Towards A Compressive-Sensing-Based Lightweight Encryption Scheme for the Internet of Things

Wanli Xue, Chengwen Luo, Yiran Shen, Rajib Rana, Guohao Lan, *Member, IEEE,* Sanjay Jha, Aruna Seneviratne, and Wen Hu, *Senior Member, IEEE*

Abstract—Internet of Things (IoT) is flourishing and has penetrated deeply into people's daily life. With the seamless connection to the physical world, IoT provides tremendous opportunities to a wide range of applications. However, potential risks exist when the IoT system collects sensor data and uploads it to the Cloud. The leakage of private data can be severe with curious database administrator or malicious hackers who compromise the Cloud. In this work, we propose Kryptein, a compressive-sensing-based lightweight encryption scheme for Cloud-enabled IoT systems to secure the interaction between the IoT devices and the Cloud. Kryptein supports random compressed encryption, statistical decryption, and accurate raw data decryption. According to our evaluation based on two real datasets, Kryptein provides strong protection to the data. It is 250 times faster than other state-of-the-art systems and incurs 120 times less energy consumption. The performance of Kryptein is also measured on off-the-shelf IoT devices, and the result shows Kryptein can run efficiently on IoT devices. After comparing with other state-of-the-art lightweight ciphers on IoT (Simon and Speck), IoT system with Kryptein is expected to have a much more longevity with about 35% extended lifetime. Further, experiments illustrated IoT data variance will not affect Kryptein's accuracy in a long term usage, and Krpytein is also able to support basic analytics tasks like machine learning (e.g., classification).

Index Terms—Compressive Sensing; Security; Encryption; Internet of Things.

1 Introduction

In recent years, the proliferation of sensor-equipped smartphones, wearable devices, and a variety of connected smart things have enabled a new spectrum of novel applications for Internet of Things (IoT) [1] and penetrated into people's daily life. For example, wearable devices such as wrist bands and smart watches record users' daily activities and health-related statistics: they monitor heart rate series to infer the health status of the wearers, detect potential diseases (arrhythmia, etc.), and provide customized health recommendations. GPS trackers record users' walking or running trajectories for daily logging or social interaction purposes. Many of these new types of sensing applications enabled by the IoT technologies have substantially changed the way people perceive information about themselves and the surrounding environment. We envision them to continually gain their popularity with the blooming IoT.

On the other hand, the sensing applications produce a large amount of sensor data that needs to be processed and stored. Storage on the typical IoT devices such as smartphones, wearables, and static sensor nodes are usually limited, the data storage and processing functionalities have been largely shifted to the Cloud or edge [2]. Consequently , the Cloud[1] provides data storage for the continuously generated sensor data and handles online queries on the data such as retrieving the raw sensor values or computing statistics. The Cloud-enabled IoT has released the storage limitation of typical IoT devices and eased the application deployment. However, it also poses severe security risks to the IoT systems. Since the privately collected sensor data is uploaded to the Cloud, without appropriate protection the Cloud services could lead to two major threats: (1) Privacy leakage to the curious database administrator (DBA). (2) Arbitrary threats due to the Cloud/edge being compromised.

To this end, lightweight encryption schemes for IoT are significantly important not only for Cloud computing, but especially required for achieving edge computing and edge intelligence. Booming IoT services (storage, query, analytics, etc.) seem long for more responsive and secure Cloud/edge services.

Encrypted Query Processing. Stimulated by the security

- *W. Xue, S. Jha and W. Hu are with the Schoole of Computer Science and Engineering, University of New South Wales and Cyber Security Cooperative Research Centre, Sydney, NSW, 2032, Australia. E-mail: (wanli.xue,sanjay.jha)@cybersecuritycrc.org.au,wen.hu@unsw.edu.au*
- *C. Luo is with the College of Computer Science and Sftware Engineering, Shenzhen University, Shenzhen 518060, China. E-mail: chengwen@szu.edu.cn*
- *Y. Shen is with Data61, CSIRO, Pullenvale, QLD, 4069, Australia. E-mail: yiran.shen@csiro.au*
- *R. Rana is with Institute of Resilient Regions, University of Southern Queensland, Springfield, QLD 4300, Australia. E-mail: Rajib.Rana@usq.edu.au*
- *G. Lan is with Department of Electrical and Computer Engineering, Duke University, Durham, NC 27710, USA . E-mail: guohao.lan@duke.edu*
- *A. Seneviratne is with the School of Electrical Engineering and Telecommunications, University of New South Wales and Cyber Security Cooperative Research Centre, Sydney, NSW, 2032, Australia. E-mail: a.seneviratne@unsw.edu.au*

Manuscript received Sep 19, 2019; revised Nov 26, 2019. (Corresponding author: ChengWen Luo.) The original paper is published as "Kryptein: a compressive-sensing-based encryption scheme for the internet of things." in IPSN 2017, this work is the extension.

1. The Cloud and edge is interchangeable in this paper's scope.

problem of the Cloud-enabled IoT systems, many research work has been done to improve the security and privacy of IoT systems. CryptDB [3] is an online encryption system that provides practical and provable confidentiality for query processing on the Cloud without requiring modification to the Cloud databases. It allows executing SQL queries (e.g., insertion, selection, etc.) over encrypted data using a collection of SQL-aware encryption schemes. However, since CryptDB was proposed mainly as support of web applications, the computation and energy efficiency is not the focus of the work. The recent improvement Talos [4] extends CryptDB and is specially designed for IoT systems to improve query efficiency and reduce energy consumption through several optimization techniques that accelerate partial homomorphic encryptions and order preserving encryptions. However, though the computation can be reduced using the optimization techniques proposed by Talos, we will show in this paper that system security can be achieved while performance can be improved significantly.

Kryptein: a new Compressive-Sensing-Based Scheme for Energy-Efficient Encrypted Query Processing in IoT. In this paper, we propose Kryptein[2], a new compressive sensing [5] based encryption scheme for secure query processing in IoT systems. Through several novel techniques, e.g., *random compressed encryption*, *statistical computation over cipher*, and *decryption on demand*, Kryptein provides secure data insertion, accurate raw data decryption, and efficient statistics computation in Cloud-enabled IoT systems. Since IoT systems have their unique characteristics, we are facing several challenges in designing the secure Cloud-enabled IoT systems. (1) The communication costs need to be minimized when uploading the encrypted sensor data to the Cloud; (2) Some frequent queries on data statistics such as finding the distributions can be computed over the cipher texts directly; (3) The system should support **decryption-on-demand (DoD)**, that is, different types of functionalities can be supported using different types of decryption. Incorporating these properties, Kryptein has the following features that make it a promising encrypted query processing scheme for Cloud-enabled IoT systems:

Random compressed encryption. Unlike the cryptographic based approach such as CryptDB and Talos, Kryptein adopts compressive sensing as the underlying encryption mechanism seamlessly integrating data encryption and data compression during data processing. As a result, the size of the cipher texts are significantly reduced as well as the data uploading cost and data storage cost. The light-weighted encryption process also reduces the computation cost on the IoT device side achieving better energy efficiency for constrained IoT devices.

Enabling efficient statistical computation over cipher. To avoid expensive raw data decryption on IoT devices for frequent queries on data statistics such as finding the average(AVG), summation(SUM) and standard deviation(SD), Kryptein supports *statistical computation over cipher*. Kryptein will efficiently compute the statistics over the cipher without requiring to decrypt the raw data values.

Decryption on demand. Since different data might require different computations for different applications, Kryptein adopts the onion-structure [3] and provides three layers of decryption. The decryption on demand (DoD) provides flexibilities to the IoT applications and only decrypts data when necessary.

Contribution. In summary, this paper makes the following contributions:

- We design and evaluate Kryptein, a compressive sensing based encryption scheme for query processing in Cloud-enabled IoT systems.To the best of our knowledge, it is the first compressive-sensing-based encryption scheme that achieves energy-efficient secure data processing for the Cloud-enabled IoT.
- We propose the DoD mechanism, which supports multi-layer decryption including statistical computation over cipher and raw data decryption. The DoD mechanism avoids unnecessary decryption operations and provides the efficient computation of statistics and raw data decryption based on application requirements.
- We implement the system prototype and evaluate its real-world performance.The results show that Kryptein is 250 times faster than the state-of-the-art competing system (Talos), while reducing the energy consumption by 99.16%). Furthermore, Kryptein reduces storage requirement by 70% with little performance (accuracy) sacrifice. Moreover, the further comparison with the lightweight ciphers targeting IoT applications (Simon and Speck) shows Kryptein (serving as cipher) can extend IoT devices' lifetime for about 35%.
- At last, the usability about Kryptein is evaluated. The accuracy of Kryptein is investigated in a long-term usage (about half-year), which proves Kryptein can remain a steady accuracy for long-term usage. Besides, a real-world analytics task, face recognition is tested with Kryptein with accuracy and privacy as the evaluation metrics. The preliminary result suggests Kryptein is feasible to support basic IoT analytics tasks like classification.

2 System Overview

Fig. 1 shows Kryptein's architecture and its workflow. Kryptein works by compressing and encrypting all data entries in the trusted **client side** (Alice); after that, only send the *CS-encrypted* (compressed and encrypted) data (cipher) to the **server side** and delete the original data locally. With the key consist of pre-learned dictionary (sparse representation dictionary learned from the user's history data) and random perturbation matrix (generated based on user's password), new streaming IoT data can be CS-encrypted and uploaded to server efficiently. When information such as SUM, AVG and SD is requested (by family doctor for instance), the server only needs sending back the cipher text to trusted IoT devices (Bob), which own the partial correct keys. On those IoT devices, SUM, AVG and SD can be calculated directly over cipher with very little cost. With partial key, IoT devices can only calculate the statistical information rather than reconstruct the raw data. When the raw data is request (by Alice), the cipher will be decrypted to intermediate data (high computation cost) on server and then converted back to raw data (low computation cost)

2. Kryptein is a Latin verbal adjective and means "to hide".

only on trusted devices (Alice's). During the whole process, only the trusted devices (who own the entire correct key) can reconstruct the raw data. We assume any user doesn't own an appropriate key as attacker (Eve), and the server is honest-but-curious. Even an attacker (or eavesdropper) collected the intermediate data or cipher cannot reconstruct to the raw data or calculate the statistical information. The server never knows the meaning of the data or its calculation results.

