Talos: Encrypted Query Processing for the Internet of Things

Hossein Shafagh, Anwar Hithnawi,
Andreas Dröscher
Department of Computer Science
ETH Zurich, Switzerland
{shafagh, hithnawi, drandreas}@inf.ethz.ch

Simon Duquennoy
SICS Swedish ICT
Kista, Sweden
simonduq@sics.se

Wen Hu
School of Computer Science
and Engineering, UNSW
and NICTA, Sydney, Australia
wen.hu@unsw.edu.au

ABSTRACT

The Internet of Things, by digitizing the physical world, is envisioned to enable 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. In order to achieve this in 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 in terms of 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 a reasonable overhead. We envision Talos as an enabler of secure IoT applications.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*

General Terms

Security, Design

Keywords

Data Security; Internet of Things; Computing on Encrypted Data; Homomorphic Encryption; Cloud Computing

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

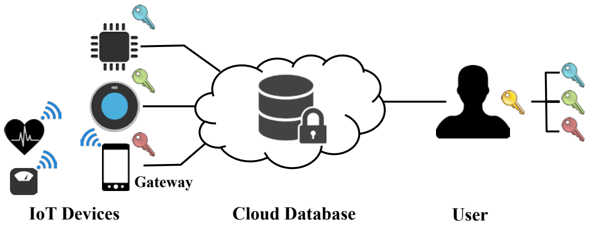


Figure 1: Talos enables protection of IoT data already at the origin. The Cloud database has no access to the encryption keys, but is able to process queries over encrypted data. All keys are derived from a master secret held by the user.

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 [51], 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 [31, 39].

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 into undesirable application delays. To overcome this limitation, several encrypted query processing approaches [11, 17, 36, 53, 54] 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 [53] is one of the first practical systems that integrates efficient encrypted query processing into the database management system. In CryptDB, the cloud

SenSys'15, November 01-04, 2015, Seoul, Republic of Korea
ACM 978-1-4503-3631-4/15/11.
http://dx.doi.org/10.1145/2809695.2809723.

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 modest computation overhead of 25% [53]. 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 paper, we present Talos[1], an IoT data protection system which securely stores encrypted IoT data on the Cloud database, while allowing for efficient database query processing over the encrypted data (see Figure 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 utilizies additive homomorphic encryption (see §3.1 for the detailed descriptions of different cryptographic terms). The location is potentially also sensitive. Thus, Talos applies deterministic encryption, allowing encrypted queries correlating heart rate with location. Finally, the timestamps are encrypted with order-preserving encryption, to allow order-related searches.

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

Contributions. This paper makes the following contributions:

- Design and evaluation of Talos, a fully-implemented E2E secure system for IoT. Talos is compatible to the core CryptDB, which implements *SQL-aware encryption schemes.* We make our prototype implementation for the Contiki OS [18] 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 in order to enable EC-ElGamal as an alternative additive homomorphic encryption scheme.
- We demonstrate experimentally the feasibility of Talos. We quantify the performance of Talos in terms of 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 [20, 42], a public testbed, emulating IoT-typical scenarios.

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 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 regularly offload their data into a Cloud database. In case the IoT device lacks any Internet-connectivity, the personal gateway (e.g., the smartphone for wearables) runs the Talos engine. We investigate the role of personal gateways in encrypted data processing in [59]. *(iii)* the Cloud database, which stores IoT data securely and has the ability to process queries over encrypted data. Note that IoT devices would potentially need to query the data on 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 §3, we detail the four main types of encryption and explain what functionality and security each type provides. We apply the data protection already at the IoT device, in order to reduce the attack surface and limit the need of trusted parties (see Figure 2).

[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

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

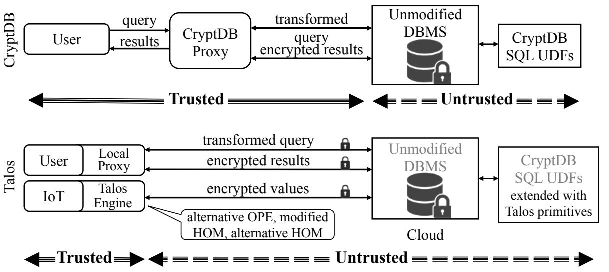


Figure 2: Talos extends CryptDB [53] to secure IoT data. Talos supersedes a trusted proxy with access to all keys. Instead, data protection is performed at IoT devices or personal gateways, e.g., smartphones.

