CrypSH: A Novel IoT Data Protection Scheme Based on BGN Cryptosystem

Subir Halder and Mauro Conti, *Senior Member, IEEE*

Abstract—The Internet of Things (IoT) is an emerging paradigm and has penetrated deeply into our daily life. Due to the seamless connections of the IoT devices with the physical world through the Internet, the IoT applications use the cloud to store and provide ubiquitous access to collected data. Sharing of data with third party services and other users incurs potential risks and leads to unique security and privacy concerns, e.g., data breaches. Existing cryptographic solutions are inapt for resource-constrained IoT devices, because of their significant computational overhead. To address these concerns, we propose a data protection scheme to store the encrypted IoT data in a cloud, while still allowing query processing over the encrypted data. Our proposed scheme features a novel encrypted data sharing scheme based on Boneh-Goh-Nissim (BGN) cryptosystem, with revocation capabilities and in-situ key updates. We perform exhaustive experiments on real datasets, to assess the feasibility of the proposed scheme on the resource constrained IoT devices. The results show the feasibility of our scheme, together with the ability to provide a high level of security. The results also show that our scheme significantly reduces the computation, storage and energy overheads than the best performed scheme in the state-of-the-art.

Index Terms—BGN cryptosystem, Cloud computing, Data security, Internet of Things, Somewhat homomorphic encryption, Secure sharing.

1 Introduction

A rapidly growing number of wireless sensor nodes and hybrid networks are leading to the emergence of the Internet of Things (IoT) and its various applications, including health and activity monitoring, home-automation, elderly care [1]. Many of these IoT applications have significantly transformed the way users perceive information about themselves and the adjacent environment. The current ecosystem of the IoT consists of typically designated low-power devices equipped with sensors that collect data. The collected data usually consist of sensor readings (e.g., skin temperature, breathing rate), health-related symptoms (e.g., heart rate variability, physiological stress), activity metadata (e.g., social lives, preference). For example, the wearable devices such as wrist bands and smart watches that can record users' stress levels from skin conductance [2], GPS trackers that can store physical activities [3] and fertility apps that can predict the most suitable time to conceive [4]. We envision that these new types of IoT based sensing applications will gain massive popularity with further advancement in the IoT.

Generally, the IoT based sensing applications produce a significant amount of data that need to be processed and stored. Due to the limited processing and data storage capabilities of the IoT devices, the processing and data storage functionalities are largely shifted to the cloud [5]. Additionally, processing and storing the data in the cloud improves scalability, ubiquitous access and sharing possibilities [6]. For example, an IoT device can send online queries on the data stored in the cloud to retrieve the raw sensor values or compute any statistics (e.g., average, standard deviation). In this way, the cloud based IoT eliminates the processing and storage limitations of resource-constrained IoT devices and eases the application deployment. However, secure data sharing in the cloud based IoT systems poses typical privacy risks, including unauthorised access of personalised data stored in the clouds, arbitrary threats due to the cloud being compromised (e.g., a curious cloud employee) [6]. Furthermore, due to the unique characteristics of the IoT system, the designing of secure cloud based IoT systems faces the following major challenges: (i) the system must support decryption on request, i.e., several types of functionalities can be supported using several types of decryption, (ii) some frequent queries on data statistics, e.g., finding the standard deviation, average can be calculated over the cipher texts directly, (iii) data owners must have complete control over their data and decide with whom, what and at which granularity they can share their data.

A promising approach to secure data in the cloud based IoT systems is to store data in encrypted form and have all the data encryption and decryption operations performed in the IoT device. However, this is quite impractical as the IoT devices are resource-constrained, and therefore prohibit any cloud server side query processing. Furthermore, storing encrypted data with conventional symmetric key encryption techniques, like AES, would serve the purpose, but render data unsearchable and unsharable. To overcome these limitations, in recent years, many encrypted query processing approaches [6], [7], [8], [9] have been proposed for the cloud based IoT systems. However, none of these approaches support calculation of multi-variate polynomials of degree 2 on the ciphertext, e.g., regression and variance analysis, which are essential for many IoT applications, e.g.,

• *S. Halder and M. Conti are with the Department of Mathematics, University of Padua, via Trieste 63, Padua, 35121, Italy.*
E-mail: subir.halder@math.unipd.it; conti@math.unipd.it

healthcare, smart meter. In this work, to overcome these limitations, we mainly focus on Somewhat Homomorphic Encryption (SHE) [10] technique and in particular, Boneh-Goh-Nissim (BGN) cryptosystem [11]. By leveraging the additive and multiplicative homomorphism properties of BGN cryptosystem, an IoT device can perform algebraic operations to evaluate the multi-variate polynomials of degree 2 on encrypted data by keeping the ciphertext size constant [10]. There are many IoT applications, where regression and variance analysis are essential and important for the future course of actions. or instance, in the health monitoring application, a medical practitioner can prescribe appropriate medicine to the patient using the variance analysis of heartbeat or blood pressure. Further, a medical practitioner can estimate the criticality of a patient with the help of regression analysis on different health parameters, e.g., heartbeat, blood pressure [6]. Another important property of BGN cryptosystem is that the execution time depends on the size of the plaintext [11]. Due to the shorter message size of the IoT device (typically, vary between 32 and 160-bit [1]), we can speed up the overall execution process using BGN cryptosystem in the IoT devices.

Recently, Xue et al. [7] and Shafagh et al. [6] designed Kryptein and Pilatus, respectively, to secure the interactions between the IoT devices and the cloud. Kryptein is a compressive sensing [12] based encryption scheme for secure query processing in the IoT systems. The major limitation of Kryptein is that it does not support any sharing features. On the contrary, Pilatus is the first to combine encrypted query processing with sharing, for the cloud based IoT systems. More importantly, Pilatus supports both individual and group sharing features with in-situ key-update capability. However, Pilatus does not guarantee freshness or correctness of the shared encrypted data. Furthermore, since the design of Pilatus is based on Paillier cryptosystem [13], which is an additive homomorphic scheme, it only supports the addition related queries, e.g., range, sum. Moreover, due to their significant computational overhead, most of the Partially Homomorphic Encryption (PHE) based schemes, e.g., Telos [8], CryptDB [9] are not suitable for the energy constrained IoT devices. In short, most of the current state-of-the-art solutions either support secure query processing or sharing, but not both at the same time. Also, when both secure query processing and sharing are supported, the solutions do not guarantee freshness of the shared data. Moreover, most of the PHE schemes are not suitable for the energy constrained IoT devices, due to their significant computational overhead. In this paper, we address the challenges of encrypted query processing and secure sharing of IoT data, as shown in Fig. 1.

Contribution: In this paper, we present CrypSH, a new IoT data protection scheme, which stores the encrypted IoT data on the cloud, while allowing for secure query processing and sharing operation over the encrypted IoT data. One of the major features of CrypSH is that it supports efficient statistical computation on ciphertexts. In particular, to avoid expensive data decryption on the IoT devices for frequently used queries, e.g., determining the standard deviation (SD), average (AVG) and summation (SUM), the CrypSH efficiently computes the statistics directly on ciphertexts in the cloud. Our CrypSH enables secure sharing of the IoT

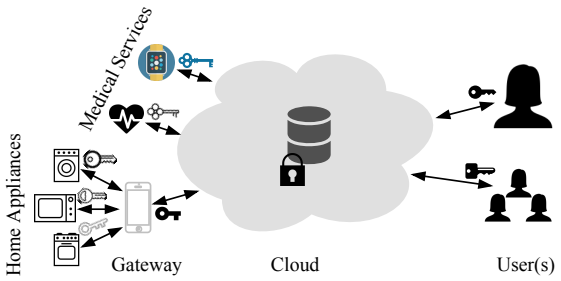


Fig. 1. Sharing and processing of encrypted data in Cloud based IoT systems.

data based on our design of the re-encryption scheme. In particular, our designed re-encryption scheme allows the cloud server to convert ciphertext generated from the sender's key to ciphertext generated from the receiver's key, without revealing the plaintext. For example, if a sender, say Alice, intends to share data with a receiver, say Bob, Alice calculates a token that enables the cloud server to re-encrypt Alice's data (without decrypting) for Bob. Similar procedure is followed by Alice to share her data with a group. It is worth mentioning that Alice generates the re-encryption token based on her own private key and Bob's public key. We also introduce a key revocation scheme that allows the IoT devices to terminate data sharing operation whenever required. Further, to protect the owner's old data, we proposed an in-situ key update technique at the cloud end. The major contributions of this paper are as follows.

- We design and evaluate the CrypSH, a BGN cryptosystem based encryption scheme for the query processing and data sharing in the cloud based IoT systems. To the best of our knowledge, it is the first BGN cryptosystem based encryption scheme that achieves energy-efficient secure query processing and data sharing for the cloud based IoT systems.
- We also devise a sharing revocation mechanism to terminate a sharing operation at any time and in-situ key update technique. One of the main features of the CrypSH is that it ensures data freshness in addition to authentication, confidentiality and integrity of the IoT devices in the cloud based IoT systems.
- We experimentally demonstrate the feasibility of the CrypSH. We measure the performance of the CrypSH in terms of the computational overhead, storage overhead, energy overhead, throughput, data freshness and end-to-end delay. Experimental results show that the CrypSH significantly improves the underlying overheads without compromising the network performance metrics like throughput, end-to-end delay compared to the state-of-the-art realization, Pilatus [6].

This work is an extended version of our previously published work [14]. In particular, the current work extends our prior work in the following aspects: (i) We summarise and present more existing works most relevant to our context in a more structured manner. (ii) We enhance the system model by including the background of BGN cryptosystem, and design and security goals. (iii) We propose a

sharing technique to enable the group sharing operations. (iv) We provide security analysis of the CrypSH. (v) We conduct more comprehensive and rigorous experiments on real datasets to validate the correctness of the CrypSH.