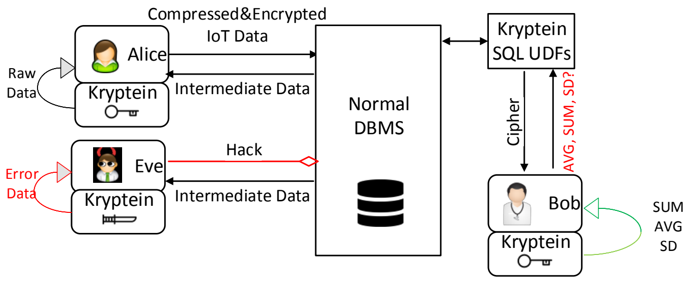


Fig. 1: Workflow and architecture: Alice is the user; Eve is the malicious user (not own Alice's Key); Bob is Alice's family doctor (own the partial Alice's Key). Key is never sent to server. Alice who own the correct key can calculate the SUM, AVG and SD and reconstruct the raw data; Eve without a correct key can only reconstruct erroneous data. When Bob send request about calculating AVG, SUM or SD to server, Kryptein SQL specified defined functions will only send cipher back to Bob. Bob with the partial correct key can only compute SUM, AVG and SD over cipher text.

3 Background and Threat Models

3.1 Encrypted Query Processing

In Cloud-based systems, the interaction between the end devices and the Cloud can be abstracted into different queries. For example, to store the data in the Cloud, insertion queries can be used to upload data to the Cloud database. Similarly, frequent interactions involve selecting and searching the records in the Cloud and computing aggregations such as summations and averages. To protect the data in the Cloud during query processing, researchers have recently proposed the encrypted query process. For example, CryptDB first adopts an SQL-aware encryption scheme which uses a set of well-defined primitive operators, such as equality checks, order comparisons, aggregates (sums), and joins. With specified user defined functions (UDF), corresponding queries can be operated on specified encrypted database.

As shown in Fig. 2, Kryptein adopts similar architecture as CryptDB but it uses a new encryption/decryption engine. To be compatible with existing systems, Kryptein is built on the unmodified database management system (DBMS). The new compressive sensing based engine offers several advantages compared with the CryptDB and Talos. First, by seamlessly integrating compression and encryption, the communication costs are significantly reduced during query processing. Second, through statistical computation over cipher, aggregations such as finding average, summation, and standard deviation can be efficiently computed over cipher texts without raw data decryption. Third, through optimized dictionary learning mechanisms, the raw values

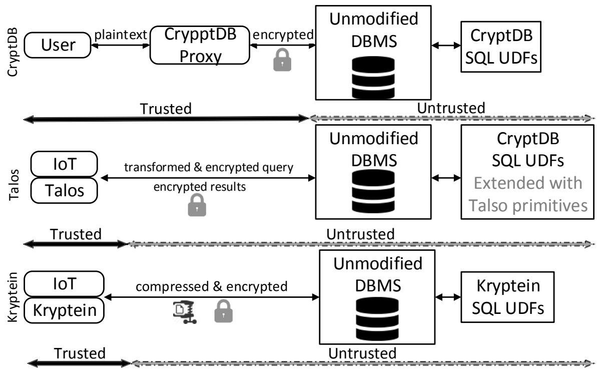


Fig. 2: Architecture comparison between CryptDB, Talos, and Kryptein.

can be accurately decrypted, while ensuring only the end devices are able to compute the raw values with very small computation overhead. We will detail the Kryptein design in the later sections.

3.2 Compressive Sensing

Consider a data vector $x \in \mathbb{R}^n$. It is called k-sparse if there exists a basis, $\Psi \in \mathbb{R}^{n\times n}$, which can represent x with only $k(< n)$-coefficients. The vector x is k-compressible, if there is a basis or data dictionary that can represent x, with k-significant coefficients and $(n-k)$ nearly zero or negligible coefficients. Natural signals are compressible. Therefore, we will use the notion of a compressible signal in the rest of paper. Compressive Sensing mainly utilizes compressibility to recover a data vector from a small number of measurements. In the Compressive Sensing (CS) literature,"measurements" and "projections" are used interchangeably, where a projection is a multiplication of x, with a vector ϕ_i, and the vector pair satisfies certain conditions. If we take m such measurements and stack them as a measurement matrix $\Phi \in \mathbb{R}^{m\times n}$, the projections can be represented by:

$$y = \Phi x = \Phi\Psi\theta \tag{1}$$

where $\theta \in \mathbb{R}^n$ is the representation of x in Ψ. As $m < n$, (1) is an underdetermined problem, i.e., there are more variables than the number of equations, therefore, (1) will have more than one solution. Encouragingly, adding the sparsity constraint that the data vector is sparse or compressible allows one to find a unique solution to this underdetermined system of linear equations.

The classical solution to this underdetermined system of linear equations is to minimize the ℓ_2 norm, i.e. minimize the amount of energy in the system. This is usually simple mathematics, however, this leads to poor results for most practical applications. ℓ_1-norm been shown as the most feasible in its tractability and accuracy. Most importantly, ℓ_1-norm involves solving an easier convex optimization problem:

$$\hat{\theta} = \arg\min_{\theta} ||\theta||_{\ell_1}, \text{ s.t. } y = \Psi\Phi\theta, \tag{2}$$

which can be solved in polynomial time.

In the context of compression secrecy, the data vector x is the plain text and y is the encrypted or cipher texts.

Representing (1) as (3), where Φ is measurement matrix and $\mathcal{A}$ is holographic dictionary [5], [6], the cipher texts y can be decrypted to the raw plain text $\hat{x}$ (estimated x) only with the correct $\mathcal{A}$ and Ψ.

$$y = \Phi\Psi\theta = \mathcal{A}\theta \tag{3}$$

CS in encryption will sacrifice accuracy (only get the estimated $\hat{x}$) while achieving reduction in wireless transmission volume. The compression is lossy and the space savings can be represented as $\{1 - \frac{m}{n}\}$.

Previously, [7] have shown that to achieve a high accuracy while maintaining high compression, the data dictionary and the projection matrix need to be tailored to the data vector. Intuitively, a tailored data dictionary provides a highly compressible representation of the data vector and a projection matrix can be tailored to achieve higher incoherence with the data dictionary, which is a requirement for successful reconstruction. In addition, a tailored dictionary and a tailored project matrix can also produce a unique holographic dictionary $\mathcal{A}$ while maintaining higher reconstruction accuracy compared to using an off-the-shelf dictionary (e.g., Fourier, Discrete Cosine Transform) with a random projection matrix (e.g., i.i.d Gaussian, Bernoulli etc.). Note that despite the uniqueness of $\mathcal{A}$ it could be compromised if an attacker gets access to the transmission channel, but without Ψ, it is not possible to reconstruct x. The results later in this paper (see section 9) show that without the "same data" to learn Ψ, it is difficult to recover it. This makes the proposed system highly reliable.

3.3 Learning Sparsifying Dictionary

In dictionary learning, given a set of P data vectors $\{x_1, x_2, ..., x_P\} \in \mathbb{R}^n$ a dictionary $D \in \mathbb{R}^{n\times d}$ is computed such that the vector x_is are simultaneously sparse in D. In other words, given the representations s_i of x_i via $x_i = Ds_i$, all the vectors $s_{i,i=1,...,P}$ need to be sparse in D. Previously, we used a notation Ψ to represent a basis. However, we introduced a different notation D since dictionaries are usually overcomplete (number of columns are greater than the number of rows), whereas basis is usually a square matrix. Due to the change in the dimension, we also replaced the previously used notation θ_i for coefficients with a new notation s_i.

Dictionary learning can be formulated as an optimization problem. The requirement that s_i is sparse can be imposed by forcing the vectors s_i to have a small ℓ_1 norm. Let d_j denote the j-th column of D. The optimization problem can be defined as follows:

$$\min_{s_i, D} \sum_{i=1}^{P} (\frac{1}{2}\|x_i - Ds_i\|_2^2 + \lambda\|s_i\|_1) \quad s.t. \quad \|d_i\|^2 \leq 1 \forall_{j=1,...,n}, \tag{4}$$

where λ is a regularization parameter. It regulates s_i so that it does not grow too sparse or too dense. The optimization problem in (4) is convex with respect to each of the variables D and s_i, when the other one is fixed. Therefore, generally, the solution could be obtained in two steps: getting the most sparse coefficient s_i thus keeping the dictionary D fixed, and then learning the most sparse representation dictionary D while keeping the coefficients s_i fixed [8]. In this paper, we use SPAMS [8], which was previously used by [7] for trajectory reconstruction. For dictionary learning, SPAMS uses the LARS-Lasso algorithm [9], which is a homotopy method [10] providing the solutions for all possible values of λ. Using SPAMS because [7] achieved a higher reconstruction accuracy for GPS trajectory compare to other competing methods.

3.3.1 *Construction of Projection Matrix*

We adopted the method proposed in [7] for an construction of the projection matrix. This method performs better than the best-reported methods [11], [12] in the literature. The method first computes the Singular Value Decomposition (SVD) of the data dictionary D:

$$D = U\Lambda V^T, \tag{5}$$

where ^{T} denotes matrix transpose, $\Lambda \in \mathbb{R}^{n\times d}$ contains the singular values in its main diagonal, and $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{d\times d}$ are orthonormal matrices. Then the m columns in U corresponding to the largest m singular values of D are chosen to form the projection matrix. Let assume that the SVD in eq. (5) has been permuted so that singular values appear in non-increasing order in the diagonal of Λ. With this notation, let U_m denote the sub-matrix containing the left-most m column of U, which also corresponds to the largest m singular values of D. Our choice of the projection matrix is therefore, U_m^T. In [7] it has been shown that this choice of projection matrix maximizes the signal power of y. A higher signal power typically translates to lower estimation error.

3.4 Threat Models

Kryptein compresses and encrypts all data entries in the trusted *client side* and only sends the compressed and encrypted data to the *server side*. The client devices (e.g., smartphones) encrypt the user data using the learned dictionary and password key. For low cost computation such as average (AVG) and standard deviation (SD) , related cipher texts will be sent back to the users' smart phone for fast statistical computation over cipher. For high-cost raw data decryption, the computation-intensive tasks will be processed by the Cloud. Then the computed sparse coefficients will be sent back to the client for efficient raw data reconstruction. Note that with only the sparse coefficients, the Cloud cannot decrypt the raw data, hence keeping the data protected during the query processing.