For the Cloud database, we rely on our extended version of CryptDB [53] which is tailored towards IoT-application scenarios.

2.1 Background: CryptDB

CryptDB brings together powerful cryptographic tools, to create an encrypted query processing system targeted for database management systems (DBMS). In order to remain transparent and seamless, the en/-decryption process is performed on a trusted proxy which has access to the keying material (see upper part of Figure 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 [48] 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 few data types need to 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 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 §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 Security Analysis

CryptDB addresses two important threat models consisting of DMBS compromise, and a more severe one including the compromise of 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 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 [23]) that enable any computations over encrypted data, without revealing any information. However, these approaches are still computationally very expensive, rendering them impractical [55]. 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 devices. We introduce and elaborate our findings and the resulting design decisions further in a dedicated section (§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 §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 2). This allows to provide 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.

Approach. DTLS [47] provides confidentiality, authenticity, and integrity protection of communication. This is achieved by means of 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. Talos applies DTLS in 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.

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.

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 [23] 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 a 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 [58] 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 [62]. 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 a 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.

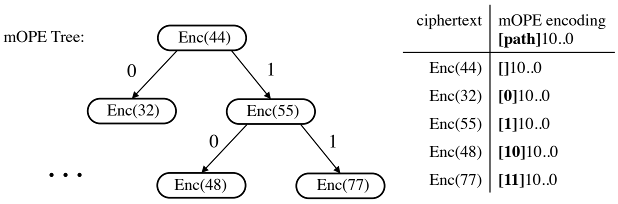
AES and Blowfish in CBC mode with a fixed initialization vector (IV) have the deterministic property, however due to undesirable leaks of prefix equality, they are not secure options for plaintexts larger than the block size. Therefore, we utilize AES-CMC [56] for plaintexts larger than 16 Byte, as recommended 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.:

$$if\ m_1 \leq m_2 \leq m_3,\ then\ c_1 \leq c_2 \leq c_3 \qquad (1)$$

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 by means of OPE [13,57]. One of the first provably secure OPE schemes is the approach introduced by Boldyreva et al. [9]. This



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

OPE scheme is, however, as computationally-intensive as asymmetric encryption.

In order to cope with resource constraints of IoT devices, we rely on a more recent interactive OPE approach by Popa et al. [55], 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. [55] 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 further reduce the communication overhead in §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 [23]. However, the involved computations are yet prohibitively high [55] even for full-fledged devices and by far infeasible for resource constrained devices. In order to support sum and average operations over encrypted data, it is however sufficient to utilize additive homomorphic encryption schemes, such that:

$$decrypt(c_1 \circ c_2) = decrypt(c_1) + decrypt(c_2) \qquad (2)$$

Several cryptosystems exhibit homomorphic properties [21]. To start with, the textbook RSA and ElGamal's cryptosystem are multiplicatively homomorphic. Goldwasser-Micali's (GM) [28] 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 during encryption results into large ciphertexts (i.e., $|pq|$ bits) which, given the single bit inputs, renders this scheme impractical. Benaloh [8] 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 [50] 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 [37] to reduce the encryption expansion of Paillier from 2 times k have the side-effect of significantly higher computation cost.

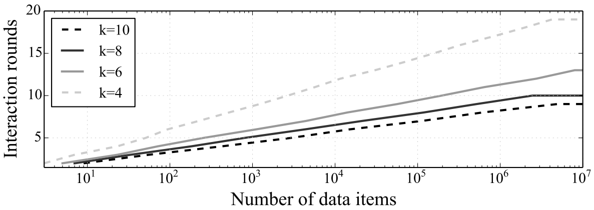


Figure 4: mOPE is an interactive approach. The number of interaction rounds depends on the current size of the database (i.e., the tree) and the parameter k of the k-ary tree in use. Talos selects k=10 in favor of fewer interaction rounds, resulting into a higher average rewrites per item in the Cloud (e.g., factor 4.7 for 10^7 items).

