytes of Image Files for a File-Storage Service. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1–15, 2017.

[114] Wen Hu, Peter Corke, Wen Chan Shih, and Leslie Overs. secFleck: A Public Key Technology Platform for Wireless Sensor Networks. In *Embedded Wireless Systems and Networks (EWSN)*, pages 296–311, 2009.

[115] Yih-Chun Hu, Markus Jakobsson, and Adrian Perrig. Efficient Constructions for One-way Hash Chains. In *Springer Conference on Applied Cryptography and Network Security (ACNS)*, pages 423–441, 2005.

[116] Yin Hu, William J. Martin, and Berk Sunar. Enhanced Flexibility for Homomorphic Encryption Schemes via CRT. In *Springer Conference on Applied Cryptography and Network Security (ACNS)*, pages 93–110, 2012.

[117] René Hummen, Jens Hiller, Hanno Wirtz, Martin Henze, Hossein Shafagh, and Klaus Wehrle. 6LoWPAN Fragmentation Attacks and Mitigation Mechanisms. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 55–66, 2013.

[118] René Hummen, Hossein Shafagh, Shahid Raza, Thiemo Voigt, and Klaus Wehrle. Delegation-based Authentication and Authorization for the IP-based Internet of Things. In *IEEE Conference on Sensing, Communication, and Networking (SECON)*, pages 284–292, 2014.

[119] René Hummen, Jan H Ziegeldorf, Hossein Shafagh, Shahid Raza, and Klaus Wehrle. Towards Viable Certificate-based Authentication for the Internet of Things. In *ACM workshop on Hot Topics on Wireless Network Security and Privacy (HotWiSec)*, pages 37–42, 2013.

[120] Yong Ho Hwang, Jae Woo Seo, and Il Joo Kim. Encrypted Keyword Search Mechanism Based on Bitmap Index for Personal Storage Services. In *IEEE Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 140–147, 2014.

[121] I. Damgård and M. Jurik. A generalisation, a Simplification and Some Applications of Paillers Probabilistic Public-key System. In *Springer Workshop on Practice and Theory in Public-Key Cryptography*, pages 119–136, 1992.

[122] Influxdata. Modern IoT Data Platform. Online: `https://www.influxdata.com/customers/iot-data-platform/`, (accessed March 2, 2018).

[123] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas Anderson. Privacy-preserving P2P Data Sharing with Oneswarm. In *ACM SIGCOMM*, pages 111–122, 2010.

[124] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access Pattern Disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

[125] Anca Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *Network and Distributed System Security Symposium (NDSS)*, 2003.

[126] Stephanie Jernigan, Sam Ransbotham, and David Kiron. Data Sharing and Analytics Drive Success With IoT. *MIT Sloan Management Review*, 58(1), September 2016.

[127] Yaoqi Jia, Tarik Moataz, Shruti Tople, and Prateek Saxena. OblivP2P: An Oblivious Peer-to-Peer Content Sharing System. In *USENIX Security*, pages 945–962, 2016.

[128] Juan Benet. IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3). `https://github.com/ipfs/papers`, (accessed March 2, 2018).

[129] Nikolaos Karapanos, Alexandros Filios, Raluca Ada Popa, and Srdjan Capkun. Verena: End-to-End Integrity Protection for Web Applications. In *IEEE Symposium on Security and Privacy*, pages 895–913, 2016.

[130] Chris Karlof, Naveen Sastry, and David Wagner. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 162–175, 2004.

[131] Kate Kaye. FTC: Fitness Apps Can Help You Shred Calories – and Privacy. Online: `http://adage.com/article/privacy-and-regulation/ftc-signals-focus-health-fitness-data-privacy/293080/`, May 2014.

[132] Sriram Keelveedhi, Mihir Bellare, and Thomas Ristenpart. DupLESS: Server-Aided Encryption for Deduplicated Storage. In *USENIX Security*, pages 429–446, 2013.

[133] Keybase. Publicly Auditable Proofs of Identity. Online: `https://keybase.io/`, (accessed March 2, 2018).