3.4.1 *Threat 1: Curious Database Administrator*

In this threat model, we assume the Database Administrator (DBA) is honest-but-curious, and the DBA doesn't have the knowledge of dictionaries learned on the client side. The honest-but-curious adversary is a legitimate participant (e.g., DBA) in a communication/computing procedure who will not deviate from the defined procedure but will attempt to learn all possible information from legitimately received/result messages [13]. The attacker (e.g., DBA) is assumed to be only 'curious' and attack is passive. The attacker is only curious about the confidential data, but will not modify any data stored in the Cloud. Our goal is to achieve confidentiality (data secrecy) of the data. This

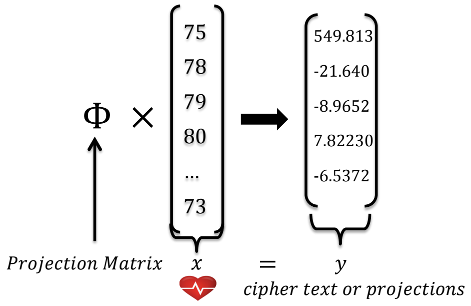


Fig. 3: Compression process: The x on the left is an example of one real column heartbeat data and the y on the right is the result after it has been compressed with user's specified projection matrix Φ.

threat includes DBMS software compromises, root access to DBMS machines, and even access to the RAM of physical machines [3].

Approach. Kryptein adopts the random compressed encryption to protects the data from the curious DBA or other attackers who have the full access to the data stored in the DMBS server. Since the data stored in the Cloud database is compressed and the DBA doesn't have the knowledge of the dictionary, Kryptein prevents the private information from being reconstructed by curious DBAs. As shown in Fig. 3, even the cipher text y is stored on the Cloud, the curious DBA cannot decrypt the real heartbeat data (even the range of the data) without the dictionary (Ψ).

3.4.2 Threat 2: Arbitrary Threats

In this threat, the entire server is compromised and all the compressed data stored in the server is exposed to a malicious hacker, one who is an active attacker and is able to launch arbitrary attacks. For example, she can use the dictionaries learned by herself to reconstruct the raw data (x) from cipher texts (y).

Approach. We tackle this threat by taking the compressive sensing gross encryption approach on numerical data. Our approach in Kryptein provides secure protection against the malicious hackers by ensuring that without the user "keys" (locally learned dictionaries) the raw private data cannot be accurately reconstructed. The server will never request decrypted data from the client. Even the fully compromised Cloud server will not send malicious requests to clients instructing them to perform raw data decryption and send back to server. The server will only perform as a safe vault and never get access to raw data. More detailed discussions on the security analysis will be presented in the security evaluation section.

3.4.3 Threat 3: Eavesdropper Threats

In this threat, the eavesdroppers receive all the communication data by sniffing the communication channel. The eavesdroppers are able to perform active attacks to reconstruct the user data.

Approach. In Kryptein, even if the eavesdroppers are able to obtain the whole communication information, the threat can be viewed the same as Threat 2 discussed above. In this situation, the data leakage becomes same as Threat 2 when the entire server gets compromised. Therefore, Kryptein provides security guarantee to the eavesdropper threats as well.

TABLE 1: A Summary of Key Notations

Symbol	Meanning
x_i	the i-th raw data segment (with dimension n)
$\hat{x}_i$	i-th raw data segment get recovered correctly
$\hat{x}'_i$	i-th raw data segment get recovered by the hacker
$\overline{x}_i, \mu$	average of i-th raw data segment
σ	standard deviation of i-th raw data segment
X	raw data segments used to learned the dictionary
y	projections or cipher text
y'	perturbed projections or cipher text
n	the raw data segment dimension
m	the compressed (or perturbed) data dimension
i	data segment index
Ψ	learned dictionary/basis
Φ	compressive sensing projection matrix generated from Ψ
A	compressive sensing encryption key ($A = \Phi\Psi$)
A'_i	perturbed compressive sensing encryption key ($A'_i = \rho_i\Phi\Psi$)
Γ	the function generate the perturb transformation matrix
θ_i	the genuine (estimated) sparse representation of x_i
$\hat{\theta}_i$	the recovered sparse representation of x_i
ρ_i	random perturbation matrix
Alice	Normal user
Eve	Malicious user/eavesdropper

4 Random Compressed Encryption

This section illustrates how Kryptein compresses and encrypts the user data. Unlike the deterministic encryption method (decrypted value exactly equals to the plaintext), CS only reconstructs (decrypts) the data to estimated values, and is regarded as a gross secure method. For some sensing apps, close estimation of the raw data instead of getting the exact value satisfies the application requirement. For example, in GPS tracker, with a close estimation of the GPS values, the app can approximately get the user's walking trajectory. There are less requirements to get the exact GPS values for those kind of applications.

Furthermore, sensors themselves inherently contain the measurement errors. For example, civilian GPS has a 7.8 metre 95% interval horizontal accuracy[3]. The optical heart rate sensors have 85% to 90% accuracy in real world test [4]. We use both GPS sensors and optical heart rate sensors in our evaluation (Section 9).

On the other hand, compared with the measurement errors from sensors, CS reconstruction error can be very small. Take heartbeat data as an example, the percentage error of data reconstruction can be as small as 1%, which will be illustrated in the later part of this paper (Section 9.1.2). In addition, the CS encrypted data has a significantly smaller size and incurs less communication costs. Therefore, in Kryptein, we use CS as the underlying encryption and compression method to protect the data in IoT systems.

Kryptein first learns a dictionary ($\Psi \in \mathbb{R}^{n\times n}$) using the method discussed in section 3.3, then it constructs a projection matrix $\Phi \in \mathbb{R}^{m\times n}$ by SVD method eq. (5) refers from section 3.3.1, m is the dimension after compress. After that, the user is also required to select a key to generate corresponding perturbation matrix. In our prototype, we select a signed 32bit integer as a secret number (depend on

3. http://www.gps.gov/systems/gps/performance/accuracy/
4. http://www.wareable.com/fitness-trackers/heart-rate-monitor-accurate-comparison-wrist

Algorithm 1 Compress and Encrypt Data

```
1: Inputs: Original data segment $x_i$ with timestamp $i$, seed, and projection matrix $\Phi$.
2: Generate each perturbation matrix $\rho_i$ with the seed.
3: Calculate compressed and encrypt data segment $y'_i$ by multiply with projection matrix and perturbation matrix $y'_i = \rho_i \Phi x_i$
4: Send the $y'_i$ to server and delete the original data segment $x_i$.
```

the platform) and the integer is used as the $seed$ to generate a series of random normal distributed matrices $\rho \in \mathbb{R}^{m\times m}$ as perturb matrices. For different application purposes, there could be different ways to generate secure random perturb matrices. As this is not the focus of our paper, we adopted the perturbation matrix generation method in [14] to provide an efficient and secure protection to the data.

Kryptein utilize a linear perturbation method to randomly permute the cipher text y. It is clear that combining with random perturbation, the cipher texts will be different even though the original plain texts are the same, hence achieving higher secrecy. The linear computation will cause little more computation overhead. Therefore:

$$\rho_i = \Gamma((seed + i)), \tag{6}$$

where $seed$ is the user's secret number, i is the current data index and Γ is the random normal distribution matrix generation.

Once the secret dictionary and seed are generated and stored in the client side, the client then starts to generate perturb matrices and use the dictionary combined with perturbation matrix to compress and encrypt each new sensor data segment. In the rest of this paper, we will use client or smartphone alternatively as in most current IoT systems, smartphones are used as the gateways for nearby IoT devices and act as the client. Each user creates his/her own key (learned dictionary and seed), which is secretly stored in the client.

After the perturbation matrix and learned dictionary are built, new sensor data will be compressed and encrypted in the client, which results in less energy and computation consumption than traditional encryption methods as we will show in section 9.2. After the data is compressed and encrypted, the original data will be deleted and the CS-encrypted data will be uploaded to the server. However, due to the characteristic of matrix computation, the data can only be processed once being accumulated to one segment, for example 60 heartbeat data. If the data size is less than one single segment, Kryptein cannot function. This assumption can be easily satisfied as IoT systems generate abundant data. For instance, current wearable devices only take 3 to 10 minutes to generate 60 heartbeat data [15], [16].

The CS-encrypted process is shown in algorithm 1. The projection matrix Φ is generated from the uniquely learned dictionary Ψ (also D in eq. (4)). Then each new coming data segment will be projected with the Φ, meanwhile a random perturbation matrix is generated based on the user seed. Then, each compressed data will be randomly perturbed multiplying with the random perturbation matrix ρ_i. After that, the client sends the compressed and encrypted data to the server and deletes the local data segment. The final compressed and encrypted data becomes:

Algorithm 2 Average Calculation on Compressed and Encrypt Data

```
1: Inputs: CS-encrypted and perturbed data segment $y'_i$ with $m$ data records from server; projection matrix $\Phi$ from client.
2: Use the user's seed and the time stamp of $y'_i$ to determine the corresponding perturbation matrix $\rho_i$ of cipher text $y'_i$.
3: Calculate the AverageMultiply matrix $\mathcal{M}_i$. (see algorithm 3)
4: Multiply the $\mathcal{M}_i$ to the $y'_i$ to get the average of the original data segment $\overline{x}_i = \vec{\nu} \cdot \vec{x_i} \approx \mathcal{M}_i y'_i$.
5: Output: Estimate average of raw data segment $\overline{x_i}$
```

Algorithm 3 Calculate the *AverageMultiply* matrix $\mathcal{M}_i$

```
1: Inputs: Corresponding perturbation matrix $\rho_i \in \mathbb{R}^{m\times m}$; projection matrix $\Phi \in \mathbb{R}^{m\times n}$
2: Get the dimension $m \times n$ of $\Phi$.
3: Set a support vector $\nu = [v_1, v_2, ..., v_n], s.t. \quad v_1 = v_2 = ... = v_n = \frac{1}{n}$.
4: Calculate the $\mathcal{M}_i$ with dimension $1\times m$ by solving deterministic linear equation $\mathcal{M}_i = \nu\Lambda^T(\Lambda\Lambda^T)^{-1}, s.t. \quad \Lambda = \rho_i\Phi$. (Take pseudo-inverse of $\Phi$ as it is not square.)
5: Output: $\mathcal{M}_i$
```

$$y'_i = \rho_i \Phi x_i \tag{7}$$

The CS-encrypted data on the server is able to be recovered only with user's correct "key" (learned dictionary and the appropriate perturbation matrix).