The Paillier [50] 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 1). In Talos, we apply a slight modification to Paillier, inspired by Ge and Zdonik [22], rendering it more efficient in terms of average bandwidth and computation per data item. Moreover, we explore the elliptic curve (EC)-ElGamal encryption scheme as an alternative additive homomorphic encryption scheme.

In §3.3, we detail our modification to the Paillier cryptossytem, our findings about EC-ElGamal, and the efficiency of each approach.

3.2 Optimized Order-Preserving Encryption

The traditional OPE by Boldyreva et al. [9] is computationally 5 orders of magnitude more intensive than symmetric encryption. mOPE [55] is a recent interactive order-preserving encryption scheme that allows us to drastically reduce this overhead. 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 basically the path from the root of the tree to the node, as illustrated in Figure 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 the values and replies. The server traverses the tree (i.e., in worst case $\mathcal{O}(\log n)$ interactions) until it finds the right spot to insert the new value. As we show later in our evaluation in §5, the communication overhead of mOPE is lower than the computation cost of the traditional OPE.

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 × $\lceil \frac{n}{16} \rceil$ (+ 16 RAND)
mOPE (integer)	(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 [9] (integer)	(0, 8]	16

Table 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.

In order to avoid long paths in the search tree, the tree needs to be rebalanced regularly. The server is able to rebalance without the help of the client, based on the order relation between nodes. However, as an 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 3. We chose 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 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. 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 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 §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 worse-case number of interactions depends on the tree size. In our case the worst-case is $\mathcal{O}(\log_{10} n)$ interactions (e.g., with 10^9 existing items the worst-case is 9 rounds).

In Talos, we rely on the prototype implementation of mOPE [55] and extend it with support for UDP and IPv6. To cope with the connectionless UDP, retransmission timers on both ends reassures 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 1, both OPE schemes produce final ciphertexts with the same length.

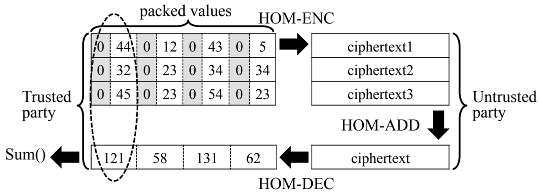


Figure 5: Illustration of our Paillier optimization. Several values are packed into one block, considered as one large number. The structure of packed values is maintained after decryption. After decryption, the sum of the final block is equal to the sum of all packed values.

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, in order 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:

$$n = pq, \quad d = (q-1)(p-1) \tag{3}$$

Encryption of the message m is performed as follows:

$$c = (g^m)(r^n) \bmod n^2 \tag{4}$$

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

$$m = L(c^d \bmod n^2)\ \mu \bmod n \tag{5}$$

where $L(x) = \dfrac{(x-1)}{n}$, 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).

In order to render Paillier more efficient, we apply a trick introduced by Ge and Zdonik [22] and illustrated in Figure 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 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 §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

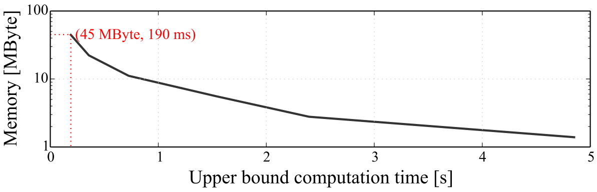


Figure 6: Memory-computation tradeoff in the Baby-Step-Giant-Step algorithm on a Google Nexus 5. Talos stores a pre-computed look-up table of 45 MByte and solves an ECDLP for an unsigned 32-bit value in maximum 190 ms with only 1 thread.

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 (for more details on elliptic curve cryptography (ECC) refer to [48]). 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'':

$$C = (C', C''), \text{ where } C' = M + rQ, \ C'' = rG \qquad (6)$$

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:

$$M = C' - dC'' \qquad (7)$$

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

$$\begin{aligned} C_1 + C_2 = (C_1', \ C_1'') + (C_2', \ C_2'') = \\ (M_1 + M_2 + r_1Q + r_2Q, \ r_1G + r_2G) \\ M_{1+2} = C_{1+2}' - dC_{1+2}'' = \\ M_1 + M_2 + r_1Q + r_2Q - d(r_1G + r_2G) \end{aligned} \qquad (8)$$

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 [40]. Koblitz suggests encoding of a message m into the x-coordinate of an elliptic curve [40]. This approach, however, is not homomorphic and therefore not suitable for EC-ElGamal.