[134] Olga Kharif. 2016 Was a Record Year for Data Breaches. Cloudflare, Online: `https://www.bloomberg.com/news/articles/2017-01-19/data-breaches-hit-record-in-2016-as-dnc-wendy-s-co-hacked`, January 2017.

[135] Jun Young Kim, Wen Hu, Sanjay Jha, Hossein Shafagh, and Mohamed Ali Kaafar. Poster Abstract: Toward Efficient and Secure Code Dissemination Protocol for the Internet of Things. In *ACM Conference on Embedded Networked Sensor Systems (SenSys).*, 2015.

[136] Jun Young Kim, Wen Hu, Hossein Shafagh, and Sanjay Jha. SEDA: Secure Over-The-Air Code Dissemination Protocol for the Internet of Things. In *IEEE Transactions on Dependable and Secure Computing (TDSC)*, volume 99, December 2016.

[137] Brian King. Mapping an Arbitrary Message to an Elliptic Curve when Defined over GF(2n). *International Journal of Network Security*, 8(2):169–176, 2009.

[138] Kevin Kiningham, Mark Horowitz, Philip Levis, and Dan Boneh. CESEL: Securing a Mote for 20 Years. In *Embedded Wireless Systems and Networks (EWSN)*, pages 307–312, 2016.

[139] Wilhelm Kleiminger and Christian Beckel. ECO Data Set. Online: `https://www.vs.inf.ethz.ch/res/show.html?what=eco-data`, (accessed March 2, 2018).

[140] Wilhelm Kleiminger, Christian Beckel, and Silvia Santini. Household Occupancy Monitoring Using Electricity Meters. In *ACM Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*, pages 975–986, 2015.

[141] JeongGil Ko, Kevin Klues, Christian Richter, Wanja Hofer, Branislav Kusy, Michael Bruenig, Thomas Schmid, Qiang Wang, Prabal Dutta, and Andreas Terzis. Low Power or High Performance? A Tradeoff Whose Time Has Come (and Nearly Gone). In *Embedded Wireless Systems and Networks (EWSN)*, pages 98–114, 2012.

[142] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *USENIX Security*, pages 279–296, 2016.

[143] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *IEEE Symposium on Security and Privacy*, 2018.

[144] John Kolb, Kaifei Chen, and Randy H. Katz. The Case for a Local Tier in the Internet of Things. In *Technical Report No. University of California, Berkeley, EECS-2016-222*, 2016.

[145] J. Zico Kolter, Siddarth Batra, and Andrew Y. Ng. Energy Disaggregation via Discriminative Sparse Coding. In *Conference on Neural Information Processing Systems (NIPS)*, pages 1153–1161, 2010.

[146] Ben Lampert, Riad S Wahby, Shane Leonard, and Philip Levis. Robust, Low-cost, Auditable Random Number Generation for Embedded System Security. In *ACM Conference on Embedded Network Sensor Systems (SenSys)*, pages 16–27, 2016.

[147] Florian Lautenschlager, Michael Philippsen, Andreas Kumlehn, and Josef Adersberger. Chronix: Long Term Storage and Retrieval Technology for Anomaly Detection in Operational Data. In *USENIX Conference on File and Storage Technologies (FAST)*, pages 229–242, 2017.

[148] LevelDB by Google. `https://github.com/google/leveldb`, (accessed March 2, 2018).

[149] Amit Levy, Bradford Campbell, Branden Ghena, Daniel B Giffin, Pat Pannuto, Prabal Dutta, and Philip Levis. Multiprogramming a 64kB Computer Safely and Efficiently. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 234–251, 2017.

[150] Amit A Levy, James Hong, Laurynas Riliskis, Philip Levis, and Keith Winstein. Beetle: Flexible Communication for Bluetooth Low Energy. In *ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 111–122, 2016.

[151] Roman Lim, Federico Ferrari, Marco Zimmerling, Christoph Walser, Philipp Sommer, and Jan Beutel. FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems. In *ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN)*, pages 153–166, 2013.

[152] An Liu and Peng Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN)*, pages 245–256, 2008.