Organisation The rest of this paper is organised as follows. Section 2 discusses the related work. We describe the system model considered for the present work in Section 3. We introduce the privacy preserving query processing and sharing mechanisms based on the BGN cryptosystem in Section 4. In Section 5, we provide a security analysis, based on the design and security goals introduced in Section 2. In Section 6, we evaluate the performance of CrypSH and compares them to the state-of-the-art. Finally, we conclude the work in Section 7.

2 Related Work

Recently, many schemes were reported with an objective of secure communication in the cloud based IoT systems. To achieve this goal, the researchers have focused on secure end-to-end channels [21], [22], privacy-preserving cryptographic schemes [15], [23], [24], and encrypted query processing techniques [6], [9], [20]. Since we are focusing on the encrypted query processing techniques, we present here some of the works more relevant to our context.

Song et al. [5] formalised the searchable encryption technique for the purpose of permitting text search over encrypted data. Based on the usage of either symmetric or asymmetric encryption technique, they classified the searchable encryption technique into two types, namely, symmetric searchable encryption and public key based searchable encryption. The major advantage of public key based searchable encryption is that it provides stronger security and more expressive query than symmetric searchable encryption. In our work, we consider the public key based searchable encryption. Later, Boneh et al. [24] initially defined the notion of public key based searchable encryption and developed a three-party secure query processing protocol that supports the homomorphic computations on encrypted data. Popa et al. [9] developed the CryptDB, one of the first practical schemes, that integrates the efficient online encrypted query processing on the cloud without requiring the modification of the cloud database.

In [15], Henecka et al. presented TASTY, a novel tool for two-party secure function evaluation. Basically, the TASTY is a new compiler that can generate protocols based on the homomorphic encryption and efficient garbled circuits, as well as a combination of both. Most importantly, it is observed that the TASTY significantly improves the online latency for securely evaluating the AES functionality. Recently, different from [15], Liang et al. [16] presented a novel Identity based Data Sharing and Searching (IDSS) technique for the cloud. In addition to data sharing and searching, IDSS supports both the re-encryption and secret key update features. However, both the re-encryption and secret key update are performed at the end of the proxy server. To minimise the query latency, Wang et al. [17] designed a Secure Graph DataBase (SecGDB) encryption scheme. The SecGDB uses the efficient additive homomorphic encryption and distorted circuits for supporting the shortest distance queries having optimal time and storage complexities. Further, the authors proposed an auxiliary data structure, called queue history for obtaining better time complexity over multiple queries. In [18], the authors proposed a secure and highly available cloud database scheme, SHAMC, for the multi-cloud environment. The SHAMC uses the idea of secure multi-party computation and homomorphic encryption for storing data and executing queries directly on the ciphertext.

Different from the aforementioned schemes, Kotamsetty and Govindarasu [19] proposed an Adaptive Latency-aware Query Processing (ALQP) scheme for the cloud based IoT system. The main objectives of the work are to minimise query latency and energy consumption overhead. To achieve the goals, they divided the result of a large query into several suitable smaller sized results and processed them concurrently. Shafagh et al. [8] designed the Telos, an extended version of the CryptDB, specially for the IoT devices to improve the query efficiency and reduce energy overhead. The Telos relies on several algorithms that accelerate order-preserving and partially homomorphic encryption. However, though the computation overhead is reduced through the optimisation algorithms, the expensive energy and computation overheads prohibit the application of all of these schemes for the constrained IoT devices. Unlike any cryptographic technology, Shafagh et al. [25] present a blockchain based distributed access control and data management scheme for the cloud based IoT system. They combine the blockchain with an off-chain storage for a scalable secure data storage purpose. Recently, Kim et al. [20] proposed a secure and fault-tolerant key agreement mechanism for group data sharing in the cloud based IoT systems. The authors used symmetric balanced incomplete block design and group signature techniques to protect the security of the outsourced data and secure group data sharing in the cloud. Different from [17] and [20], to address the concerns generated due to data sharing with third-party services and other users, Shafagh et al. [6] presented a data protection platform named as Pilatus. Pilatus enables the cloud to store only encrypted data, though it is able to process specific queries. This work mainly focuses on the Paillier cryptosystem [13]. Similar to [6], Xue et al. [7] proposed a compressive sensing based encryption scheme, Kryptein, for secure query processing in the cloud based IoT systems. The Kryptein uses several techniques, like random compressed encryption, statistical decryption to ensure secure data insertion, accurate raw data decryption and efficient statistical computation in the cloud based IoT system. However, it does not support secure data sharing operation.

In Table 1, we summarise the key features of the existing state-of-the-art works and the CrypSH. In the specific context of secure query processing and sharing, except the Pilatus [6], none of the current solutions support secure sharing. Even if both secure query processing and sharing features are supported, the solutions do not guarantee freshness of the shared data. Moreover, due to their significant computational overhead, most of the schemes are not energy efficient. In this paper, we consider the Pilatus as our main competitor and show that our proposed scheme not only achieves system security but also improves on performance significantly than the Pilatus.

TABLE 1
Comparison of typical query processing and sharing techniques

Scheme	Query processing	Sharing	Authentication	Confidentiality	Integrity	Data Freshness	Environment
TASTY [15]	Yes	Yes	Yes	Yes	Yes	No	Cloud
IDSS [16]	Yes	Yes	Yes	Yes	Yes	No	Cloud
SecGDB [17]	Yes	No	Yes	Yes	Yes	No	Cloud
SHAMC [18]	Yes	No	Yes	Yes	Yes	No	Cloud
CryptDB [9]	Yes	No	Yes	Yes	Yes	No	Cloud
ALQP [19]	Yes	No	Yes	Yes	Yes	No	Cloud-IoT
ESIoT [20]	Yes	No	Yes	Yes	Yes	No	Cloud-IoT
Talos [8]	Yes	No	Yes	Yes	Yes	No	Cloud-IoT
Kryptein [7]	Yes	No	Yes	Yes	Yes	No	Cloud-IoT
Pilatus [6]	Yes	Yes	Yes	Yes	Yes	No	Cloud-IoT
CrypSH	Yes	Yes	Yes	Yes	Yes	Yes	Cloud-IoT

3 System Overview

In this section, we briefly discuss the different models used in the CrypSH. Particularly, Section 3.1 presents the background of the BGN cryptosystem. Section 3.2 discusses the CrypSH architecture. We then introduce the threat model in Section 3.3. Section 3.4 presents the design and security goals. Finally, in Section 3.5, we present the assumptions.

3.1 BGN Cryptosystem

The homomorphic encryption is a type of encryption scheme, which enables a third party (e.g., cloud) to execute operations on plaintexts, to be performed on their respective ciphertexts without disclosing the plaintexts [10]. The BGN cryptosystem is a novel homomorphic public key encryption scheme based on finite groups of composite order that support a bilinear map [11]. It assimilates the Paillier [13] encryption scheme with the Okamoto-Uchiyama [26] encryption scheme. The use of Paillier and bilinear map in BGN cryptosystem allows to perform any number of additions and one multiplication on the ciphertexts. Generally, the BGN cryptosystem consists of three algorithms, namely, key generation, encryption and decryption. The details of these three algorithms are described below.

Key Generation: Initially, each user executes the key generation algorithm to generate it's own private and public keys. The steps are as follows:

- The user picks two distinct prime numbers, x' and y' such that $x' = 2x + 1$ and $y' = 2y + 1$, where x and y are two distinct numbers.
- The user selects e as a bilinear map [27] such that $e : Z_n \times Z_n \rightarrow Z_n^*$, where Z_n and Z_n^* are groups of order $n = x'y'$. The user computes $\lambda = xy$, chooses a random generator $g \in Z_n$ with order λ. Subsequently, it computes $\eta = g^y \mod n$ and η is the random generator of Z_n.
- The user publishes the public key $pk_u = (n, g, \eta)$ and store the private key $prk_u = x$.

Encryption: The user executes the encryption algorithm to generate the ciphertext c of the plaintext (or, message) m. The steps are as follows:

- The user first randomly chooses a number $r \in Z_n$.
- The user then calculates $c = E(m) = g^m \eta^r \mod n$.
- Finally, the user outputs c as the ciphertext.

Decryption: To decrypt the ciphertext c to extract the plaintext m, the user performs the following steps:

- The user first computes $c^x = (g^m \eta^r)^x = (g^x)^m$ (Note that $\eta^x \equiv 1 \mod n$).
- The user solves the discrete logarithm of $(g^x)^m \mod n$ with the base g^x using the Pollard's lambda algorithm [28] and produces the plaintext $m = D(c)$.

It is worth mentioning that the decryption algorithm of the BGN cryptosystem takes polynomial time and depends on the size of the message space M, where $0 \leq m \leq M$. Thus, the BGN cryptosystem as described earlier is more suitable for encrypting small sized messages of the IoT devices. Generally, the size of a message in an IoT device ranges between 32 to 160-bit [1], [6].

3.2 Architecture

The CrypSH uses the BGN cryptosystem to allow query processing over encrypted data in the cloud. The CrypSH works by encrypting all data entries stored in the IoT devices; next, only the encrypted data (i.e., cipher) is sent to the cloud (i.e., server) and the originally stored data is deleted from the IoT device. With the help of the precomputed key, an IoT device encrypts the streaming data and uploads the same to the cloud efficiently. So, when data statistics such as SUM, AVG and SD are requested by an IoT device, the cloud will compute the statistics over the ciphertext (i.e., without requiring to decrypt the raw data) and send back the ciphertext to the IoT device. Only the IoT device, which owns the correct keys can decrypt the ciphertext and obtain SUM, AVG and SD results with little additional computational cost. Finally, the CrypSH offers secure sharing of encrypted data with an IoT device and/or a group of IoT devices, based on re-encryption techniques. In fact, during secure sharing of encrypted data with an IoT device and/or a group of IoT devices, the CrypSH features include access withdrawal with in-situ re-keying.