We use a theoretical assumption from cryptography [48], 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

$$dec(C_1 + C_2) = M_1 + M_2 = m_1G + m_2G = (m_1 + m_2)G \quad (9)$$

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 6). Using the Baby-Step-Giant-Step algorithm [61], Talos is able to 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.

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: 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.

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).

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 [30] 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 [53] that concatenates a well defined ID, e.g., column-name or data type, to the master secret (MS):

$$\text{PRF}(\text{MS}|\text{ID}_i) = key_i \qquad (10)$$

The MS 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 are exploring data sharing for such systems in future work.

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 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].

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 [18] and *(ii)* the Cloud database component is an extended implementation of CryptDB [53]. For space reasons, we discuss only the former in more details.

OpenMotes are based on the TI CC2538 microcontroller [63], 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 [41,45]. Moreover, low-priced and energy-efficient cryptographic accelerators are predicted to find their ways into low-power platforms [2]. 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 [4] 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. 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.

We assume hardware AES block encryption to be available (which has been integrated in 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.

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

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.

In the following, we first define our evaluation objectives, describe the setup, metrics, and methodology. We continue in §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 §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 [42], 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 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 [38], 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

[4] OpenMote platform: openmote.com

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

[6] tinyDTLS; http://tinydtls.sourceforge.net/

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

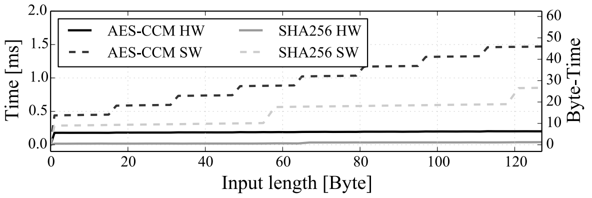


Figure 7: Secure E2E channel (AES-CCM, SHA256) with support of cryptographic accelerator (HW) compared to pure software implementation (SW).

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 × voltage × transmission current = 32 μs × 2.1 V × 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 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.

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 [35]. Note, the cryptographic accelerator allows concurrent computations on the main CPU. Hence, during the long computations ideally other tasks could be scheduled.

Table 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 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 6). More importantly, the computation of AES-CCM in hardware can run in parallel to the transmission of the preamble (8 symbols × 16 μs = 128 μs)

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 5: Energy consumption of data-protection components of Talos based on the *current* values in Table 3. Except for Blowfish, all crypto operations utilize the hardware crypto engine.

Note that the steps in Figure 7 in the software implementation are due to padding to a full block size. 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 pure hardware-based modes.

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

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 §3. In the following, we first discuss the performance of the individual crypto algorithms, and then elaborate on the overall system performance.

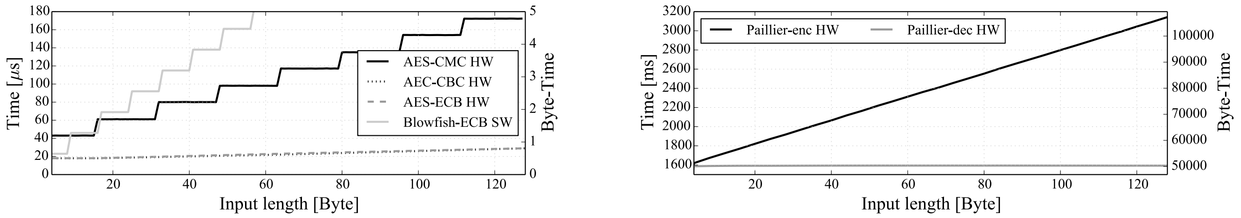
5.2.1 Microbenchmarks

We now discuss the time and energy measurements of the individual crypto algorithms, as summarized in Figure 8 and Table 5, respectively. The performance results combined with the ciphertext overheads of each algorithm (see Table 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 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 due to the fact that

[8] We open-source our test interface utilizing accurate system timers (see footnote 3).