[153] Torsten Lodderstedt, Mark McGloin, and Phil Hunt. OAuth 2.0 Threat Model and Security Considerations. *IETF, RFC 6819*, January 2013.

[154] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In *ACM Symposium on Theory of Computing (STOC)*, pages 1219–1234, 2012.

[155] Sam Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems (TODS)*, 30(1):122–173, 2005.

[156] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schröder. Privacy and Access Control for Outsourced Personal Records. In *IEEE Symposium on Security and Privacy*, pages 341–358, 2015.

[157] Sinisa Matetic, Moritz Schneider, Andrew Miller, Ari Juels, Cornell Tech, and Srdjan Capkun. DelegaTEE: Brokered Delegation Using Trusted Execution Environments. In *Cryptology ePrint Archive: Report t Report 2018/160*, 2018.

[158] Stephanos Matsumoto and Raphael M Reischuk. IKP: Turning a PKI Around with Decentralized Automated Incentives. In *IEEE Symposium on Security and Privacy*, pages 410–426, 2017.

[159] Friedemann Mattern and Christian Floerkemeier. From the Internet of Computers to the Internet of Things. In *From Active Data Management to Event-based Systems and More*, pages 242–259. Springer, 2010.

[160] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *IEEE Symposium on Security and Privacy*, pages 143–158, 2010.

[161] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A Fistful of Bitcoins: Characterizing Payments Among Men with no Names. In *ACM Internet Measurement Conference (IMC)*, pages 127–140, 2013.

[162] Microsoft. Always Encrypted (Database Engine). Online: `https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine`, April 2017.

[163] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The Honey Badger of BFT Protocols. In *ACM Conference on Computer and Communications Security (CCS)*, pages 31–42, 2016.

[164] Claire Cain Miller. Tech Companies Concede to Surveillance Program. Online: `http://www.nytimes.com/2013/06/08/technology/tech-companies-bristling-concede-to-government-surveillance-efforts.html`, June 2013.

[165] Victor S. Miller. The Weil Pairing, and its Efficient Calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

[166] Nagendra Modadugu and Eric Rescorla. The Design and Implementation of Datagram TLS. In *Network and Distributed System Security Symposium (NDSS)*, 2004.

[167] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[168] Arvind Narayanan. *Data privacy: The non-interactive Setting*. PhD thesis, The University of Texas at Austin, 2009. Advisor: Vitaly Shmatikov.

[169] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies*. Princeton University Press `http://bitcoinbook.cs.princeton.edu`, 2016.

[170] Arvind Narayanan and Vitaly Shmatikov. Myths and Callacies of Personally Identifiable Information. *Communications of the ACM*, 53(6), 2010.

[171] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference Attacks on Property-Preserving Encrypted Databases. In *ACM Conference on Computer and Communications Security (CCS)*, pages 644–655, 2015.

[172] Jude Nelson, Muneeb Ali, Ryan Shea, and Michael J Freedman. Extending Existing Blockchains with Virtualchain. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[173] Netem. `https://wiki.linuxfoundation.org/networking/netem`, (accessed March 2, 2018).

[174] Jakob Nielsen. *Usability Engineering*. Elsevier, 1994.

[175] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-Preserving Ridge Regression on Hundreds of Millions of Records. In *IEEE Symposium on Security and Privacy*, pages 334–348, 2013.

[176] Parmy Olson. For Google Fit, Your Health Data Could Be Lucrative. Forbes, Online: `https://www.forbes.com/sites/parmyolson/2014/06/26/google-fit-health-data-lucrative`, June 2014.

[177] Omni. `http://www.omnilayer.org/`, (accessed March 2, 2018).

[178] Open Assets. `https://openasset.com/`, (accessed March 2, 2018).

[179] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based Encryption with Non-monotonic Access Structures. In *ACM Conference on Computer and Communications Security (CCS)*, pages 195–203, 2007.

[180] Pascal Paillier. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In *Springer Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 223–238, 1999.

[181] Adrian Perrig, John Stankovic, and David Wagner. Security in Wireless Sensor Networks. *Communications of the ACM*, 47(6):53–57, June 2004.