Our CrypSH comprises three main modules, namely, the client engine, the cloud engine and an ICA. Figure 2 illustrates the architecture and workflow of the CrypSH. The main functionalities of each of the three modules are given below.

Client Engine: The client engine runs in each IoT device or gateway. Basically, it generates raw data (e.g., time, location, audio/image/video files) on computational devices and stores encrypted data in the cloud. Also, it selectively shares the encrypted versions of data, i.e., ciphertexts with a third party user (i.e., IoT device). During sharing of ciphertext(s), the client engine interacts with an ICA to verify the

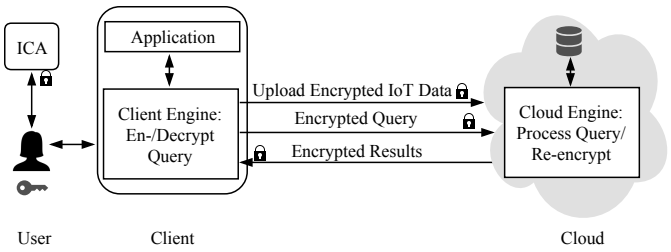


Fig. 2. The CrypSH architecture.

identity of the other client engine and with the cloud engine for secure storage, sharing and retrieving data. Particularly, in the CrypSH, the client engine encrypts, decrypts, stores the keys, and handles sharing-related activities like generating re-encryption tokens, triggering revocation and in-situ re-keying. Since the client engine places encrypted data on the cloud, hence, raw data is never revealed to the cloud administrator. This protects the confidentiality of the data if the cloud service provider is malicious or compromised. Using private key and time stamp (see Section 4.1), the client engine also protects the integrity and freshness of the data that the cloud delivers to an IoT device. For the sake of convenience, in this paper, the term 'user', 'client', 'client engine' and 'IoT device' are used interchangeably.

Cloud Engine: In CrypSH, the cloud engine is application-agnostic and supports primary database interface and features. It provides storage for the continuously generated IoT data and has the ability to handle online queries. The cloud engine stores and accesses data only in encrypted form. It supports the mechanisms needed for processing encrypted data, i.e., insertion, homomorphic addition and multiplication, re-encryption and in-situ re-keying. For example, to store encrypted data in the cloud, insertion queries are used in the cloud database. During designing of the CrypSH, we considered a structural database such as MySQL and used user defined functions to replace the default routines with crypto enabled ones.

ICA: The ICA is an independent trusted authority responsible for authenticating the identity of each IoT device during the initialisation phase (see Section 4). It is also responsible for storing the public key of each IoT device. In CrypSH, the ICA is used by the client engine to search for the public key of other IoT devices. In multi-user systems, the ICA is a standard requirement and considered as a trustworthy external entity [6]. Our scheme is independent of the ICA and borrows this function from systems such as the Keybase [29]. Basically, the Keybase is an independent party, where users reveal their identity by posting their own public key or token.

3.3 Threat Model

The Pilatus and Kryptein, both addressed two important threat models, namely, curious Data Base Administrator (DBA) and a more severe one cloud server compromise. In our threat model, we not only inherit these two models, but also address a more stronger network based threat model. The threat model that we consider in this work is as follows.

Threat 1 (Curious DBA): In the CrypSH, we assume that the DBA is honest but curious and performs passive attack [6], [7]. Specifically, the DBA correctly executes the database protocols and neither alters stored data nor query results that would be noted by the client/tenant and would malign the reputation of the cloud service provider. This threat includes Database Management System (DBMS) software compromises, root access to DBMS machines, and even access to the random access machine of physical machines [30]. With the increase in database consolidation within the cloud data centres, outsourcing of databases to public cloud computing infrastructures, and the use of third-party DBAs, this threat is increasingly important [9]. One of the goals of the CrypSH is to attain confidentiality of the stored data. We believe that this is a valid model, because if any protocol violation is detected, the client could penalise the service provider.

Approach: The CrypSH allows processing of SQL queries over encrypted data. It uses one of the popular SHE techniques, i.e., BGN to protect data from the curious DBA or other attackers who have complete access to the data stored in the cloud. As the data stored in the cloud are in encrypted form, hence, the CrypSH thwarts the private information from being accessed by the curious DBA. We discuss the design procedure in Section 4.

Threat 2 (Network based Attacks): In this threat, the attacker targets the communication between the IoT device and the cloud. The attackers can perform passive or active attacks to reconstruct the client data. For example, a passive attacker can launch eavesdropping, replay and traffic analysis attacks. Conversely, an active attacker can launch man-in-the-middle or impersonating attacks.

Approach: The CrypSH provides secure protection from the passive attackers by ensuring that without the user private key the raw data cannot be accurately retrieved. This allows providing confidentiality, integrity protection, data freshness and authenticity of the data. In contrast, the CrypSH addresses the active attacks using a public key, i.e., BGN cryptography based authentication scheme. In CrypSH, an IoT device first registers itself in an ICA and only authenticated IoT devices are allowed to share encrypted data with another IoT device. Only an IoT device possessing the correct private key can retrieve the raw data. We provide more detailed discussion on protection from passive and active attackers in Section 4.

Threat 3 (Server Compromised): Under this threat, the cloud server is compromised and all the encrypted data stored in the cloud are revealed to an attacker. An attacker accesses the cloud server and can launch different attacks.

Approach: The CrypSH provides the following guarantees: (i) under any circumstances, the cloud server has no access to any raw data, (ii) the cloud server never requests for decrypted data from the client, and (iii) during query processing, the raw data always remain encrypted. Hence, even if the cloud is fully compromised, in the CrypSH, the attacker cannot retrieve the raw data.

3.4 Design and Security Goals

The objective of the CrypSH is to give a solution to basic security during secret storing of data in the cloud and

prevent replay attacks while retrieving and sharing secret information. The essential properties of a cloud based IoT system required for maintaining basic security goals are as follows:

Authentication: To ensure that the shared data are coming from a reliable source, the CrypSH should provide the authenticity of the IoT devices. Specifically, the ICA should verify the authenticity of the IoT devices

Confidentiality: To ensure the confidentiality of shared data, the CrypSH should protect the IoT data stored in the cloud, i.e., neither external adversaries nor internal adversaries should be able to access IoT data. In fact, the CrypSH is designed in such a way that in case an IoT device is compromised and the group key information is revealed, only the data associated with the compromised device and the group is exposed.

Integrity: To ensure that correct data are received from the cloud, the CrypSH should provide integrity. Particularly, the IoT device should be able to detect any modification of data during communication with the cloud.

Data Freshness: The data received by an IoT device at a particular time should be recent data and not old data, which may be replayed by the adversaries.

3.5 Assumptions

In this work, we assume that the cloud is honest but curious [6], [7]. Without loss of generality, we also assume that an adversary does not collude with the parties that hold the secret/private key of the BGN cryptosystem, as we cannot protect this type of attack by using any encryption scheme. Furthermore, we assume that the ICA correctly authenticates the IoT devices and publishes the public key of other IoT devices for generating re-encryption tokens. Finally, under normal scenario, the CrypSH assumes that the application behaves correctly and does not reveal keys to malicious users/clients.

4 The Proposed Scheme: CrypSH

In this section, we present a detailed description of the CrypSH. Particularly, in Section 4.1, we discuss the processing and sharing mechanisms of encrypted data. Section 4.2 presents the key revocation mechanism. In Section 4.3, we put forward a group sharing mechanism. Finally, we discuss the homomorphic operations in Section 4.4.

4.1 Processing and Sharing of Encrypted Data

This section illustrates how the CrypSH encrypts and shares the user data. Our CrypSH scheme operates in two modes: standard and sharing. The standard mode of operation covers the single-key case, whereas the sharing mode enables cryptographically protected sharing.

4.1.1 Standard Mode

A CrypSH client selects this mode of operation when it requires to upload the data in the cloud and retrieve data from the cloud as illustrated in Fig. 3. It consists of three phases, namely, initialisation, encryption and decryption. The detailed description of the three phases is given below.

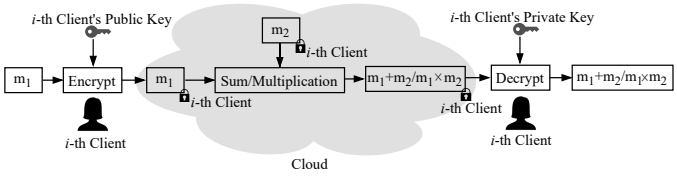


Fig. 3. Standard mode of operation. The data is encrypted and decrypted using i-th client's public and private keys, respectively. Computations on ciphertexts take place in the cloud.

Initialisation Phase: In the initialisation phase, all the CrypSH clients initially register themselves with the ICA. Before starting of the registration process, the ICA publishes its public key through the following process.

- The ICA picks two distinct prime numbers x' and y' such that $x' = 2x + 1$ and $y' = 2y + 1$, where x and y are two distinct numbers.
- We select e as a bilinear map such that $e : Z_n \times Z_n \rightarrow Z_n^*$, where Z_n and Z_n^* are groups of order $n = x'y'$. The ICA computes $\lambda = xy$, chooses a random generator $g \in Z_n$ with order λ. Subsequently, it computes $\eta = g^y \mod n$ and η is the random generator of Z_n.
- The ICA publishes the public key $pk_{ICA} = (n, g, \eta)$ and stores the private key $prk_{ICA} = x$.