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

AES-CMC applies AES-CBC twice in order to avoid early block leakage. Talos limits the use of AES-CMC only for data $\geq$16 Byte.

OPE. The traditional OPE, as introduced by Boldyreva [9], relies on the hypergeometric distribution (HGD). HGD is computationally intensive and in similar orders as Paillier. Talos utilizes mOPE which is about 2 orders of magnitude more efficient than OPE. This is due to the fact 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 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.

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 §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 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 in 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 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. In order 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 10, the energy per item reduces 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 to pack 32 items in one plaintext. This results in an optimal energy efficiency, by a factor 2 better than by EC-ElGamal. The possibility of packing values, however, is application scenario-dependent. In order to remain as generic as possible, we use EC-ElGamal as the default HOM for Talos.

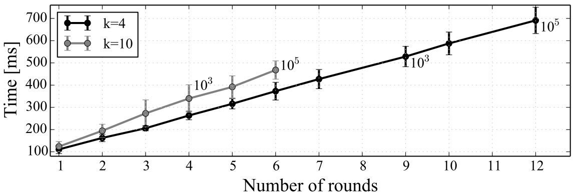


Figure 9: mOPE. Time of an insert operation based on the number of interactions for up to 10^5 items. Client is one wireless hop away from the gateway in Flocklab. Average RTT per interaction is 53 ms for k=4 and 68 ms for k=10. The per interaction CPU time is 264 μs for k=4 and 314 μs for k=10.

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 into a daily energy budget of 29.6 J. 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 §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 sport activities and 6 times per hour otherwise. For a person with 1 h sport activity per day, this would result into a daily 3,738 data items encrypted with additive homomorphic encryption. The location, logged every 15 min, is maybe less sensitive for 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 order-preserving encryptions in this scenario accounts to 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 5 days lifetime and a lithium-polymer battery capacity of 1.2 Wh (4.32 kJ).

6. DISCUSSION

This paper 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.

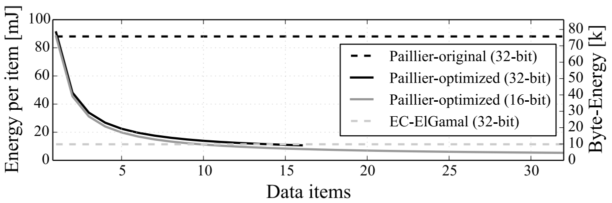


Figure 10: Additive homomorphic encryption. Our optimized Paillier is more efficient than original Paillier and can reach the efficiency of EC-ElGamal.

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 results into 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 into 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 §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 [24, 25]. The already achieved improvements of

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

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 [25], 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 a FHE-compatible form where arbitrary computations over the hidden plaintext are possible.

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 [43,49], which facilitated research on secure communication protocols for IoT. Hummen et al. [35] 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. [33] investigated the feasibility and advantages of hardware accelerators on low-power devices.

Privacy-Preserving Cryptography. There has been significant amount of work on cryptographic schemes [7, 10, 11, 26, 27, 44, 60] that could be utilized in privacy-preserving computations. Gentry's work [23, 24] 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 [25]. 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 [12, 14, 34].

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 database for queries such as equality check or comparison. This is the main reason of using weaker encryption schemes such as OPE and DET to allow the database to reduce the scope of computation.

Secure multi-party computation approaches [3,32] 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 [16, 52]. This would increase the security of approaches providing a distributed database interface for WSNs, such as TinyDB [46].

Differential privacy [19] 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. [17] 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 [36, 53, 54], among them CryptDB [53] which we discussed in depth in §2. Mylar [54] 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. [15] 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 [29], encrypt data before storage, 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* [5, 6] at the database is an alternative approach to encrypted query processing. This, however, implies that the user considers the *trusted hardware* trustworthy.

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* appear to be 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 anticipate that Talos will facilitate further research in this direction.