5 STATISTICAL COMPUTATION OVER CIPHER

When a user launches a specific query to get the statistical information (e.g., AVG, SUM and SD), Kryptein requires no raw data decryption on the server. Instead, it sends back the CS-encrypted data segment y'_i $(i = 1, 2, 3...)$ to the client for statistical computation over cipher. As a result, in the worst case, when the server gets fully accessed by the hacker (Threat 2), there are no more raw data leakage due to decryption.

Eq.7 represents the data stored in the server after compression and encryption. The random perturbation matrix (ρ_i) is used to permute to cipher text. Because the client has the knowledge of the perturbation matrix sets, he/she can then correspond to a data segment with relative perturbation matrix.

The algorithm 2 illustrates how Kryptein calculates the average over the cipher texts. As the ρ_i is generated based on the user's specific $seed$ and the time stamp for each data segment, the Kryptein client could always reproduce the perturbation matrix with the seed and the time stamp i since the starting $seed$ and transformation Γ is known to the client. Since eq. (7) contains the raw x_i, we can find a medium transformation matrix ($\mathcal{M}_i$) to consist a support vector (algorithm 3 line 3) to help approximate the average through $\overline{x}_i = \vec{\nu} \cdot \vec{x_i}$. And due to the linear matrix transformation property in $\mathbb{R}$, we have:

$$\mathcal{M}_i y_i' = \mathcal{M}_i \rho_i \Phi x_i \approx \vec{\nu} \cdot \vec{x_i} = \frac{1}{n} \sum_{\tau=1}^{n} x_\tau \tag{8}$$

where $\frac{1}{n} \sum_{\tau=1}^{n} x_\tau$ is the mean value of the data segment x_i. Since the product of $\rho_i \Phi x_i$ is an underdetermined matrix, we can only take pseudo-inverse to get the estimated M_i, thus leading to an estimated average calculation. Theoretically, one can calculate the average over the cipher even cipher size is less than one single segment by setting the support vector ν in algorithm 3. For instance $\nu = [\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, 0, 0, 0, 0, 0]$ can be used in calculating the average of the first five data points in a data segment of dimension 10. However, the more 'absence'(zero element) in ν , the less accuracy of the calculated average. Otherwise, one can directly reconstruct the data by setting an appropriate ν (only that data point position is 1 and other position 0). Because too much information is ignored, the accuracy will decrease. The accuracy decrease can be applied to different applications depending on the requirements. The error brings by the pseudo-inverse (or the underdetermined factor) is small and can be ignored, this is further discussed in the evaluation section (refers to Section. 9.1.2)

In algorithm 3, Kryptein has to re-calculate the dimension of Φ (line 2) because the client has no knowledge of x. It has to get the related information from the dictionary Ψ. Regarding the fact that the Φ is generated from Ψ (illustrated in section 3.3 and section 3.3.1), Kryptein could then gain knowledge of Φ via eq. (5), where Ψ and Φ relate to D and U respectively.

With the algorithm 2 and algorithm 3, Kryptein could calculate the average of the data segment of x in client. The client can also calculate the summation as the dimension of x becomes known (same to row dimension of Ψ). Another important piece of statistical information which is supported by Kryptein, is the standard deviation (σ): $\sigma = \sqrt{E(x-\mu)^2} = \sqrt{E[x^2] - (E[x])^2}$, where $E[x] = \mu$ is the mean value (AVG) of x.

We can get $E[x^2]$ through $\|x\|_2^2/n$, and in compressive sensing theory, only random Φ can be proved satisfy the Johnson-Lindenstrauss (J-L) lemma [17]. If J-L lemma get satisfied, it has $\|x\|_2^2 \approx \|y\|_2^2$. We tested our Φ experimentally to see if it satisfies J-L lemma. Based on our experiment result in 9.1.2, our method is able to show a very small error between $\|x\|_2^2$ and $\|y\|_2^2$ (e.g. Root Mean Square Error of 0.02) based on a real heartbeat dataset consists of 20 subjects. Hence, we can estimate $\|x\|_2^2$ by using $\|y\|_2^2$, and y is the compressed data segment.

With $E[x]$ and $\|y\|_2^2$, we can get the AVG and SD of a single segment x_i, and it is straightforward to calculate the total average of a period of time t through $\mu_{total} = \frac{1}{t} \sum_{i=1}^{t} (\mu_i)$, and estimate the total standard deviation by:

$$\sigma_{total} = \sqrt{E[x_{total}^2] - (E[x_{total}])^2} = \sqrt{\frac{1}{t} \sum_{i=1}^{t} (E[{x_t}^2]) - \mu_{total}} \tag{9}$$

Therefore, Kryptein computes the AVG, SUM and SD over cipher without additional information leakage.

6 Raw Data Decryption

In many applications, raw data need to be decrypted by the client. For state-of-the-art cryptography system, the decryption typically has high computation consumption[3], [18], [19]. To reduce energy consumption when decrypting the raw data in Kryptein, we offloaded the heavy computation to the server, only leaving the light-weighted computation and sensitive raw data decryption on the client. There are existing works in Cloud/Edge-computing context offloading the complex computation to the (edge) server [20], [21], [22], [23] with special design about privacy protection. Kryptein is also designed with the privacy consideration on the Cloud.

Kryptein can directly decrypt the raw data without going through the statistical computation over cipher. In Kryptein, eq. (7) can be converted to $y_i' = \rho_i A \theta_i$,where y_i' represents the cipher text in server and it keeps CS-encrypted and perturbed, A refers to the compressive sensing encryption key ($\Phi\Psi$) and θ_i is the sparse coefficient for i th data segment, $i = 1, 2, 3...$ represents different data indexes. Therefore the raw data decryption is mainly about calculating the θ and $\hat{x}$ (the estimated value of raw data x). Our purposed method guarantees that the malicious hacker who gets control of the server could not reconstruct the original data $\hat{x}$. We show these results through experiments in section 9.

To reconstruct the raw data segment x, the client sends the perturbed CS key (A_i') to the server to solve the eq. (10):

$$\min_{\hat{\theta}_i \in \mathbb{R}^n} \|\hat{\theta}_i\|_1 \quad \text{subject to} \quad \|y_i' - A_i' \hat{\theta}_i\|_2 \leq \epsilon \tag{10}$$

where $A_i' = \rho_i \Phi \Psi$ and has the same $\hat{\theta}_i$ to eq. (11). This is because the deterministic matrix (ρ_i) will not intervene with the result of the ℓ_1 optimization.

$$\min_{\hat{\theta}_i \in \mathbb{R}^n} \|\hat{\theta}_i\|_1 \quad \text{subject to} \quad \|y_i - \Phi\Psi \hat{\theta}_i\|_2 \leq \epsilon \tag{11}$$

As shown in the equation above, we want to get the $\hat{\theta}$ in eq. (10) as the same one without perturbation (in eq. (11)).

After Kryptein calculates the $\hat{\theta}$ in the server, the $\hat{\theta}$ is sent back to the client. Then, with the genuine user dictionary Ψ, the user could estimate the raw value using $\hat{x_i} = \Psi\hat{\theta}$. For different CS-encrypted data segment y_i', the corresponding A_i' has to be sent to the server to solve the high computation ℓ_1 optimization. Under the worst situation, the hacker could hold all the A_i'. However, the hacker still cannot reconstruct to the raw data segment x_i or the genuine estimation $\hat{x_i}$. By making the complex computation on the Cloud, the client device decrypts the raw data only by taking the simple matrix calculation ($\hat{x_i} = \Psi\hat{\theta}$) on the intermediate results sent by the Cloud.

7 Decryption on Demand

To satisfy the need of IoT systems, we design the system with a layering structure to decouple different functionalities. Different layers correspond to different functionalities and different computation burdens. As shown in Fig. 4, we design an onion-like model to implement the function layers. The consumption of Kryptein in energy, time, and computation increase when getting deeper inside to the core of the onion. Comparing with other onion-like systems (e.g., CryptDB), our onion layers are less coupling on each other layers. For instance, the system could reach the third layer by decrypting the cipher directly instead of peeling the outside layers.

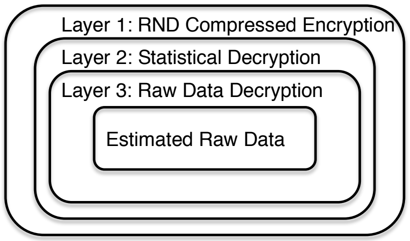

Layer 1: RND Compressed Encryption
Layer 2: Statistical Decryption
Layer 3: Raw Data Decryption
Estimated Raw Data

Fig. 4: Onion-like system layers.

In Layer 1, the data stored on the server has been randomly compressed and perturbed. Without knowing the perturbation matrix ρ_i, the server cannot get to the genuine compressed data segment y when both y_i and ρ_i are unknown in $y_i' = \rho_i y_i$. As illustrated in eq. (7), Layer 1 only randomly permutates the processed data and stores it on the sever. In this way the secrecy guarantee (see section 8) as well as significant energy saving is achieved (see section 9.2).

When a query is launched to the Cloud to collect the statistical information, Kryptein responds with the Layer 2 function (Section 5). The Cloud only needs to respond to the query by sending back the relative CS-encrypted data segments. The client side finally compute those aggregation results (AVG, SUM and SD).

When applications require the user's raw data, Kryptein will decrypt cipher texts to the raw data while keeping them secret to the Cloud.To balance the security and efficiency, we decouple the decryption task between Cloud and the client. The assigned cipher text will be decrypted into the form of intermediate text (instead of plain-text) on the Cloud. Afterwards, the intermediate text will be sent to the client to complete the entire decryption. This design offloads the heavy decryption computation to the Cloud with the guarantee that the information leakage is minimized. In section 9.2, we evaluate and show that Layer 2 could be more efficient (in terms of total time and energy consumption) than Layer 3 for AVG calculation.

8 Privacy Analysis

In this section, we discuss the secrecy properties of our proposed compressed sensing encryption method. To analyze the secrecy, consider a K-sparse message $x \in \mathbb{R}^N$ is chosen from a sample of a probability distributions. A key set $\gamma \in \{1, ..., S\}$ corresponds to an $M \times N$ compress and perturbs matrix $NounceM_\gamma$ ($NounceM_\gamma = \rho_\gamma * Phi$). Put into Kryptein, the user wants to send a secret message to the server. The user chooses her own key Δ (with uniform probability among the keys) and encrypts the message x using the $NounceM_\gamma$ matrix via matrix multiplication $y = NounceM_\gamma x$. Only the cipher text y is transmitted to the server, even the server does not know what key is being used to compress and encrypt the message. If an adversary, Eve, intercepts the user's encrypted message y, but she does not know which key was used to encrypt the message. It will be very difficult for Eve to recover x. According to the Shannon's theory [24], perfect secrecy can be achieved if the probability of a message conditioned on cryptogram is equal to a priori probability of the message, P $(X = x \mid Y = y) = P(X = x)$. Earlier work [6], [25] proves that perfect security can only be achieved in compressive sensing if three conditions get satisfied: 1) the number of measure M is equal or greater than two times the sparsity of the messages, i.e., $M \geq 2k$; 2) the sensing matrix (called 'measurement matrix') satisfies RIP (see section 3.2); and 3) the number of source messages goes to infinity. Theoretically, the perfect secrecy gets proved based on the Shannon's definition only if the number of sources goes to infinity, which is not achievable in real application. However, the argument holds if $I(X; Y)$ (I refers to the mutual information) becomes small enough.