After publication of the public key by the ICA, each CrypSH client performs the following steps for registering themselves with the ICA as well as to generate system parameters.

- An i-th client randomly picks two distinct prime numbers p_i' and q_i' such that $p_i' = 2p_i+1$, $q_i' = 2q_i+1$, where p_i and q_i are distinct numbers, and computes $n_i = p_i'q_i'$ and $\lambda_i = p_iq_i$, chooses a random generator $g_i \in Z_{n_i}$ with order λ_i. Subsequently, it computes $\eta_i = g_i^{q_i} \mod n_i$ and η_i is the random generator of Z_{n_i}.
- An i-th client calculates $P_i = g^{p_i'} \mod n$, $Q_i = g^{q_i'} \mod n$ and α_i, where α_i is the current time stamp.
- An i-th client then sends $pk_i = (n_i, g_i, \eta_i, P_i, Q_i, \alpha_i)$ to the ICA as its own public key and stores p_i as the private key prk_i.
- After receiving pk_i from i-th client, the ICA performs the authentication of the i-th client if $n_i = (P_i, Q_i)^x = p_i'q_i'$ holds. If the authentication is successful, the ICA publishes $(n_i, g_i, \eta_i, P_i, Q_i, \alpha_i)$ as the public key of the i-th client, otherwise, discards the registration process of the i-th client.

Encryption Phase: An i-th client in CrypSH executes the following steps to produce the ciphertext c_i of the message m. We assume that the message space consists of integers in the set $\{0, 1, \cdots, T\}$, where T is an integer and $T < q_i$.

- An i-th client first randomly chooses a number $r_i \in Z_{n_i}$.
- Then, i-th client calculates $c_i = E(m) = {g_i}^m {\eta_i}^{r_i} \mod n_i$.
- After determining the ciphertext, c_i, the i-th client stores it in the cloud.

Decryption Phase: During decryption, the i-th client executes the following process on the ciphertext c_i to extract the message m.

- The i-th client first computes $c_i^{p_i} = (g_i{}^m \eta_i{}^{r_i})^{p_i} = (g_i{}^{p_i})^m$.
- The i-th client solves the discrete logarithm of $(g_i{}^{p_i})^m \mod n_i$ with the base $g_i{}^{p_i}$ using the Pollard's lambda algorithm and produces the message $m = D(c_i)$.

Note that the decryption time in the BGN cryptosystem depends on the size of the message space. Typically, the message space in the CrypSH is varied between 32 to 64-bit (see Section 6.2) for efficient decryption. Further, to reduce the decryption time, we precompute a table of powers of $g_i{}^{p_i}$, so that the solution of the discrete logarithm occurs in constant time.

4.1.2 Sharing Mode

As mentioned earlier, CrypSH is based on the BGN cryptosystem, so it supports both addition and multiplication operations on the encrypted data during sharing mode. This enables cryptographic protected sharing of encrypted data, without the need to disclose any private key to the cloud. Our sharing mode consists of two phases, i.e., the token generation and re-encryption, as depicted in Fig. 4. In the first phase, the i-th client generates the re-encryption token for the intended client, say j-th (where $i \neq j$), based on its own private key and the j-th client's public key. The re-encryption token is generated as follows.

- Given the private key p_i of the i-th client and the attributes α_j and g_j of pubic key of the j-th client, we generate the re-encryption token as:

$$Token_{i \to j} = g_j^{\alpha_j / p_i} \in Z_{n_i}.$$

Next, the cloud with access to the $Token_{i \to j}$ performs the re-encryption as follows:

$$c_i^{sh} = (g'_j, g_j^{\alpha_j / p_i}), \tag{1}$$

where $g'_j = e\left(c_i, g_j^{1/p_i}\right) = e\left(c_i, g_j\right)^{1/p_i}$. The j-th client can now decrypt the ciphertext c_i^{sh} with its own private key and the pairing e as:

$$\frac{\left(g_j^{\alpha_j / p_i}\right)^{1/\alpha_j}}{g'_j} = \frac{\left(g_j^{\alpha_j / p_i}\right)^{1/\alpha_j}}{e(c_i, g_j)^{1/p_i}} = c_i^{p_i}.$$

Finally, the j-th client solves the discrete logarithm of c_i^{sh} using the Pollard's lambda algorithm and produces the message m.

In the CrypSH, the re-encryption tokens are unidirectional and non-transitive. Specifically, we design the CrypSH in such a way that using $Token_{i \to j}$ it is only possible to re-encrypt i-th client's ciphertext c_i to j-th client's ciphertext c_i^{sh}. The inverse operation is cryptographically not feasible. In case there is a replay attack, the attacker captures the re-encrypted ciphertext and tries replaying the same to the j-th client during the next round of communication. The j-th client decrypts the re-encrypted ciphertext using the time stamp generated during the current registration phase.

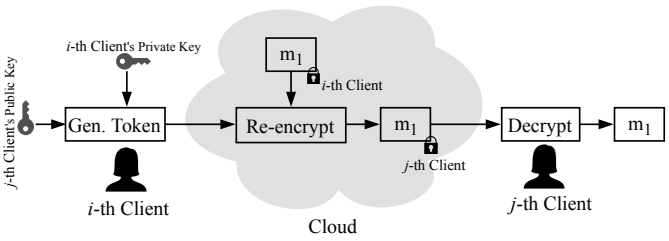


Fig. 4. Sharing mode of operation. The i-th client generates a token based on its own private key and the j-th client's public key. The cloud re-encrypts i-th client's data as j-th client's data. A similar technique is used during group sharing.

Since both the time stamps are not the same, the j-th client understands that a replay attack has taken place and ensures data freshness.

4.2 Key Revocation

In this section, we present the key revocation mechanism, i.e., the key update and in-situ key update.

4.2.1 Key Update

In the CrypSH, we employ a strategy for key update similar to the one proposed in [6]. Specifically, when the IoT devices decide to cancel a data sharing operation, they simply begin using a new key for new data. This prevents previously shared tokens in the cloud from being outdated and prevents new ciphertexts from being re-encrypted with the old token. Once the old key is updated by the new key in the ICA, valid sharing relationships are also updated with the new token in the cloud such that the sharing flow continues. Furthermore, in the CrypSH, a key revocation operation is triggered once the encryption key of an IoT device is compromised.

We consider two cases for the key update: (i) the malicious cloud, and (ii) the semi-honest cloud. In the first case, we leave old data protected with the old key as data is already shared and perhaps cached in the sharing IoT devices. However, in case of the semi-honest cloud, it is necessary to update the encryption key of the old data with the fresh key for steady access. It is thus important to develop a secure scheme that allows the semi-honest cloud to perform the re-keying operation without access to any private key.

4.2.2 In-situ Key Update

During our re-encryption process, we assume that the i-th client has access to both the old and new private keys p_i and p_i^*, respectively. Since the private key is not revealed to any IoT devices, therefore, during our re-encryption process, the private key of the i-th client is not compromised.

In the CrypSH, the re-keying operation is performed on encrypted data stored in the cloud end prior to the sharing operation. The i-th client generates a key update token $\tau = g_i^{p_i / p_i^*}$ and sends it to the cloud. The cloud performs the re-keying operation by modifying c_i^{sh} given in (1) as follows:

$$c_i^{rk} = \tau \left(g'_j, g_j^{\alpha_j / p_i}\right).$$

Once the re-keying operation is finished, all the ciphertexts at the cloud end are now encrypted with the updated key. It is worth noting that, unlike re-encryption operation, re-keying operation is transitive. In particular, we can perform the re-keying operation any number of times on the same encrypted data. It is also worth noting that, as only the i-th client is aware of both the old and new key pairs, so a curious DBA can not obtain any information about the private key from the key update token. However, a malicious DBA could perform frequency based attack [31] by reversing the key update token and disclosing the encrypted ciphertexts using updated key from earlier key. We refer to the readers to [31] for the state-of-the-art solution on frequency based attack. Detailed discussion on the mitigation technique is out of the scope of this work.

4.3 Group Sharing

As mentioned in Section 3.1, data in the CrypSH can be shared either with an IoT device or a group of IoT devices. In this section, we design a sharing authorisation mechanism to enable group sharing operations. To start with, we discuss the structure of the authorisation mechanism. It is important to design a proper structure of the authorisation mechanism since it ensures that a joining IoT device (i) issues the re-encryption token for the genuine group and (ii) retrieves the correct group key. Our sharing authorisation mechanism is based on the access graph [32]. We initially create an access graph for each group of IoT devices, where the root node is the group initiator, as shown in Fig. 5. We identify each group by the group initiator's ID. The IoT devices joining the group form the access graph nodes. Each joining IoT device is registered under the immediate parent node, whereas the root node is registered under the ICA. The access graph allows the group members to access membership information of other members. After forming the access graph, the next important step is: how an IoT device will join the group or leave the group? We discuss the solution in the next subsection.

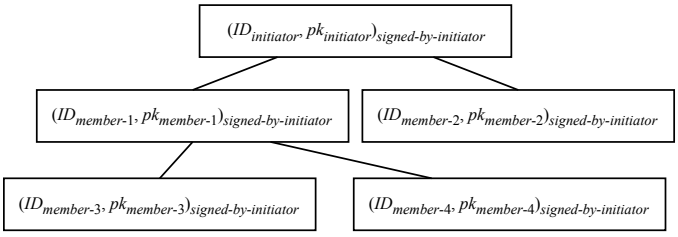


Fig. 5. Example of access graph for group sharing.