9. ACKNOWLEDGMENTS

The authors would like to express their gratitude to Raluca Ada Popa and Björn Tackmann for the insightful discussions during design phase of Talos. We thank Friedemann Mattern and Wilhelm Kleiminger for their comments on earlier versions of this paper. Moreover, we are thankful to the anonymous reviewers and our shepherd, Lu Su, for their valuable feedback. We thank Lukas Burkhalter for the support with the smartphone-related development. We would like to thank the Flocklab team, specially Roman Lim, for the support for extending Flocklab with OpenMotes. This work was partly funded by the European Commission through Nobel Grid (H2020-646184) and VINNOVA (Sweden's Innovation Agency).

10. REFERENCES

[1] A. Chen. GCreep: Google Engineer Stalked Teens, Spied on Chats, Gawker. Online: `www.gawker.com/5637234`, 2010.

[2] A. Perrig, J. Stankovic, and D. Wagner. Security in Wireless Sensor Networks. *Communications of the ACM*, 47(6):53–57, June 2004.

[3] Andrew C. Yao. Protocols for Secure Computations. In *Annual Symposium on Foundations of Computer Science (SFCS)*, 1982.

[4] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient Library for Cryptography. Online: `https://github.com/relic-toolkit/relic`.

[5] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal Security with Cipherbase. In *Conference on Innovative Data Systems Research (CIDR)*, 2013.

[6] S. Bajaj and R. Sion. TrustedDB: A Trusted Hardware Based Database with Privacy and Data Confidentiality. In *ACM Special Interest Group on Management of Data (SIGMOD)*, 2011.

[7] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and Efficiently Searchable Encryption. In *Advances in Cryptology (CRYPTO)*, 2007.

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

[9] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-Preserving Symmetric Encryption. In *EUROCRYPT*, 2009.

[10] D. Boneh and M. Franklin. Identity-based Encryption from the Weil Pairing. In *Advances in Cryptology (CRYPTO)*, 2001.

[11] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu. Private Database Queries Using Somewhat Homomorphic Encryption. In *Applied Cryptography and Network Security (ACNS)*, 2013.

[12] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *Theory of Cryptography (TCC)*, 2005.

[13] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. Semantically Secure Order-Revealing Encryption: Multi-Input Functional Encryption Without Obfuscation. In *EUROCRYPT*, 2015.

[14] D. Boneh, G. Segev, and B. Waters. Targeted Malleability: Homomorphic Encryption for Restricted Computations. In *Theoretical Computer Science Conference (ITCS)*, 2012.

[15] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine Learning Classification over Encrypted Data. In *Network and Distributed System Security Symposium (NDSS)*, 2015.

[16] H. Chan, A. Perrig, and D. Song. Secure Hierarchical In-network Aggregation in Sensor Networks. In *ACM Computer and Communications Security (CCS)*, 2006.

[17] D. X. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *IEEE Symposium on Secuirty and Privacy*, 2000.

[18] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *IEEE LCN*, 2004.

[19] C. Dwork. Differential Privacy. In *International Conference on Automata, Languages and Programming (ICALP)*, 2006.

[20] ETH Zurich. FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems. Online: `http://www.flocklab.ethz.ch/`.

[21] C. Fontaine and F. Galand. A Survey of Homomorphic Encryption for Nonspecialists. *EURASIP Journal on Information Security*, Jan. 2007.

[22] T. Ge and S. Zdonik. Answering Aggregation Queries in a Secure System Model. In *Very Large Data Bases (VLDB)*, 2007.

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

[24] C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Annual ACM Symposium on Theory of Computing (STOC)*, 2009.

[25] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic Evaluation of the AES Circuit. In *Advances in Cryptology (CRYPTO)*, 2012.

[26] O. Goldreich and R. Ostrovsky. Software Protection and Simulation on Oblivious RAMs. In *Journal of the ACM (JACM)*, 1996.

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

[28] S. Goldwasser and S. Micali. Probabilistic Encryption & How to Play Mental Poker Keeping Secret All Partial Information. In *Anual ACM Symposium on Theory of Computing (STOC)*, 1982.

[29] Google Cloud Database. Security and Integration with Google Cloud. Online: `https://cloud.google.com/sql/docs`.

[30] D. Hardt. The OAuth 2.0 Authorization Framework. In *RFC 6749*, 2012.