[182] Andreas Peter, Erik Tews, and Stefan Katzenbeisser. Efficiently Outsourcing Multiparty Computation under Multiple Keys. *IEEE Transactions on Information Forensics and Security*, 8(12):2046–2058, 2013.

[183] Raluca Ada Popa, Frank H Li, and Nickolai Zeldovich. An Ideal-Security Protocol for Order-Preserving Encoding. In *IEEE Symposium on Security and Privacy*, pages 463–477, 2013.

[184] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 85–100, 2011.

[185] Raluca Ada Popa, Emily Stark, Jonas Helfer, Steven Valdez, Nickolai Zeldovich, M Frans Kaashoek, and Hari Balakrishnan. Building Web Applications on Top of Encrypted Data Using Mylar. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 157–172, 2014.

[186] PRISM Surveillance Program. `https://www.theguardian.com/us-news/prism`, April 2016.

[187] Privacy Rights Clearinghouse. Chronology of Data Breaches from 2005 to Present Date. Online: `https://www.privacyrights.org/data-breaches`, (accessed March 2, 2018).

[188] Proof of Existence. `https://poex.io/`, (accessed March 2, 2018).

[189] Protocol Labs. Filecoin: A Cryptocurrency Operated File Network. Technical Report, Online: `http://filecoin.io/filecoin.pdf`, August 2014.

[190] Bartosz Przydatek, Dawn Song, and Adrian Perrig. SIA: Secure Information Aggregation in Sensor Networks. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 255–265, 2003.

[191] Python Cryptography Library. `https://cryptography.io/`, (accessed March 2, 2018).

[192] Python DHT library (Kademlia). `https://github.com/bmuller/kademlia`, (accessed March 2, 2018).

[193] Shahid Raza, Hossein Shafagh, Kasun Hewage, René Hummen, and Thiemo Voigt. Lithe: Lightweight Secure CoAP for the Internet of Things. *IEEE Sensors Journal*, 13(10):3711–3720, October 2013.

[194] Joel Reardon, David Basin, and Srdjan Capkun. SoK: Secure Data Deletion. In *IEEE Symposium on Security and Privacy*, pages 301–315, 2013.

[195] Ling Ren, Christopher Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Constants Count: Practical Improvements to Oblivious RAM. In *USENIX Security*, pages 415–430, 2015.

[196] Laurynas Riliskis, Hossein Shafagh, and Philip Levis. Poster: Computations on Encrypted Data in the Internet of Things Applications. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1668–1670, 2015.

[197] Amit Sahai and Brent Waters. Fuzzy Identity-Based Encryption. In *Springer Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 457–473, 2005.

[198] Tahmineh Sanamrad. *Encrypting Databases in the Clouds*. PhD thesis, ETH Zurich, Switzerland, 2014. ETH Zurich: 22450, Advisor: Donald Kossmann.

[199] Tahmineh Sanamrad, Lucas Braun, Donald Kossmann, and Ramarathnam Venkatesan. Randomly Partitioned Encryption for Cloud Databases. In *IFIP Conference on Data and Applications Security and Privacy (DBSec)*, pages 307–323, 2014.

[200] Stefan Saroiu and Alec Wolman. I Am a Sensor, and I Approve This Message. In *ACM Workshop on Mobile Computing Systems and Applications (HotMobile)*, pages 37–42, 2010.

[201] Andy Sayler, Taylor Andrews, Matt Monaco, and Dirk Grunwald. Tutamen: A Next-Generation Secret-Storage Platform. In *ACM Symposium on Cloud Computing (SoCC)*, pages 251–264, 2016.

[202] Bruce Schneier. Description of a New Variable-length Key, 64-bit Block Cipher (Blowfish). *Fast Software Encryption, Springer*, pages 191–204, 1994.

[203] Hossein Shafagh. Leveraging Public-key-based Authentication for the Internet of Things. *Thesis, RWTH Aachen University, Germany*, 2013.