4.3.1 Joining Group

We follow a similar strategy during joining of an IoT device in a group as given in [6]. However, unlike [6], the group initiator is responsible for authorising the new group member during the joining phase in the CrypSH. The authorisation is performed in two steps: (i) signing the extended identity (i.e., $ID_{initiator}$ name) and the public key of the new member (verified from the ICA) and (ii) issuing a re-encryption token and storing the re-encryption token in the cloud. For the sake of clear illustration, we here introduce an example to demonstrate our proposed concept. Let us assume that Alice is the initiator of the new group and Bob wants to join the group. Alice first generates the identity, ID_g, the public key, pk_g, and the private key, prk_g, of the group. Alice then generates a re-encryption token, $Token_{g \to b}$, for the group and stores it in the cloud. After receiving a joining message from Bob, Alice first verifies the public key of Bob from the ICA. Upon successful verification, Alice authorises Bob to access several information stored in the cloud related to the group through sharing of the $Token_{g \to b}$. Specifically, the information stored in the cloud is as follows:

$$(ID_g, pk_g, ENC_g(prk_g))_{signed-by-initiator},$$

where $ENC_g(prk_g)$ is the encrypted private key of the group. After authorisation, Bob joins as a new member of the group, ID_g. Bob then provides a re-encryption token, $Token_{b \to g}$, needed for sharing with the group. To issue this re-encryption token and subsequent decrypting of group data, Bob uses the public key, pk_g, and the private key, prk_g, of the group. It is worth noting that Alice's (i.e., group initiator) signature on the key information prevents an attacker in misleading Bob into issuing a token for a malicious group with the same name.

4.3.2 Leaving Group

To leave the group, an IoT device initiates the key revocation mechanism, as presented in Section 4.2.1. Upon completion of the key revocation mechanism, new data are encrypted with the new key. However, older data of the leaving IoT device will remain in the cloud and can still be used by the group members. If the leaving IoT device decides to stop sharing the old data with the group members, then it triggers the in-situ key update mechanism (see Section 4.2.2).

4.4 Homomorphic Operation

In this section, we first present the homomorphic properties of the BGN cryptosystem in Section 4.4.1. We then discuss how the CrypSH performs the homomorphic operations, i.e., addition and multiplication on ciphertexts in Section 4.4.2 and Section 4.4.3, respectively. The homomorphic encryption is a type of encryption scheme, which allows a third party, e.g., cloud to perform certain computable functions on the encrypted data [10]. Unlike the Pilatus and Kryptein, the CrypSH supports an arbitrary number of additions and one multiplication operation, as it considers the BGN cryptosystem as the underlying encryption mechanism.

4.4.1 Homomorphic Properties of BGN

The homomorphic properties of the BGN cryptosystem are as follows:

Homomorphism over Addition: The homomorphic addition of plaintexts m_1 and m_2 using ciphertexts $E(m_1) = c_1$ and $E(m_2) = c_2$ are performed as follows:

$$\begin{aligned} c = c_1 c_2 \eta^r &= (g^{m_1} \eta^{r_1})(g^{m_2} \eta^{r_2}) \eta^r \\ &= g^{m_1 + m_2} \eta^{r_1 + r_2 + r}, \end{aligned}$$

where c is the ciphertext. According to the property, anyone can retrieve the addition of plaintexts m_1 and m_2 by decrypting c.

Homomorphism over Multiplication: The homomorphic multiplication of plaintexts m_1 and m_2 using ciphertexts $E(m_1) = c_1$ and $E(m_2) = c_2$ are performed as follows:

$$\begin{aligned} c = e(c_1, c_2)\eta_1^r &= e(g^{m_1}\eta^{r_1}, g^{m_2}\eta^{r_2})\eta_1^r \\ &= g_1^{m_1 m_2}{\eta_1}^{m_1 r_2 + m_2 r_1 + \tilde{\beta} q_1 r_1 r_2 + r}, \end{aligned}$$

where $\tilde{\beta} \in Z_n$. Here, ciphertext c is uniformly distributed encryption of $m_1 \times m_2$. Thus, any one can retrieve the multiplication of plaintexts m_1 and m_2 by decrypting c.

4.4.2 Addition Operation in the CrypSH

Let i-th client intends to share the addition of plaintexts m_1 and m_2 with the j-th client via the cloud. Initially, i-th client encrypts m_1 and m_2 using the CrypSH (see Section 4.1.1) and the resulting ciphertexts $c_{m1} = E(m_1)$ and $c_{m2} = E(m_2)$ are stored in the cloud. In the next step, the i-th client generates a token, $Token_{i\to j}$, and shares it with the cloud. After receiving the token, the cloud first performs homomorphic addition operation using ciphertexts c_{m1} and c_{m2} as follows:

$$\begin{aligned} c_{m1+m2} = c_{m1}c_{m2}\eta_i^r &= \left(g_i^{m_1}{\eta_i}^{r_1}\right)\left(g_i^{m_2}{\eta_i}^{r_2}\right)\eta_i^r \\ &= g_i^{m_1+m_2}{\eta_i}^{r'}, \end{aligned}$$

where $r' = r_1 + r_2 + r$. Cloud then re-encrypt c_{m1+m2} using $Token_{i\to j}$ as follows: $c_{i\to j}^{add} = (g_j', g_j^{\alpha_j/p_i})$ (see Section 4.1.2) and send $c_{i\to j}^{add}$ to j-th client. Upon receiving $c_{i\to j}^{add}$, j-th client decrypt it with its own private key and the pairing e (see Section 4.1.2), and recovered result will be $m_1 + m_2$.

4.4.3 Multiplication Operation in the CrypSH

Let the i-th client intends to share multiplication of plaintexts m_1 and m_2 with the j-th client via the cloud. Similar to the addition operation, during multiplication, the i-th client first encrypts m_1 and m_2, and the resulting ciphertexts c_{m1} and c_{m2} are stored in the cloud. Also, the i-th client generates a token, $Token_{i\to j}$, and sends it to the cloud. After receiving the token, the cloud performs the homomorphic multiplication operation using ciphertexts c_{m1} and c_{m2} as follows:

$$\begin{aligned} c_{m1\times m2} = e(c_{m1}, c_{m2})\eta_1^r &= e(g_i^{m_1}{\eta_i}^{r_1}, g_i^{m_2}{\eta_i}^{r_2})\eta_1^r \\ &= g_1^{m_1 m_2}{\eta_1}^{m_1 r_2 + m_2 r_1 + \beta q_i r_1 r_2 + r} \\ &= g_1^{m_1 m_2}{\eta_1}^{r'}, \end{aligned}$$

where β ($\in Z_{n_i}$) is a random number and $r' = m_1 r_2 + m_2 r_1 + \beta q_i r_1 r_2 + r$. The cloud then re-encrypts $c_{m1\times m2}$ using the $Token_{i\to j}$ and sends the resultant $c_{i\to j}^{mul}$ to j-th client. After receiving $c_{i\to j}^{mul}$, j-th client decrypts it with its own private key and the pairing e (see Section 4.1.2), and the recovered result will be $m_1 \times m_2$. It is worth noting that $c_{m1\times m2}$ is in the group $Z_{n_i}^*$ instead of Z_{n_i}. Hence, no more homomorphic multiplication operation is allowed in $Z_{n_i}^*$ since there is no pairing from the group $Z_{n_i}^*$. Interestingly, resulting ciphertext $c_{m1\times m2}$ in $Z_{n_i}^*$ still allows any number of homomorphic addition operations.

5 Security Analysis of CrypSH

In this section, we analyse the security of the CrypSH based on the threat model and the security goals presented in Section 3.2 and Section 3.3, respectively. At first, in Section 5.1, we show that the CrypSH is secure against indistinguishability under the chosen plaintext attack [33] and unforgeability under the adaptive chosen message attack [34]. We then analyse the authentication and integrity, confidentiality and data freshness in Section 5.2, Section 5.3 and Section 5.4, respectively.

5.1 Security Theorem

Theorem 1. *The proposed CrypSH is secure against indistinguishability under the chosen plaintext attack, if the BGN cryptosystem is secure against indistinguishability under a chosen plaintext attack.*

Proof. By proof of contradiction, we assume that an adversary $\mathcal{A}$ runs a polynomial time algorithm $\mathcal{B}$ to break the security of the CrypSH with advantage $\epsilon(\tau)$. Particularly, given an instance (n, g, η) of the discrete logarithm problem, $\mathcal{A}$ runs $\mathcal{B}$ to determine $q \in Z_n$ such that $\eta = g^q \mod n$. Given (n, η, e, Z_n, Z_n^*) as input, algorithm $\mathcal{B}$ works as follows:

- The algorithm $\mathcal{B}$ chooses a random generator $g \in Z_n$ and gives algorithm $\mathcal{B}$ the public key $(n, g, \eta, e, Z_n, Z_n^*)$.
- The algorithm $\mathcal{B}$ then generates two messages $m_0, m_1 \in \{0, 1, \cdots, T\}$ to which $\mathcal{A}$ sends with the ciphertext $c_{m_b} = g^{m_b}\eta^{r_{m_b}} \mod n \in Z_n$ for a random $b \xleftarrow{R} \{0, 1\}$ and random $r_{m_b} \xleftarrow{R} \{0, 1, \cdots, n-1\}$.
- Finally, the algorithm $\mathcal{B}$ outputs its guess b' for actual b. If $b' = b$, $\mathcal{A}$ successfully retrieves the plaintext message; otherwise, $b' \neq b$, when $\mathcal{A}$ fails to retrieve the plaintext message.