To understand how difficult it is for Eve to recover the message x using only y, knowledge that how A is generated, and the set of seed, one possible approach for Eve is to try all 'keys' (refers to random generated matrices set corresponding to the user's secret seed) and attempt to recover the signal x. She would only stop when she thinks she has succeeded, this similar topic has been detailed discussed in [25].

There are also other applications and papers that apply compressive sensing secrecy [26], [27], [28]. Those CS based secrecy systems used a random measurement matrix as the secret, which make more sense to preserve the secrecy of the data due to the decoupling from the plaintext. Kryptein adopts a learned projection matrix to make the reconstruction accuracy loss as small as possible meanwhile added one more layer (random Layer) to decouple with plaintext. By this, the secrecy (randomness) get enhanced to the same of those random compressive sensing secrecy systems but with the least reconstruction error.

Attack methods like Principle Component Analysis (PCA) based attacks [29] and Spectral Filtering (SF) [14] based attacks are mainly to exploit the correlation that exists among the multi-dimensional data (with time series) to filter out the injected noise [30], [31]. However, in our model, we 'compress' the data instead of 'injecting' the noise and we compress the data dimension. Those attacks method cannot be used in Kryptein. The better attack method is to guess/predict the essential projection basis with current information. For the reason above, we assume Eve knows and follows our method in trying to get the most closed dictionary. Since Alice's personal dictionary is kept securely in the client, Eve could reconstruct the data through getting a 'closed' dictionary by (1) learning a dictionary from another person's data; (2) learning a dictionary from a group of people's data. This is based on the inference that: for a long and large dataset, people's behavior probably could have the similar representation. In the attack model, Eve could learn her own Ψ'. We will show in the evaluation sections that such attacks incur large reconstruction errors, and cannot get a close estimation to the real data.

9 Evaluation

9.1 Experiment with Cases

We implemented Kryptein on Microsoft Bands [16] and Fitbits[15], for data collection, and we used Motor E (2nd Generation) running Android 4.4 as the trusted client. The Cloud server ran a MySQL database on Ubuntu 12.04.

We evaluated Kryptein by using two cases: Secure GPS Track and Private Health Monitor. Secure GPS track, namely is to maintain a secure GPS track for the user who is using normal GPS services. Through Kryptein, GPS data can be shifted and stored in the Cloud safely and will only illustrate to the user with the key. Our GPS data is from the Microsoft project GeoLife [32]. We randomly selected 11 individual's data sets (10 subjects as a group and another 1 subject as an individual) as the malicious user's training data. For the private health monitor case, we collected real data from 20 subjects using Fitbit and Microsoft band [5]. Subjects are using a normal health monitor service e.g. send personal heartbeat data to Fitbit Cloud and get information summary there. We collected for about 30mins to 1 hour of heartbeat data for each user (about 4K data records) without any modification on collection devices. For both cases, the hacker launches both **'individual'** and **'group'** attacks.

Two metrics are used to evaluate the system performance. For the statistical computation over cipher accuracy, we take RMSE (Root Mean Squared Error) as the metric. For GPS data, the AVG and SD cannot be interpreted meaningfully. Hence we only measure the reconstruction accuracy for GPS data. To measure the decryption accuracy for heartbeat data, we take the percentage error (PE) as the accuracy measurement. PE value could better represent the error value to the original value.

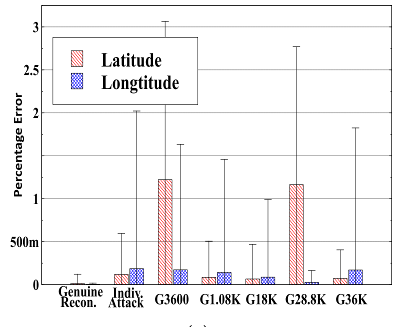
9.1.1 Secure GPS Track

It is clear that people have had an increasing understanding about how personal private information can be revealed by (raw) GPS data and what potential threats can be caused due to those information leakages [33]. Many IoT applications have embedded with GPS sensor in order to provide a location related services. For the purpose of providing better services and better resource limits of IoT devices, GPS data is always uploaded to the Cloud, which further deteriorates the leakage [34]. Besides, uploading a lot of GPS points may lead to problems like heavy network traffic and much higher power consumption [35].

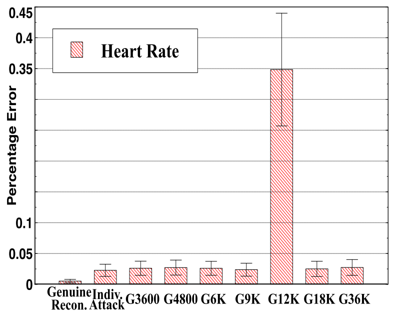
For the GPS trajectories, we adopt the method to convert the GPS to Universal Transverse Mercator (UTM) coordinate system to compress and encrypt. After the decryption process of Kryptein, the UTM was turned back to GPS and then was simulated on the map. For the benefit of spaces, we will evaluate the secrecy of GPS dataset in this paper only because the reconstruction accuracy and energy consumption has been well studies in previous work [7].The reconstruction error directly measures how close the malicious user can estimate the real user private data.

Fig. 5(a) shows the data reconstructed by the user in Kryptein using the genuine dictionary as keys. We can see that the reconstructed GPS trajectory fits the original

5. Ethical approval for carrying out this experiment has been granted by the corresponding organization (Approval Number HC16180).



(a)



(b)

Fig. 5: Individual and group attack result compare to genuine reconstruct: (a) GPS dataset (b) Heartbeat dataset. G represents for group attack, number represent for the amount of data used in that attack.

trajectory very well, which indicates that Kryptein provides an accurate raw data decryption for users. The percentage error of Latitude and Longitude are calculated separately in UTM unit.

To understand how close the malicious user can estimate the private data using his/her own trained dictionaries, we vary the amount of malicious user's training data and measure the reconstruction accuracy. We randomly selected 10 individual's GPS data (as a group) from the Geolife dataset as the training dataset of the malicious user, and increase the amount by $0.1n$ each time. For example, the notation G18K refers to collect 18K data ($10 \times 0.5 \times 3600$) records as a group. As shown in Fig. 5(a), group attacks achieved less reconstruction error than individual attacks, but they are significantly larger than the error achieved by genuine dictionary. For example, the smallest reconstruction error achieved by attacker is 15% (the average error of Latitude and Longitude is about 0.15 showed as G18K in Fig. 5(a)). This is 30 times larger than using genuine dictionary (showed as Genuine Recon. in Fig. 5(a)).

Fig. 6(a) and Fig. 6(b) show the visualization of the recovered GPS trajectories using the genuine dictionary and the least reconstruction error 'fake'dictionary (which is G18K group in Fig. 5(a)) used by the malicious user.

Although the attacker'reconstruction error using the 'fake'dictionary is not that much (the smallest one is 15%), the GPS trajectory turns quite weird (showed in Fig. 6(b)). The reason for that is GPS data is processed (encrypted/decrypted) in UTM unit. When drawing the GPS trajectory in real map, the data is required to be converted to Longitude and Latitude. Thus, the attack reconstruct GPS trajectory is quite leaping and even in different magnitude

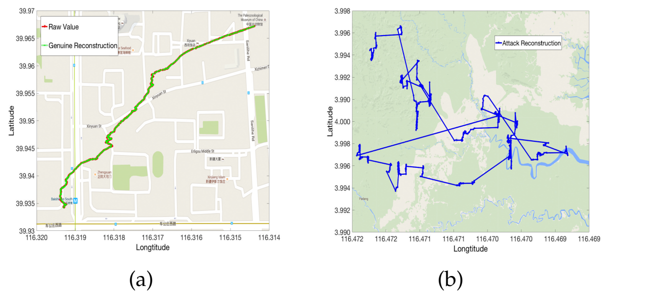


Fig. 6: Real and attack reconstruct GPS trajectory: (a) the raw GPS trajectory and real reconstruct by Kryptein (b) individual attack result. The x- and y- axis is the real latitude and longitude respectively.

like the Latitude as shown in Fig. 6(b).

The results indicate that Kryptein allows accurate raw data decryption on the user side, the malicious user who does not have the knowledge of the correct dictionary but uses the locally learned dictionaries is not able to get a close reconstruction result. In this way, Kryptein protects the data confidentiality in IoT systems.

9.1.2 Private Health Monitoring

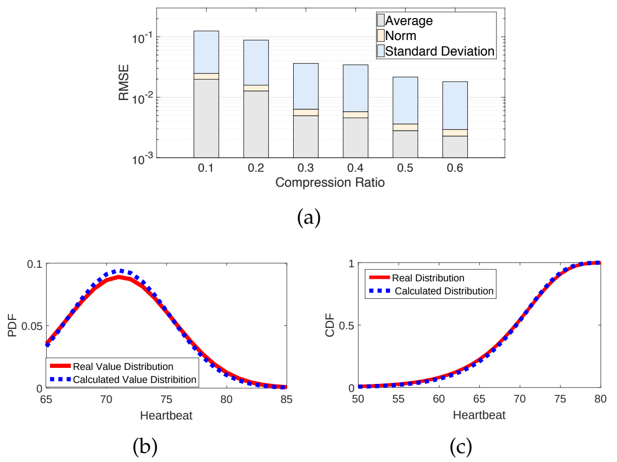


Fig. 7: Experimental results of parameters choice. Note log-scale Y-axis in (a).

With the help from the prosperous sensor-embedded wearable devices, people's attention has been re-drawn back to how IoT systems can help in improving daily health. Though the continually generated health data can be shifted to server to ease the burden on IoT device meanwhile provide a Cloud-based service, the privacy of that health data is also a big concern. We evaluate Kryptein with our prototype IoT system to determine how Kryptein's utility and security functions as a health monitor application.