[31] 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: `http://www.imedicalapps.com/2014/08/health-apps-insurance-companies/`, 2014.

[32] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-party Computations. In *ACM Computer and Communications Security (CCS)*, 2010.

[33] W. Hu, P. Corke, W. C. Shih, and L. Overs. secFleck: A Public Key Technology Platform for Wireless Sensor Networks. In *Embedded Wireless Systems and Networks (EWSN)*, 2009.

[34] Y. Hu, W. J. Martin, and B. Sunar. Enhanced Flexibility for Homomorphic Encryption Schemes via CRT. In *Applied Cryptography and Network Security (ACNS)*, 2012.

[35] R. Hummen, H. Shafagh, S. Raza, T. Voigt, and K. Wehrle. Delegation-based Authentication and Authorization for the IP-based Internet of Things. In *IEEE Sensor and Ad Hoc Communications and Networks (SECON)*, 2014.

[36] Y. H. Hwang, J. W. Seo, and I. J. Kim. Encrypted Keyword Search Mechanism Based on Bitmap Index for Personal Storage Services. In *IEEE Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2014.

[37] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Pailliers probabilistic public-key system. In *Workshop on Practice and Theory in Public-Key Cryptography*, 1992.

[38] C. Karlof, N. Sastry, and D. Wagner. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[39] 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/`, 2014.

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

[41] J. Ko, K. Klues, C. Richter, W. Hofer, B. Kusy, M. Bruenig, T. Schmid, Q. Wang, P. Dutta, and A. Terzis. Low Power or High Performance? A Tradeoff Whose Time Has Come (and Nearly Gone). In *Embedded Wireless Systems and Networks (EWSN)*, 2012.

[42] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. 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)*, 2013.

[43] A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN)*, 2008.

[44] A. Lopez-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In *Annual ACM Symposium on Theory of Computing (STOC)*, 2012.

[45] M. P. Andersen, and D. E. Culler. System Design Trade-Offs in a Next-Generation Embedded Wireless Platform. In *Technical Report No. UCB/EECS-2014-162*, 2014.

[46] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. In *ACM Transactions on Database Systems (TODS)*, 2005.

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

[48] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[49] P. Szczechowiak, L. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In *Embedded Wireless Systems and Networks (EWSN)*, 2008.

[50] P. Paillier. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, 1999.

[51] Privacy Rights Clearinghouse. Chronology of Data Breaches from 2005 to Present Date. Online: `www.privacyrights.org/data-breach`.

[52] B. Przydatek, D. Song, and A. Perrig. SIA: Secure Information Aggregation in Sensor Networks. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[53] R. A. Popa, C. M. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

[54] R. A. Popa, E. Stark, S. Valdez, J. Helfer, N. Zeldovich, and H. Balakrishnan. Building Web Applications on Top of Encrypted Data Using Mylar. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[55] R. A. Popa, Frank H. Li, and N. Zeldovich. An Ideal-Security Protocol for Order-Preserving Encoding. In *IEEE Symposium on Security and Privacy*, 2013.

[56] S. Halevi and P. Rogaway. A tweakable enciphering mode. In *Advances in Cryptology (CRYPTO)*, 2003.

[57] T. Sanamrad, L. Braun, D. Kossmann, and R. Venkatesan. Randomly Partitioned Encryption for Cloud Databases. In *Data and Applications Security and Privacy (DBSec)*, 2014.

[58] B. Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). *Fast Software Encryption, Springer*, 1994.

[59] H. Shafagh, A. Hithnawi, A. Dröscher, S. Duquennoy, and W. Hu. Poster: Towards Encrypted Query Processing for the Internet of Things. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2015.

[60] A. Shamir. Identity-based Cryptosystems and Signature Schemes. In *Advances in Cryptology (CRYPTO)*, 1984.

[61] D. Shanks. Class Number, a Theory of Factorization, and Genera. In *Symposium Mathematical Society*, 1971.

[62] W. Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson Education, 3rd edition, 2002.

[63] Texas Instruments. CC2538 System-on-Chip for 2.4-GHz IEEE 802.15.4. Online: `www.ti.com.cn/cn/lit/ug/swru319c/swru319c.pdf`, 2013.