It is worth mentioning that when η is uniform in Z_n, the challenged ciphertext c_{m_b} is uniformly distributed in Z_n and is independent of the bit b. Thus, in this case, the probability that $\mathcal{A}$ correctly guesses the plaintext message is $1/2$ [11]. Conversely, when η is uniform in the q-subgroup of Z_n, the probability that $\mathcal{A}$ correctly guesses the plaintext message is greater than $1/2 + \epsilon(\tau)$ [11]. At the beginning of the proof, we assume that $\mathcal{A}$ can break the security of the CrypSH with advantage $\epsilon(\tau)$, which is contradicting the calculated probability. Hence, the CrypSH is semantically secure against the chosen plaintext attack.

□

Theorem 2. *The proposed CrypSH is semantically secure against the adaptive chosen message attack, if the discrete logarithm problem is hard.*

Proof. In Theorem 1, we show the steps that an adversary $\mathcal{A}$ follows to guess the plaintext message. The correct guessing is only possible when $\mathcal{A}$ can solve the discrete logarithm problem. We show that the probability of correct guesses of the plaintext message is $1/2$ when η is uniform in Z_n. Whereas the probability of correct guesses of the plaintext message is $1/2 + \epsilon(\tau)$ when η is uniform in the q-subgroup

of Z_n. These contradict the hardness of the discrete logarithm problem [28]. Thus, the CrypSH is semantically secure against the chosen message attack.

□

5.2 Authentication and Integrity

To ensure that the shared data is coming from a reliable source and without any modification, the CrypSH should provide the authenticity of the IoT devices and integrity of data. In the CrypSH, authenticity of the IoT devices is verified in two ways. First, the ICA verifies the authenticity of the IoT devices. Second, during sharing mode, an IoT device verifies the authentication of another IoT device. Whereas the cloud and the IoT device collaboratively check the integrity of data. In the CrypSH, during the initialisation phase (Section 4.1), first the ICA publishes its public key, (n, g, η) for IoT devices. An i-th IoT device in turn registers itself by sending its own public key, $(n_i, g_i, \eta_i, P_i, Q_i, \alpha_i)$. After receiving the public key of the i-th IoT device, the ICA verifies if $(P_i, Q_i)^x = p_i' q_i'$ holds for $i \in \{1, \ldots, N\}$. If verification is successful, then the i-th IoT device is allowed to participate in the standard and sharing modes of operations. Otherwise, the ICA aborts the verification phase of the i-th IoT device. In summary, the CrypSH allows only the authentic IoT devices to register themselves in the ICA.

The CrypSH uses the BGN as the underlying cryptographic scheme, where a plaintext is encrypted using the private key of a particular IoT device and the resultant ciphertext is stored in the cloud. Theorem 2 shows that no adversary can produce the plaintext by solving the discrete logarithm problem. Hence, any tampering of the plaintext can be detected by the authentic IoT device. Therefore, the designed CrypSH can ensure the integrity of the message.

5.3 Confidentiality

An adversary, say $\mathcal{A}$, can access the public key of the ICA and the i-th IoT device, i.e., (n, g, η) and $(n_i, g_i, \eta_i, P_i, Q_i, \alpha_i)$, respectively. In addition, an adversary can access the ciphertext of the i-th IoT device, i.e., c_i. However, based on Theorem 1, an adversary cannot remove ${\eta_i}^{r_i} \mod n_i$ from $c_i = {g_i}^m {\eta_i}^{r_i} \mod n_i$. Even if it is possible to extract ${\eta_i}^{r_i} \mod n_i$, but due to the hardness of solving the discrete logarithm problem (see Theorem 2), it is thus quite impossible to extract an IoT data m without employing the private key, i.e., p_i. Therefore, the designed CrypSH is capable of providing data confidentiality.

5.4 Data freshness

We intend to protect the CrypSH from replay attacks, where the attacker tries re-transmitting a previously-transmitted valid packet [35]. We prevent replay attacks by ensuring data freshness. We use a time stamp to check the data freshness. In standard mode, during the registration phase, only the authentic IoT devices are allowed to register themselves with the ICA by sending the public key, $pk_i = (n_i, g_i, \eta_i, P_i, Q_i, \alpha_i)$. In the public key, α_i attribute represents the time stamp of the i-th IoT device, which is used for checking data freshness. During the sharing mode, an i-th IoT device generates a re-encryption token based on the current time stamp of the intended IoT device accessed from the ICA. The cloud uses the token to re-encrypt the encrypted shared packet. Upon receiving the encrypted shared packet, the receiver verifies the time stamp of the re-encrypted shared packet with its own time stamp. If the verification is unsuccessful, the receiver realises that a replay attack has been performed and it discards the packet.

6 Experimental Evaluation

In this section, we evaluate the performance of our proposed CrypSH under practical cloud-IoT application scenarios. We compare the CrypSH with our main competitor, the Pilatus [6].

6.1 Experimental Setup

Our experimental evaluation setup consists of the client engine running with the Contiki 3.0 [36], which is an open source operating system for the IoT devices. We run our client engine on an Ubuntu 14.04 with 8 GB of RAM in a Dell XPS laptop with Intel Core i7 processor running at 2.4 GHz. In our experiments, we considered an IoT device with similar specifications as Texas Instrument SensorTag [37]. Conversely, our cloud server resides on a normal desktop, i.e., AMD Athlon workstation with 8 GB RAM, running MySQL database on 64-bit Ubuntu 14.04. We connect both the client engine and the cloud server via the internet with an average transmission delay of 10 ms [6]. Similar to [6], our experimental results are based on single-threaded execution times, as most of the existing libraries do not have an efficient multi-threaded implementation. To make the experiment results more realistic, we use a publicly available state-of-the-art HElib library [38]. Further, we use the THEW database [39] as a source of real data. THEW is a large collection of 24 hours anonymized heartbeat recordings of real patients. For our experiment, we extracted approximately 30 minutes to 1 hour heartbeat data for 50 patients (roughly 4 K data records for each patient).

For implementing our CrypSH, we select two prime numbers $x' = 2x + 1$ and $y' = 2y + 1$, where x and y are also prime numbers of size 128-bit. Next, we compute n, where $n = x'y'$. For the BGN cryptosystem, we select e as a bilinear map such that $e : Z_n \times Z_n \rightarrow Z_n^*$ defined on the super-singular elliptic curve $E : v^2 = u^3 + au + b$, where a and b are chosen such that $4a^3 + 27b^2 \neq 0$.

6.2 Performance Metrics

Evaluation of the performance of CrypSH and Pilatus is divided into several stages. First, we evaluate the computation overhead of the schemes. Here, the computation overhead is measured using the three metrics: (i) the time required for key generation, (ii) the time required to upload encrypted data into the cloud and (iii) the time required for encryption/decryption operation. Second, we evaluate the storage overhead of both the competing schemes. To measure the storage overhead, we consider ciphertext size as a performance metric. Additionally, we measure the energy overhead, system throughput, data freshness and end-to-end delay for both schemes. We define the throughput as the rate of operation on encrypted data performed in the

cloud. We are the first to measure the data freshness for the cloud based IoT systems. Similar to [35], we define the data freshness as the percentage of the number of encrypted data received by an IoT device containing current data out of the total number of data sent by the cloud during one communication cycle. Following the recommendation by NIST [40], we use 128-bit security for encrypting data in both the CrypSH and Pilatus. We generate the plaintext inputs randomly and uniformly for two integer inputs, viz., 32-bit and 64-bit sizes. Extensive simulation is performed with a 95% Confidence Interval (CI) and average results of 100 independent runs are taken in plotting the results.

6.3 Computation Overhead

In this section, we discuss the performance of both the CrypSH and Pilatus in terms of computation overhead. Here, we measure the computation overhead using three metrics, as mentioned in Section 6.2.

6.3.1 Key Generation

We first evaluate the average setup time required to generate the keying materials for both the CrypSH and Pilatus, the results of which are shown in Table 2. Table 2 shows that irrespective of the mode of operations, the CrypSH requires less setup time compared to the Pilatus. For example, in standard mode of operation, the CrypSH requires 0.39 ms to generate the keying material, whereas the Pilatus requires 0.57 ms. Similar kind of results are also observed during the key setup and token generation phases of sharing mode of operation. The probable reason for 0.18 ms additional time is that the Pilatus relies on several optimized algorithms. Also, the key generation operation does not take place frequently in contrast to the encryption and decryption operations.

TABLE 2
Average setup time for 128-bit security level

Scheme	Mode of operation	Setup time [ms]	CI [ms]
Pilatus	Standard	0.57	[0.53, 0.60]
	Sharing: Key setup	6.48	[5.99, 6.97]
	Sharing: Token gen.	4.41	[4.10, 4.72]
CrypSH	Standard	0.39	[0.36, 0.42]
	Sharing: Key setup	4.02	[3.78, 4.26]
	Sharing: Token gen.	2.61	[2.41, 2.85]

6.3.2 Cloud Database

Table 3 shows the total time required to upload the encrypted real time heartbeat data in the cloud database for both the competing schemes. We notice that the CrypSH requires less time than the Pilatus for both the modes of operations. Overall, the sharing mode is slower than the standard mode. In particular, during the standard mode of operation, the Pilatus is slower than the CrypSH by a factor of 2.24. In contrast, in the sharing mode of operation, the Pilatus is slower than the CrypSH by a factor of 2. This indicates that the CrypSH is faster than the Pilatus in both the modes of operations.