Fig. 7(a) shows the RMSE of heartbeat data by varying compression ratios. To understand how different compression ratios affect the reconstruction results, we vary the ratio in the ranges of [0.1 0.6]. **AVG** refers to the average value of each data column x, **Norm** is the 2-norm squared of the vector x, with the AVG and Norm, we could calculate the **SD** (assuming the data satisfies the standard normal distribution) which represents the standard deviation of x. As the compression ratio gets larger, the RMSE decreases significantly. It is clearly showed that with compression ratio 0.3, those RMSEs have been smaller than 0.02. This means the estimated AVG and SD on cipher text could already be very closed to the raw data. According to ratio larger than 0.3, it is clear the RMSE is decreasing. However, a larger compression ratio means data gets less compressed which implies the burden increasing for the entire system. Which compression ratio to choose is up to the purpose of the genuine application. In the rest of this paper, compression ratio 0.3 ($m = 0.3n$) is selected as the result of balancing accuracy and resource consumption in our system.

Fig. 7(b) shows the PDF (Probability Density Function) of the heartbeat data. It is clear that with the calculated AVG and SD, the PDF curve has a high correlation with the real curve. Fig. 7(c) refers to the CDF (Cumulative Distribution Function) of the same data. The two curves are almost fully overlapped, showing that the calculated data statistics fit the real data.

Fig. 5(b) shows how close the malicious user can estimate the private heartbeat data using his own trained dictionaries. We vary the amount of malicious user's training data and then measure the reconstruction accuracy. The compression ratio is 0.3. We use 20 testers' heartbeat data (as a group) as the training dataset of the malicious user, and increase the amount of data each time. The result in Fig. 5(b) shows that group attacks cannot achieve a significant better reconstruction accuracy than individual attacks, and both of them achieved significantly lower accuracy than that by genuine dictionary.

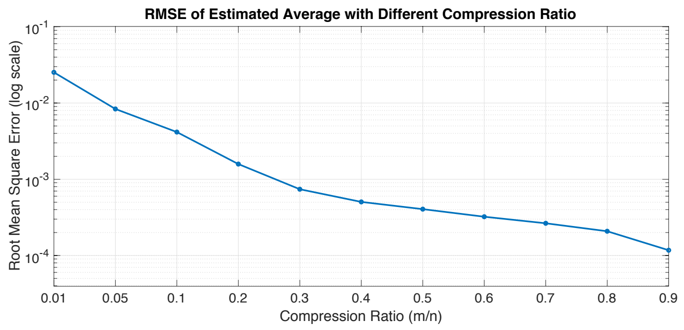


Fig. 8: Error of estimated average computed over cipher (Heartbeat dataset).

Estimated average over cipher is a very useful and crucial function for many applications like private health monitoring ones. Since the proposed CS-based encryption method can only calculate the estimated average (see Algorithm 2) rather than the lossless one, we evaluate how accurate is the estimated average with different compression ratios. Fig. 8 illustrates the RMSE between estimate average and genuine average calculated with raw heart rate data. It can be seen that even with an extremely small compression ratio ($m/n = 0.01$, 60 heart rate data compressed to only one decimal), the estimated average is still close to the genuine average with about 0.02 RMSE. At a commonly adopted compression ratio (e.g., $m/n = 0.1$), the RMSE between estimated average and the genuine average is even less than 0.005, which is considered can be ignored by most of the user cases.

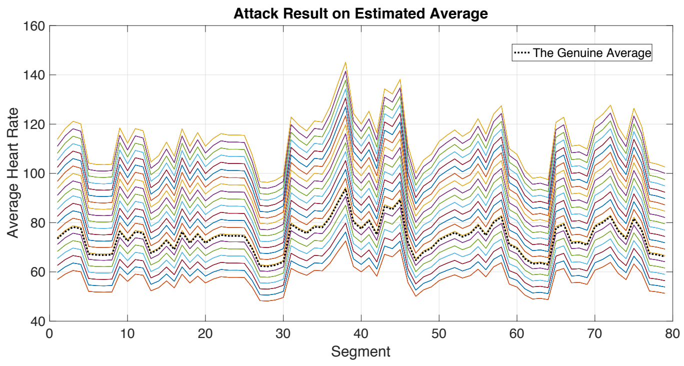


Fig. 9: Attack result on average heart rate reading. Each trace represents a possible guess by the attacker.

In the extreme case if the private health application only require a periodical average heart rate reading, one decimal cipher can handle this assignment. This is because from information theory perspective, one decimal cipher's information is sufficient to represent another decimal value (see estimated average in Fig. 8, CS-encrypted from 60 heart rate reading each segment), however, the raw heart rate data reconstruction error will rise significantly because of the limit information entropy. From the privacy perspective, in the worst case, the attacker can obtain the one decimal cipher (by compromising the Cloud or eavesdropping the channel), then the attacker can launch attacks to guess the other secret value held by Bob (family doctor in Fig. 1) in order to reveal the user's average heart rate readings.

According to Algorithm. 2, the average estimation can be derived from $\overline{x}_i = \mathcal{M}_i y_i'$, which $\mathcal{M}_i$ and y_i' degenerate to a decimal from a vector in this extreme case. Knowing the cipher (y_i') and normal average heart rate range is [60,100], attacker can only guess the range of the secret value ($\mathcal{M}_i$) to estimate the average heart rate value of each data segment ($\overline{x}_i$). The attack result is presented in Fig. 9, the attacker can guess any value for the secure number held by Bob as many times as he/she wishes. Even the attacker can obtain all segments' average cipher (e.g., about 80 data segments in Fig. 9 x-axis). Without additional information (about Bob's key or Alice's key), the attack cannot ascertain which attack result is (close to) the genuine one (i.e., privacy protection), since many of them are located within that reasonable range. In addition to that, Kryptein adopts a strategy changing the random perturbation matrix (ρ_i) periodically, it is even difficult for the attacker to obtain a fixed cipher ($\mathcal{M}_i$), which further enhance the provided privacy protection.

Fig. 10 visualizes the reconstructed heartbeat data. The attack dictionary used is the individual attack dictionary from Fig. 5(b), which has the smallest attack reconstruction error. The figure shows the results of four different random data segments. As shown in the figure, the reconstruction using genuine dictionary is very close to the original raw values. On the other hand, the reconstruction using the attack dictionary produced significant errors. The results show that it is difficult for the attackers to reconstruct original raw data with cipher texts without the knowledge of the correct dictionary. Therefore, the secrecy of raw heartbeat data is well protected.

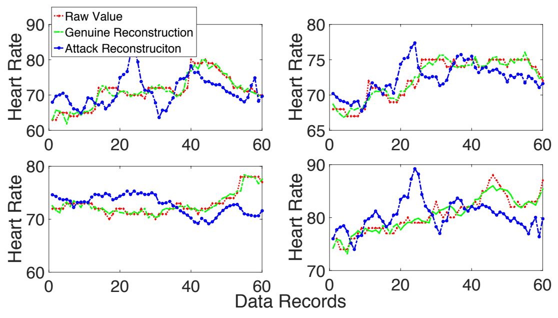


Fig. 10: Heartbeat data recovery. Four random heartbeat data columns (60 data records each column) selected to show the difference between genuine reconstruction and attack reconstruction.

9.2 Resource Consumption

At this stage, most IoT devices need a gateway to provide stable Internet connection (e.g. Fitbit, Microsoft Band). That is why we need to adopt an Android smartphone as part of the client to form a reliable IoT environment. (For wearable devices providing Internet connection (e.g., iWatch from Series 3), Kryptein can be directly installed on that.) Therefore, we also conduct experiments and measurements on ultra-low power wireless micro-controllers widely used by mainstream IoT devices to illustrate how Kryptein performs with IoT devices in isolation.

9.2.1 *Measurement on Smartphones*

We used approximately 3,000 heartbeat data records collected in 6 hours to evaluate resource consumption of different systems (normal database management system(MySQL), CryptDB, Talos, and Kryptein). This was repeated 25 times to calculate a 95% confidence interval (C.I). Since the processing time is quite short, to reduce the measurement noise we measured and recorded the total time of 25 times to show the comparison result. Table.2 shows the total time used to upload the real heartbeat data. Kryptein is approximately 250 and 160 times faster than CryptDB and Talos respectively. It has to be explained here that Kryptein is much more faster is not only because the encryption method is faster but also because Kryptein processes and uploads data in segment unit (for example, 60 heartbeat data will be compressed and upload as one segment), which other (encryption) systems have to process and upload data point one by one.

TABLE 2: Time for each system to insert data into the database

System	Sample $\bar{x}$ (s)	C.I(s)
Normal DBMS	226.02	[222.93 , 229.10]
CryptDB	462.02	[455.95 , 468.10]
Talos	326.32	[324.19 , 328.46]
Kryptein*	1.89	[1.83 , 1.95]

Table.3 shows the power consumption on the client side (a Motor E Android smartphone). Since we implemented our prototype system on the Android smartphones, we used the Trepn Power Profiler [36] to measure energy consumption of the smartphones. It shows that Kryptein (0.27mW) consumes approximately 170 and 120 times less energy than CryptDB (34.29mW) and Talos (26.85mW) respectively

Kryptein decouples the computation between client and server to balance the secrecy and efficiency. To assess the efficiency, we only concern about the time and energy consumed in the client end as we assume the server has no resource limitation. To make the measurement general and fair, we only compare with other two secrecy systems over the data insertion procedure (encryption and uploading), which can be found in all three systems and requires resource on client devices only.

TABLE 3: Power consumption for each system to insert data

System	Sample $\bar{x}$ (mW)	C.I (mW)
Normal DBMS	16.49	[14.96 ,18.01]
CryptDB	34.29	[31.06,37.50]
Talos	26.85	[23.83,29.86]
Kryptein*	0.27	[0.23,0.31]

We tested Kryptein Layer 2's and Layer 3's energy and time consumption for calculating AVG separately and the results are shown in Table 4. The complexity of Layer 2 to AVG calculation is about O($m^2 + m$) and for Layer 3, it is O($n^2 + 1$), which $m \ll n$. Only the calculation on client device counts, because it is assumed that the Cloud has no resource limitation. Besides, Layer 3 has a higher system consumption in communication. Layer 2 only sends back the compressed data from Cloud to client. The experiments are under our school's WiFi environment (upto 300Mbps for one access point) with 802.11i. Our experiments were carried out under an average 4MB/s uploading speed. Two layers almost take the same amount of time to calculate the AVG and send it back to the client. However, the result shows that Layer 3 (18.3 mJ) will take almost 12 times more energy than Layer 2 (3.3 mJ). This shows our DoD structure is more energy efficient.