[204] Hossein Shafagh. Toward Computing Over Encrypted Data in IoT Systems. In *XRDS: Crossroads, The ACM Magazine for Students Volume 22 Issue 2, Winter 2015*, pages 48–52, December 2015.

[205] Hossein Shafagh, Lukas Burkhalter, Simon Duquennoy, Anwar Hithnawi, and Sylvia Ratnasamy. Droplet: Decentralized Authorization for IoT Data Streams. *In ArXiv, arXiv:1806.02057v1 [cs.CR]*, June 2018.

[206] Hossein Shafagh, Lukas Burkhalter, and Anwar Hithnawi. Demo Abstract: Talos a Platform for Processing Encrypted IoT Data. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 308–309, 2016.

[207] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. Towards Blockchain-based Auditable Storage and Sharing of IoT Data. In *ACM Cloud Computing Security Workshop (CCSW)*, pages 45–50, 2017.

[208] Hossein Shafagh and Anwar Hithnawi. Poster Abstract: Security Comes First, A Public-key Cryptography Framework for the Internet of Things. In *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 135–136, 2014.

[209] Hossein Shafagh and Anwar Hithnawi. Poster: Come Closer - Proximity-based Authentication for the Internet of Things. In *ACM Conference on Mobile Computing and Networking (MobiCom)*, pages 421–424, 2014.

[210] Hossein Shafagh and Anwar Hithnawi. Privacy-preserving Quantified Self: Secure Sharing and Processing of Encrypted Small Data. In *ACM Workshop on Mobility in the Evolving Internet Architecture (MobiArch)*, pages 25–30, 2017.

[211] Hossein Shafagh, Anwar Hithnawi, Lukas Burkhalter, Pascal Fischli, and Simon Duquennoy. Secure Sharing of Partially Homomorphic Encrypted IoT Data. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 1–14, 2017.

[212] Hossein Shafagh, Anwar Hithnawi, Andreas Dröscher, Simon Duquennoy, and Wen Hu. Poster: Towards Encrypted Query Processing for the Internet of Things. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 251–253, 2015.

[213] Hossein Shafagh, Anwar Hithnawi, Andreas Dröscher, Simon Duquennoy, and Wen Hu. Talos: Encrypted Query Processing for the Internet of Things. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 197–210, 2015.

[214] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[215] Adi Shamir. Identity-based Cryptosystems and Signature Schemes. In *Springer Advances in Cryptology (CRYPTO)*, pages 47–53, 1984.

[216] Daniel Shanks. Class Number, a Theory of Factorization, and Genera. In *Symposium Mathematical Society*, pages 41–440, 1971.

[217] Chenguang Shen, Rayman Preet Singh, Amar Phanishayee, Aman Kansal, and Ratul Mahajan. Beam: Ending Monolithic Applications for Connected

Devices. In *USENIX Annual Technical Conference (ATC)*, pages 143–157, 2016.

[218] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. BlindBox: Deep Packet Inspection over Encrypted Traffic. In *ACM SIGCOMM*, pages 213–226, 2015.

[219] Elaine Shi, John Bethencourt, T.-H.H. Chan, Dawn Song, and Adrian Perrig. Multi-Dimensional Range Query over Encrypted Data. In *IEEE Symposium on Security and Privacy*, pages 350–364, 2007.

[220] Elaine Shi, Richard Chow, T-H. Hubert Chan, Dawn Song, and Eleanor Rieffel. Privacy-preserving Aggregation of Time-series Data. In *Network and Distributed System Security Symposium (NDSS)*, 2011.

[221] Rayman Preet Singh, S. Keshav, and Tim Brecht. A Cloud-based Consumer-centric Architecture for Energy Data Analytics. In *Conference on Future Energy Systems (e-Energy)*, pages 63–74, 2013.

[222] Nigel Paul Smart. *Cryptography: an Introduction*, volume 3. McGraw-Hill New York, 2003.

[223] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical Techniques for Searches on Encrypted Data. In *IEEE Security and Privacy*, pages 44–55, 2000.

[224] William Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson Education, 2003.

[225] Stampery. `https://stampery.com/`, (accessed March 2, 2018).