TABLE 3
Average time to upload data into the cloud database

Scheme	Mode of operation	Time [ms]	CI [ms]
Pilatus	Standard	4.16	[3.66, 4.68]
	Sharing: Key setup	13.08	[11.52, 14.65]
	Sharing: Token gen.	-	-
CrypSH	Standard	2.05	[1.84, 2.26]
	Sharing: Key setup	6.52	[5.85, 7.17]
	Sharing: Token gen.	-	-

6.3.3 Encryption and Decryption Times

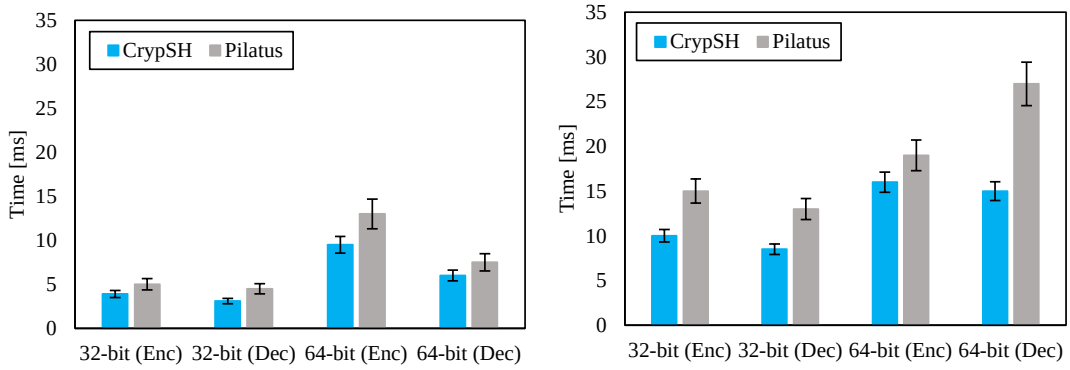
Figure 6(a) shows the performance of the standard mode as discussed in Section 4.1.1. Whereas, Fig. 6(b) shows the performance of the sharing mode as discussed in Section 4.1.2. While plotting Fig. 6, we consider 32-bit and 64-bit input integer sizes, and 128-bit security level. Further, we consider the encryption and decryption operations involved in both the modes. We observe from Figure 6 that the CrypSH outperforms the Pilatus for both encryption time and decryption time. Specifically, during the standard mode, the CrypSH requires 34% and 10% less time for encryption and decryption operations, respectively, compared to the Pilatus. Similarly, during the sharing mode Figure 6(b), the time required for encryption and decryption operations is 34% and 14.81% less in the CrypSH compared to the Pilatus. Further, irrespective of the schemes, we observe that the time required for encryption and decryption operations is more in the sharing mode than the standard mode. More importantly, irrespective of the modes, the time required for decryption operation is less than the encryption operation in the CrypSH. The reason is that, we precompute a table of powers of $g_i{}^{p_i}$ to speed up the execution of the discrete logarithm algorithm for the i-th client

6.4 Storage Overhead

We measure the storage overhead of both the CrypSH and the Pilatus considering the ciphertext size as performance metric. Figure 7 shows the ciphertext sizes of CrypSH and Pilatus for the standard and sharing modes, respectively. We notice from the plots of the CrypSH that during the standard mode of operation, each plain text input needs a different number of congruences, resulting in a ciphertext size between 140 and 190 bytes. Whereas, for sharing mode of operation, the ciphertext size varies between 500 and 700 bytes. On the contrary, in the Pilatus, during standard and sharing modes, the ciphertext sizes vary between 150 and 220 bytes, and 600 and 900 bytes, respectively. It means that the performance of the CrypSH is improved by reducing the ciphertext sizes to 13% and 25% compared to the Pilatus during the standard and sharing modes, respectively.

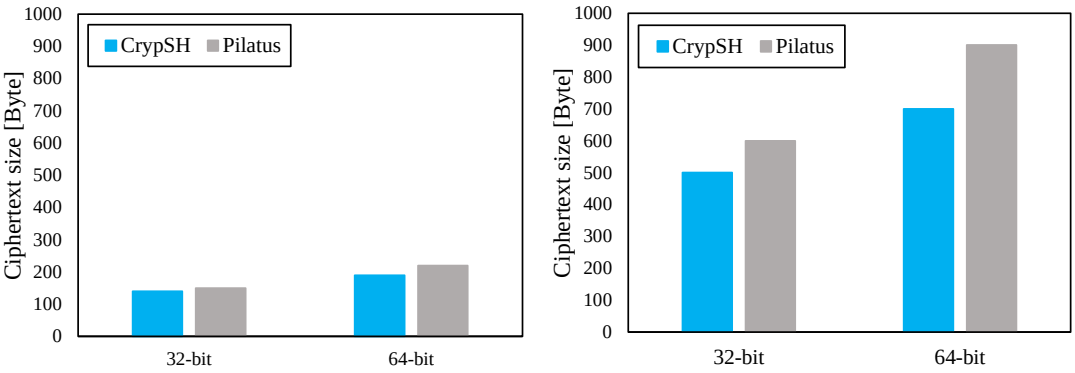
6.5 Energy Overhead

In our experimental setup, we use the values of our required parameters for SensorTag as described in [37]. For example, in receiving mode and transmitting mode, we use the current drawn as 6 mA and 6.1 mA, respectively. Table 4 shows the energy measurements of the CrypSH and Pilatus for 32-bit and 64-bit input message sizes. The



(a) Standard mode (b) Sharing mode

Fig. 6. Encryption and decryption time evaluation. The client engine performs encryption and decryption of data.



(a) Standard mode (b) Sharing mode

Fig. 7. Storage overhead comparison between CrypSH and Pilatus under different input lengths.

TABLE 4
Energy consumption of data protection component

Scheme	32-bit Integer input	CI [mJ]	64-bit Integer input	CI [mJ]
Pilatus	12.4 mJ	[11.2713.62]	25.16 mJ	[22.87, 27.94]
CrypSH	9.5 mJ	[8.79, 10.44]	19.81 mJ	[18.34, 21.77]

results show that with the increase in input message sizes, the energy overhead increases almost linearly irrespective of the schemes. It is obvious because communication overhead created with increasing input message sizes results in increase in energy consumption. From the energy overhead perspective, the CrypSH outperforms the Pilatus for both the input message sizes. Specifically, the CrypSH consumes 30.52% and 27.05% less energy than the Pilatus for 32-bit and 64-bit input message sizes, respectively. The reason for more energy consumption is due to the fact that the Pilatus relies on additional steps for solving the Chinese Remainder Theorem and the Baby-Step-Giant-Step algorithm [6] during encryption and decryption processes.

6.6 Throughput

Figure 8 shows the throughput of the cloud running on a normal desktop when executing SUM SQL queries, either over plaintext or encrypted data. During measuring of throughput, we generate variable length queries like summands and measure the time to process each, and calculate the average throughput of the cloud for a single connection. Further, for measuring throughput, we initiate the queries locally from the IoT device to the cloud over the Internet.

The results show that with the increase in the number of queries, plaintext SUM operations are roughly 2 times faster compared to the CrypSH and Pilatus. It is due to the fact that the plaintext SUM operations are performed parallelly. However, during the standard SUM operation, the throughput of the CrypSH is 16% more than the Pilatus. Similar kind of results is also found in pre-sharing SUM and post-sharing SUM operations. Specifically, during the pre-sharing SUM operations, the throughput of the CrypSH improves by 15% than the Pilatus. Whereas in the post-sharing SUM operations, it is 20% more in the CrypSH compared to the Pilatus.

6.7 Data Freshness

Figure 9 presents the results of data freshness. The experimental results are collected while the normal operations are carried out by the IoT devices and the cloud, i.e., irrespective of the standard mode and the sharing mode. While evaluating data freshness, we varied the number of malicious IoT devices from 2 to 10. The plots show that data freshness decreases with the increase in malicious nodes. Particularly, in presence of 2 malicious IoT devices, the average data freshness is 99.18% , whereas its values are 98.20% and 97.11% for 5 and 10 malicious IoT devices, respectively. It is

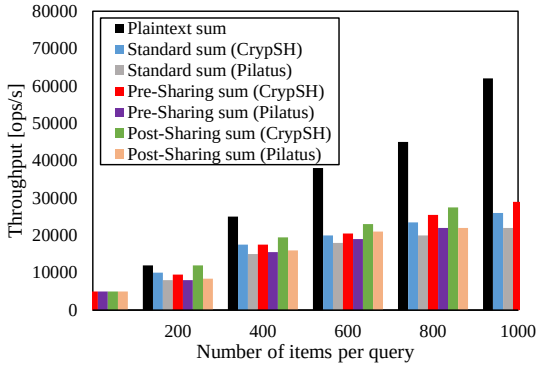


Fig. 8. Throughput evaluation under standard and sharing modes, with our cloud engine running on a desktop. Pre- and post-sharing sum refer to SUM queries before and after sharing.

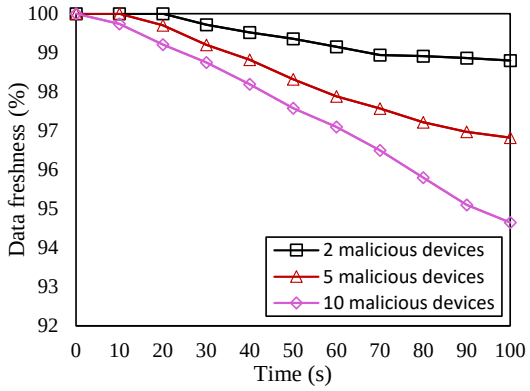


Fig. 9. Data freshness evaluation under different numbers of malicious IoT devices under the combined mode of operations.

due to the fact that as the number of malicious IoT devices increases, the number of attempts made by them to send data repeatedly increase resulting in a decrease of number of received packets containing current data.