We implemented AES128−Zip as a baseline comparison method to show Kryptein is superior as a compression and encryption method. It is clear that AES128−Zip has no data fidelity loss and Kryptein will make the cipher smaller as the trade-off of accuracy loss. We chose AES128−Zip as comparison since it is widely used in data compression. The result is shown in Table 5. AES128−Zip adopts zip4j [37] to implement the compression part and then encrypt the compressed zip data file with AES-128. We did not implement it in reverse order because cipher is usually less compressible. The parameter setting for zip and compression are both normal (medium time consumption and medium compress level). We tested the two compress and encryption system on 10min, half hour, 1 hour and 10 hour heart rate data.

TABLE 4: Energy and time consumption for AVG calculation

	Computing		Communication(WiFi)	
Consumption	Time	Energy	Time	Energy
Layer2	1.8ms	826μJ	0.9s	1.5mJ
Layer3	1.1ms	24μJ	1.2s	18.3mJ

TABLE 5: Efficiency comparison for different data length after compressed and encrypted

	AES128-Zip		Kryptein	
Original	Time(ms)	Cipher(byte)	Time(ms)	Cipher(byte)
10min	109	192	86	41
30min	112	544	102	123
1hour	115	1088	107	246
10hour	167	10816	111	2460

Table 5 shows that the computation time and cipher text length of two methods compressing and encrypting data records. It shows that Kryptein outperforms AES128−Zip in both computation time and cipher text length. In particular, the cipher texts produced by Kryptein are approximately 5 times shorter than those produced by AES128−Zip. A more comprehensive comparison with specially designed IoT cipher will be detailed in the next section.

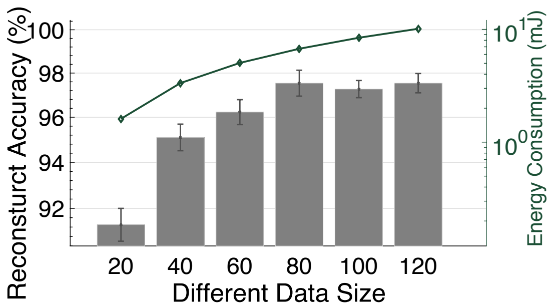


Fig. 11: Different dictionary size impacts. Note log-scale for Y-axis on the right hand side.

Fig. 11 illustrates the reconstruct accuracy and power consumption trade-off when Kryptein adjusts for different dictionary sizes. When we take a 0.3 compression ratio as in our previous experiments (see section 9.1.2), it shows that 0.3 could already achieve a good accuracy. When the dictionary size reaches 60 for our heartbeat data, Kryptein can achieve a good reconstruct accuracy (about 96%). Though Kryptein can adjust dictionary size to a larger dictionary size (e.g., 80, 100, 120), the accuracy can only improve less than 2% but will cost more than twice the energy than the dictionary size of 60. Besides, a larger dictionary size will result in a longer system latency because Kryptein client has to accumulate 80, 100, 120 data records to run compression and encryption.

9.2.2 Measurement on IoT Devices

In this part, we investigate the energy consumption of the proposed system using state-of-the-art wearable systems. We selected the Texas Instrument SensorTag as the target device, which is embedded with the ultra-low power CC2650 2.4GHz wireless micro-controller. It is widely used by today's mainstream wearable devices such as Fitbit. Our SensorTag is running with the Contiki 3.0 [38] which is an open source operation system for IoT devices. The experiment setup for the power measurement is shown in Fig. 12. We apply the INSTRUSTAR ISDS205X oscilloscope[6] to precisely measure the energy consumption in both computation and data communication of the proposed system. We connect the SensorTag with a 10Ω resistor in series and power it using a 3V coin battery. The oscilloscope probe is then connected across the resistor to measure the current going through it.

To make the measurement comply with the previous experiments, we still use one segment of heartbeat data with size 60 as the input and measure the average consumption of the system by repeating the Kryptein function for ten times. To accurately measure our system performance on

6. Different oscilloscopes are used compared to raw conference paper [39], but same experiment design is kept.

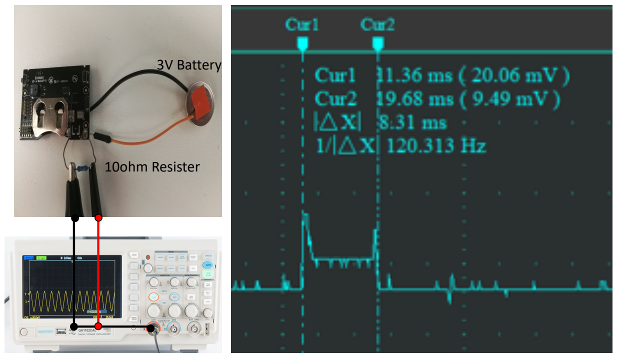


Fig. 12: Power measurement illustration.

SensorTag, we disabled all the irrelevant modules (i.e., the processing of the on-board sensors and LEDs) on SensorTag during the experiment.

TABLE 6: Energy and time consumption for Kryptein running on SensorTag.

	Computing			Communication	
Compression	Time(ms)	Voltage(mv)	Energy(μJ)	Data Size (Bytes)	Energy(μJ)
6	8.31	14	34.90	43	9.53
12	8.7	14	36.54	86	19.07
30	8.83	14	37.09	210	47.68

The results are shown in Table 6. The system energy consumption comes from two parts, namely the consumption in computation of Kryptein and the cost in transmitting the compressed and encrypted data. The 60 heartbeat data are compressed and encrypted to cipher data with a size of 6 (compressed to 10%), 12 (20%) and 30 (50%) separately, and then transmitted using a Bluetooth low energy (BLE) beacon. Assuming the same underlying platform we used in our power measurement and the system performs the computation and transmission every 10 minutes, the overall energy consumption in compressive-encryption, and data transmission in this scenario is totally 0.16J per day. This contributes approximately 0.02% of the daily energy budget of a Fitbit device with 5 days lifetime and a lithium-polymer battery capacity of 4.32kJ.

9.2.3 Overhead

Learning a sparse presentation dictionary will indeed increase the system overhead compared to commonly used random matrix like Gaussian matrix and Bernoulli matrix. However, the overhead is not a big issue/constraint to commonly home devices (e.g., personal computer, smart phone, etc.). In contrast, the data reconstruction accuracy is significantly improved [40]. We tested the dictionary learning process on a 3 GHz clock speed CPU (8th Generation Core i5). The SPAMS [8] is used as the optimization toolbox (for solving various sparse estimation problems). We adopt the setting from private health monitoring scenario. 24 Hours heart rate data (around 8K heart rate readings) is used to train a 60x60 dictionary matrix. We took 500 iterations in the learning process, which is consistent to the evaluation setting. It takes about 75s to complete the dictionary training process and 102KB for the training data storage. The dictionary learning algorithm take about 20MB extra RAM when running. And only single CPU is used during the dictionary training process. Thus, the insignificant extra overhead won't be a challenge for normal home devices.

TABLE 7: Parameters Selection of Simon and Speck.

Simon			Speck		
Key Size (bit)	Round	Cipher Block Size (bit)	Key Size (bit)	Round	Cipher Block Size (bit)
64	32	32	64	22	32
128	44	128	128	32	128
256	72	128	256	34	128

10 Comparison with Lightweight Ciphers

National Security Agency (NSA) published a family of lightweight block ciphers: Simon and Speck, targeting on maintaining an acceptable level of security on a diverse collection of IoT devices [41]. Speck has been delicately designed and optimized for software implementations, while Simon's focused on hardware implementations. To meet the various demands in complex IoT environment, Simon and Speck are provided with different parameters (key size and encryption round) to balance the security protection level and processing efficiency on IoT chips.

The parameter selection when we implement Simon and Speck approaches is illustrated in Table. 7, which is also reported in [41], [42]. It ensures a better security and efficiency trade-off for most IoT scenarios. To have a better understanding of the performance of the lightweight ciphers, we use Simon, Speck and Kryptein to encrypt and decrypt a series of heart rate data points (around one minute) and measure their time and energy consumption respectively. The energy consumption is benchmarked using the equipment shown in Fig. 12.

The results of the computation time and energy consumption on the IoT device are shown in Table. 8 which are derived from averaging over 20 measurements. From the results, we can find Simon is more efficient than Speck on SensorTag when smalle key size (e.g., 64-bits and 128-bits) and few encryption rounds (e.g., 32 and 44) are chosen. In some cases, the energy consumption of Simon and Speck is not consistent. By that it implies that some parts of its encrypt/decrypt statement will consume more energy than others. It demonstrates as the 'spike' in the power measurement illustration (refers to Fig. 12). The inconsistent energy consumption is also measured and noted in the Table. 8. The digits in brackets represent for the various voltage and its last time respectively when fluctuated energy consumption (or spike) happens. For every 1 minute's heart rates, Kryptein takes about 20% less energy with roughly same computation time (for both encryption and decryption). This is due to the spike occurred in Simon and Speck always refers to a higher momentary voltage (than the stable period), which cause a higher total power consumption. Contrast to Simon and Speck, the power consumption of Kryptein is stable and the spike period is very short (less than 1 ms). Specificaly, compared with the most efficient computing case (Simon with 128_128), IoT devices (data) running our proposed system can achieve about 35% extended lifetime.

TABLE 8: Comparison with the lightweight encryption methods for IoT: Simon and Speck

		Simon			Speck			Kryptein		
		Time (ms)	**Voltage (mw)**	**Energy Consumption (μj)**	**Times (ms)**	**Voltage (mw)**	**Energy Consumption (μj)**	**Time (ms)**	**Voltage(mw)**	**Energy Consumption (μj)**
Encryption	64_32	5.45	27	44.15	6.10	27	49.41			
	128_128	5.50	27	44.55	8.70(3.64)	27(14)	50.73	8.31	14	**34.90**
	256_128	12.6(5.32)	27(14)	73.60	8.83(3.51)	27(14)	50.75			
Decryption	64_32	8.57(4.94)	27(14)	55.26	6.88	27	55.73			
	128_128	6.1	27	49.41	8.50(3.20)	27(14)	48.18	8.57	14	**35.99**
	256_128	13.12(5.32)	27(14)	75.86	8.69(3.38)	27(14)	49.68			

11 Long-term Usability Study

There is a common concern on privacy or security-related systems using bio-information as the 'key': whether the system is robust to long-term usage when the user's bio-information changes gradually overtime. For example, human's gait information is often used as authentication criterion however it is very likely to change when one's physical body status changes (e.g., knee injuries) [43]. So we investigate if the performance (reconstruction accuracy) of Kryptein is vulnerable under such change, i.e.,if the dictionary generated from historical data (e.g., previous heart rates) will fail to compress/reconstruct the data accurately after being used for a long time.