[226] Emil Stefanov and Elaine Shi. Oblivistore: High Performance Oblivious Cloud Storage. In *IEEE Symposium on Security and Privacy*, pages 253–267, 2013.

[227] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *ACM Conference on Computer and Communications Security (CCS)*, pages 299–310, 2013.

[228] Piotr Szczechowiak, Leonardo B Oliveira, Michael Scott, Martin Collier, and Ricardo Dahab. NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In *Embedded Wireless Systems and Networks (EWSN)*, pages 305–320, 2008.

[229] Texas Instruments. CC2538 System-on-Chip for 2.4-GHz IEEE 802.15.4. Online: `https://www.ti.com.cn/cn/lit/ug/swru319c/swru319c.pdf`, May 2013.

[230] Shruti Tople, Shweta Shinde, Zhaofeng Chen, and Prateek Saxena. AUTOCRYPT: Enabling Homomorphic Computation on Servers to Protect Sensitive Web Content. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1297–1310, 2013.

[231] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing Analytical Queries Over Encrypted Data. In *Proceedings of the Conference on Very Large Data Bases (VLDB)*, pages 289–300, 2013.

[232] Frederik Vercauteren. Optimal Pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.

[233] Boyang Wang, Ming Li, Sherman S. M. Chow, and Hui L. A Tale of Two Clouds: Computing on Data Encrypted under Multiple Keys. In *IEEE Conference on Communications and Network Security (CNS)*, pages 337–345, 2014.

[234] Frank Wang, James Mickens, Nickolai Zeldovich, and Vinod Vaikuntanathan. Sieve: Cryptographically Enforced Access Control for User Data in Untrusted Clouds. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 611–626, 2016.

[235] Xinlei Wang, Jianqing Zhang, Eve M Schooler, and Mihaela Ion. Performance Evaluation of Attribute-based Encryption: Toward Data Privacy in the IoT. In *IEEE Conference on Communications (ICC)*, 2014.

[236] Shawn Wilkinson, Tome Boshevski, Josh Brandoff, James Prestwich, Gordon Hall, Patrick Gerbes, Philip Hutchins, and Chris Pollard. Storj: A Peer-to-Peer Cloud Storage Network. Technical Report, Online: `https://storj.io/storj.pdf`, December 2016.

[237] Judson Wilson, Riad S Wahby, Henry Corrigan-Gibbs, Dan Boneh, Philip Levis, and Keith Winstein. Trust but Verify: Auditing the Secure Internet of Things. In *ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 464–474, 2017.

[238] Andrew C. Yao. Protocols for Secure Computations. In *IEEE Symposium on Foundations of Computer Science*, pages 160 – 164, 1982.

[239] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2010.

[240] Thomas Zachariah, Noah Klugman, Bradford Campbell, Joshua Adkins, Neal Jackson, and Prabal Dutta. The Internet of Things Has a Gateway Problem. In *ACM Workshop on Mobile Computing Systems and Applications (HotMobile)*, pages 27–32, 2015.

[241] Kimm Zetter. Google Discovers Fraudulent Digital Certificate issued for its Domain. https://www.wired.com/2013/01/google-fraudulent-certificate/, January 2013.

[242] Ben Zhang, Nitesh Mor, John Kolb, Douglas S. Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. The Cloud is Not Enough: Saving IoT from the Cloud. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2015.

[243] Philip R Zimmermann. *The official PGP user's guide*. MIT press, 1995.

[244] Torsten Zimmermann, Jens Hiller, Jens Helge Reelfs, Pascal Hein, and Klaus Wehrle. Split: Smart protocol loading for the iot. In *International Conference on Embedded Wireless Systems and Networks (EWSN)*, pages 49–54, 2018.

[245] Guy Zyskind, Oz Nathan, and Alex Pentland. Decentralizing Privacy: Using Blockchain to Protect Personal Data. In *IEEE Security and Privacy Workshops*, pages 180–184, 2015.

[246] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized Computation Platform with Guaranteed Privacy. *arXiv preprint arXiv:1506.03471*, 2015.