6.8 End-to-End Delay

Figure 10 presents the end-to-end delay for varying SUM queries. For both the schemes, plots show that the end-to-end delay increases with the increase in number of items per query. For lower range SUM queries (i.e., <200), we observe that the end-to-end delay of the standard and sharing modes are close to queries over plaintext for both the CrypSH and Pilatus. However, for larger ranges (i.e., >600), the end-to-end delay increases abruptly. For example, in the CrypSH, the average end-to-end delay of the standard and sharing modes over plaintext increase by a factor of 1.95 and 2.22, respectively. Whereas in the Pilatus, it increases by a factor of 2.24 and 2.44. In summary, end-to-end delay of the CrypSH is 14.87% and 11.07% less than that of the Pilatus. Note that to ensure a seamless interaction among the IoT devices with encrypted data, the end-to-end delay should be <1 s [6].

7 Conclusion

In this paper, we present CrypSH, a new scheme tailored for the cloud based IoT systems. The CrypSH provides an

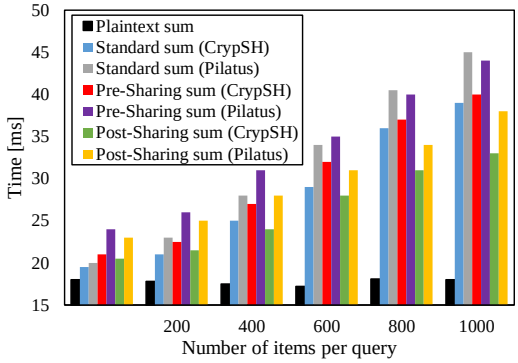


Fig. 10. End-to-end delay evaluation under standard and sharing modes, with our cloud engine running on a desktop. Pre- and post-sharing sum refer to SUM queries before and after sharing.

efficient, and highest level of security and reliability from the three significant threats, i.e., the curious DBAs, eavesdropping from the communication channel and arbitrary compromises of the cloud server confronting cloud applications. Our scheme leverages cryptographic primitives that allow computation on encrypted data without disclosing any secret keys to the cloud. Also, our scheme processes queries on encrypted data and re-encrypts it for sharing. In the CrypSH, sharing mechanism comes with cryptographic guarantees and possibility of sharing revocation at any time. To achieve this, unlike other encryption schemes, the CrypSH takes the BGN cryptosystem as the underlying encryption mechanism. The evaluation results show that the CrypSH outperforms the state-of-the-art approach, Pilatus, without compromising network performance metrics such as throughput and data freshness. Particularly, the CrypSH is 34% more computationally faster, requires 25% less storage and provides 15% more throughput than the Pilatus. In future, we envision to devise a mechanism for parallelising the homomorphic addition in the cloud.

Acknowledgment

This work was supported by the European Commission under the Horizon 2020 Programme (H2020), as part of the LOCARD project (Grant Agreement no. 832735).

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[2] J. Gong, X. D. Yang, and P. Irani, "WristWhirl: One-handed continuous smartwatch input using wrist gestures," in *Proc. of 29th Annual Symposium on User Interface Software and Technology (UIST)*, 2016, pp. 861–872.

[3] A. Alanwar, Y. Shoukry, S. Chakraborty, P. Martin, P. Tabuada, and M. Srivastava, "PrOLoc: Resilient localization with private observers using partial homomorphic encryption," in *Proc. of 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2017, pp. 41–52.

[4] Ava, "Fertility Tracking Bracelet," online, accessed April 18, 2018, https://www.avawomen.com/.

[5] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE Symposium on Security and Privacy*, 2000, pp. 44–55.

[6] H. Shafagh, A. Hithnawi, L. Burkhalter, P. Fischli, and S. Duquennoy, "Secure sharing of partially homomorphic encrypted iot data," in *Proc. of 15th ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, 2017, pp. 1–14.

[7] W. Xue, C. Luo, G. Lan, R. Rana, W. Hu, and A. Seneviratne, "Kryptein: A compressive-sensing-based encryption scheme for the internet of things," in *Proc. of 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2017, pp. 169–180.

[8] H. Shafagh, A. Hithnawi, A. Dröscher, S. Duquennoy, and W. Hu, "Talos: Encrypted query processing for the internet of things," in *Proc. of 13th ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, 2015, pp. 197–210.

[9] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proc. of 23rd ACM Symposium on Operating Systems Principles (SOSP)*, 2011, pp. 85–100.

[10] A. Acar, H. Aksu, A. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–35, 2018.

[11] D. Boneh, E. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. of Theory of Cryptography Conference (TCC)*, vol. LNCS-3378, 2015, pp. 325–341.

[12] D. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[13] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. of EUROCRYPT*, vol. LNCS-1592, 1999, pp. 223–238.

[14] S. Halder and M. Conti, "Don't hesitate to share! a novel iot data protection scheme based on bgn cryptosystem," in *Proc. of 34th ACM Symposium on Applied Computing (SAC)*, 2019, pp. 282–289.

[15] W. Henecka, A. Sadeghi, T. Schneider, and I. Wehrenberg, "TASTY: Tool for automating secure two-party computations," in *Proc. of 17th ACM Conference on Computer and Communications Security (CCS)*, 2010, pp. 451–462.

[16] K. Liang, C. Su, J. Chen, and J. Liu, "Efficient multi-function data sharing and searching mechanism for cloud-based encrypted data," in *Proc. of 11th ACM AsiaCCS*, 2016, pp. 83–94.

[17] Q. Wang, K. Ren, M. Du, Q. Li, and A. Mohaisen, "SecGDB: Graph encryption for exact shortest distance queries with efficient updates," in *Proc. of Financial Cryptography and Data Security*, 2017, pp. 79–97.

[18] L. Wang, Z. Yang, and X. Song, "SHAMC: A secure and highly available database system in multi-cloud environment," *Future Generation Computer Systems*, vol. 105, pp. 873–883, 2020.

[19] R. Kotamsetty and M. Govindarasu, "Adaptive latency-aware query processing on encrypted data for the internet of things," in *Proc. of 25th Intl Conf on Computer Communication and Networks (ICCCN)*, 2016, pp. 1–7.

[20] J. Y. Kim, W. Hu, D. Sarkar, and S. Jha, "ESIoT: Enabling secure management of the internet of things," in *Proc. of 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2017, pp. 219–229.

[21] A. Liu and P. Ning, "TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks," in *Proc. of 7th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2008, pp. 245–256.

[22] P. Szczechowiak, L. Oliveira, M. Scott, M. Collier, and R. Dahab, "NanoECC: Testing the limits of elliptic curve cryptography in sensor networks," in *Proc. of EWSN*, vol. LNCS-4913, 2008, pp. 305–320.

[23] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multiey fully homomorphic encryption," in *Proc. of 44th Annual ACM Symposium on Theory of Computing (STOC)*, 2012, pp. 1219–1234.

[24] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. Wu, "Private database queries using somewhat homomorphic encryption," in *Proc. of Applied Cryptography and Network Security (ACNS)*, vol. LNCS-7954, 2013, pp. 102–118.

[25] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy, "Towards blockchain-based auditable storage and sharing of iot data," in *Proc. of Cloud Computing Security Workshop*, 2017, pp. 45–50.

[26] T. Okamoto and S. Uchiyama, "A new public-key cryptosystem as secure as factoring," in *Proc. of Eurocrypt*, vol. LNCS-1403, 1998, p. 308–318.

[27] J. Shen, T. Zhou, X. Chen, J. Li, and W. Susilo, "Anonymous and traceable group data sharing in cloud computing," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 4, pp. 912–925, 2018.

[28] J. Katz, A. Menezes, P. Van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC press, 1996.

[29] Keybase, "Publicly Auditable Proofs of Identity," online, accessed April 18, 2018, https://www.keybase.io/.

[30] K. Xue, Y. Xue, J. Hong, W. Li, H. Yue, D. S. Wei, and P. Hong, "Raac: Robust and auditable access control with multiple attribute authorities for public cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 953–967, 2017.

[31] Q.-C. To, B. Nguyen, and P. Pucheral, "Private and scalable execution of sql aggregates on a secure decentralized architecture," *ACM Transactions on Database Systems (TODS)*, vol. 41, no. 3, pp. 1–43, 2016.

[32] R. A. Popa and N. Zeldovich, "Multi-key searchable encryption," *IACR Cryptology ePrint Archive*, vol. 508, pp. 1–18, 2013.

[33] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of computer and system sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[34] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, 1988.

[35] A. Ghosal, S. Halder, S. Sur, A. Dan, and S. DasBit, "Ensuring basic security and preventing replay attack in a query processing application domain in wsn," in *Proc. of International Conference on Computational Science and Its Applications (ICCSA)*, vol. LNCS-6018, 2010, pp. 321–335.

[36] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki: A lightweight and flexible operating system for tiny networked sensors," in *Proc. of 29th Annual IEEE International Conference on Local Computer Networks (LCN)*, 2004, pp. 455–462.

[37] Texas Instrument, "SensorTag," online, accessed December 17, 2017, http://www.ti.com/lit/ds/symlink/cc2650.pdf.

[38] Halevi, S. and Shoup, V., "HElib library," online, accessed November 06, 2017, 2018, https://www.github.com/shaih/HElib.

[39] J. Couderc, "The telemetric and holter ecg warehouse initiative (thew): A data repository for the design, implementation and validation of ecg-related technologies," in *Proc. of IEEE International Conference on Engineering in Medicine and Biology Society (EMBC)*, 2010, pp. 6252–6255.

[40] E. Barker and A. Roginsky, "Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths," in *NIST Special Publication*, vol. 800, 2011, p. 131A.