To address this concern, we conduct a long term experiment with Kryptein on heart rates compression and reconstruction. The dataset collection phase spans over half year (34 weeks) and 3 to 8 hours' heart rate series are collected in each week. The experiments are not under controlled environment, users are in their normal daily routines and activities (walking, jogging, eating, ect) when collecting the heart rates.

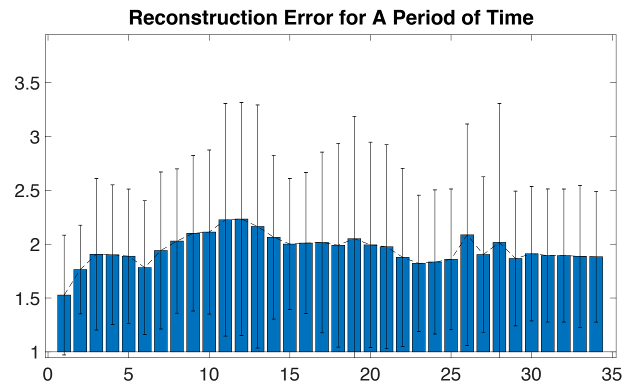


Fig. 13: Reconstruction error on a period of time hear rate data. y-axis is the real reconstruction error, x-axis represents for 34 weeks data is collected for test.

The raw heart rates data is collected without sophisticated preprocessing (e.g., filtering). They are fed directly to Kryptein and stored in the Cloud. We use the first week's (about 4 hours) heart rates to train the dictionary and generate the encryption key based on the learned dictionary. Then the dictionary is fixed and the reconstruction error of Kryptein over the 34 week's is illustrated in Fig. 13. The results show that the first week's reconstruction error is the smallest (about 1.5) since the key is generated from the same set. For the following 33 weeks, the average reconstruction error fluctuates near 2 and no explicit trend of performance loss is observed overtime. Considering the inherent error caused by the hardware and the range of heart rates, the average reconstruction error is reasonable (the average of those heart rate data is about 76). Consequently, we can claim that there is no need for Kryptein to update its dictionary for a long-term utility performance.

11.1 Security Analysis with Long-Term Key

The privacy analysis section (see Section 8) has demonstrated the proposed CS-encrypted system is secure in the "one-time-padding" situation. However, in the practical scenarios, those encryption keys are used multiply times (or in a long term). Thus, in the worst case (i.e., the Cloud get compromised), our cryptosystem cannot achieve prefect secrecy in the multiple-time-padding scenario [24].

To better illustrate the privacy protection and discuss the protection guarantee provided by our cryptosystem in long-term cases, we adopt the extended Shannon-sense (ESS) perfect secrecy and extended Wyner-sense (EWS) perfect secrecy definition from [44]. In our proposed cryptosystem $Y' = AX$ (refers to Equation 7), the ESS perfect secrecy then can be described as:

$$G(\mathrm{Y}', \mathrm{X}, \mathrm{A}) = I(\mathrm{Y}'; \mathrm{X}) + I(\mathrm{Y}'; \mathrm{A}), \tag{12}$$

where I represents the mutual information, Y' represents ciphertext, A represents the key and X represents the plaintext. Thus, it is ESS perfect secrecy if $G(\mathrm{Y}', \mathrm{X}, \mathrm{A}) = 0$. To estimate the ESS perfect secrecy, the EWS criterion is adopted, since the criterion [45] has been used to measure the security of the compression-involved cryptosystem [46], which is similar to our CS-based cryptosystem. Therefore, the ESS perfect secrecy then can be described as:

$$\lim_{n\to\infty} \frac{G(\mathrm{Y}', \mathrm{X}, \mathrm{A})}{n} = 0, \tag{13}$$

where n denotes the length of the plaintext, thus, we can have:

$$\lim_{n\to\infty} \frac{G(\mathbf{Y}', \mathbf{X}, \mathbf{A})}{n} = \lim_{n\to\infty} \frac{I(\mathbf{Y}'; \mathbf{X}) + I(\mathbf{Y}'; \mathbf{A})}{n} \tag{14}$$

From [44], one can obtain:

$$I(\mathbf{Y}'; \mathbf{X}) \geq \sum_{i=1}^{N} I\left(\mathbf{y}'_i; \mathbf{x}_i\right), \tag{15}$$

and from the Lemma in [25], one can see that each ciphertext is not independent of the corresponding plaintext, i.e., $\forall i, \exists \delta > 0$ which results in:

$$I\left(\mathbf{y}'_i; \mathbf{x}_i\right) \geq \delta \tag{16}$$

Assume the same key is used N times, then we can have:

$$\begin{aligned}\lim_{n\to\infty}\frac{G(\mathbf{Y}',\mathbf{X},\mathbf{A})}{n} &= \lim_{n\to\infty}\frac{I(\mathbf{Y}';\mathbf{X})+I(\mathbf{Y}';\mathbf{A})}{n}\\ &\geq \lim_{n\to\infty}\frac{I(\mathbf{Y}';\mathbf{X})}{n}\\ &\geq \lim_{n\to\infty}\frac{\sum_{i=1}^{N} I\left(\mathbf{y}'_i;\mathbf{x}_i\right)}{n}\\ &\geq \lim_{n\to\infty}\frac{N\delta}{n}\\ &\geq \delta\\ &> 0.\end{aligned}$$

Thus, if the same key is used $N \geq n$ times, the proposed cyrptosystem cannot achieve the EWS perfect secrecy, the detail proof can be referred in [44]. Back to our long-term usability scenario, even the same key won't affect utility accuracy, in worst case, we suggest to update the dictionary key after being used for 60 times (the learned sparse representation matrix dimension for heart rate date is 60×60) for better privacy protection purpose.

12 Analytics with Kryptein

One of those significant reasons facilitates the IoT ecosystem blooming is about the data generated by those seamless connected sensors also help people gain more insights from the cyber physical world. Usually, analytics tasks like clustering, regression, classification, etc., are conducted on those collected IoT data to help the public interact with the physical world better. However, the most common approach for those analytics model deployed today is about data owners storing their data at 'trusted' third-party service providers (SP) who can observe the raw data values. However, the plain data for analytics tasks are always a big concern for the public due to the fact there is no guarantee the data won't be leaked or misused by the SP [47], [48]. To address this concern but continuously work with those honest-but-curious SP, many privacy-preserving data analytics are proposed. However, the efficiency, privacy and utility trade-off has to be analyzed before being used for the various IoT applications.

We investigate the analytics performance on the proposed systems on a real-world scenario: face recognition/classification on camera-enabled IoT devices. Face images collected by IoT devices like IP camera are widely used in security surveillance systems. Take the airport's surveillance camera system as an example, the security group who deploys the IP camera is taking the face images from government department for training the identity recognition system purpose. The face images are sensitive which may reveal person identity information once accessed by the malicious party. Besides, the training set (face images) remained in the analytics model may pose a new threat to the privacy issues, since sensitive information in the training data may be revealed by malicious people via utilizing careful analysis [49].

With the help of Kryptein, we can CS-encrypted the face data before sending to the untrusted Cloud for analytics learning. In this case, each face image is normalised (to range [0,1]) then transferred into ciphertext domain (see y' in Algorithm 1). As long as those face data is CS-encrypted by the same key used in Kryptein (the inner distances among the data are remained), the cipher stored on the Cloud can be directly used for most of linear-regression-related analytics tasks. IoT devices embedded with Kryptein and key then can send the new CS-encrypted recording face image to Cloud for human identity recognition.

Our proposed new IoT encryption primitive is considered to address this privacy concern. Since no plain text (clear) is ever stored/revealed in the 'honest-but-curious' server, extract useful information from the analytics model is as difficult as reconstruct it from the cipher (refers to Section 8). Since the efficiency of Kryptein on IoT has already been evaluated (refers to Section 10), we will evaluate the utility for analytics and the privacy protection in this following section.

We test our proposed system using Orl dataset [50]. The face images in Orl database are greyscale. The raw size of each image is 92×112 pixels. There are 40 subjects and 10 images for every subject. In our evaluation, we select 8 images of each subject for training and rest 2 images for testing. This refers to the real-world scenario that the surveillance camera system contains some existing face images for building analytics models then recognises the appointed suspect identity in real-time video surveillance.

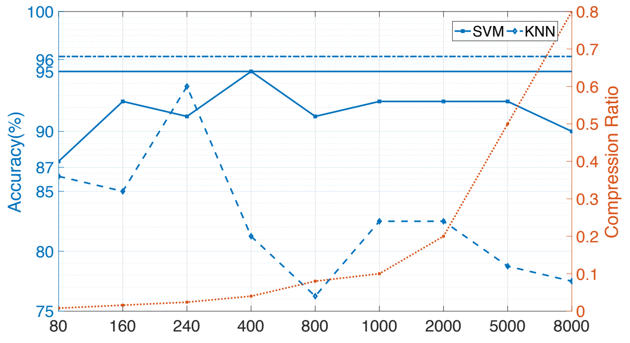


Fig. 14: Classification accuracy on various compression ratio.x-axis represents for the dimension after compression.

Kryptein only stores the compressed and perturbed data on the Cloud in the default setting, thus the Cloud cannot do anything but storage. However, for the further application purpose, if the user is willing to share partial key with Cloud (same as Bob's key), the Cloud can directly run analtyics tasks like face recognition. Thus, the Cloud can provide many useful analytics services but without (the capability of) revealing the raw sensitive data (appearance of the face). The evaluations on classification tasks are performed with commonly used classifiers: K-nearest neighbour (KNN) and support vectors machine (SVM) and the results are shown in Fig. 14. The baseline classification accuracy with raw face images is 96.25% and 95% for KNN and SVM respectively while classifiers with the ciphers encoded by Kryptein are able to achieve comparable classification accuracy. For example, when face images are compressed and encrypted from 10,304 dimension to 240, the accuracy of both the classifiers can still achieve above 90%. Considering the energy and time saved (e.g., communication) by Kryptein, the accuracy sacrifice is acceptable.

We then evaluate the performance of Kryptein on privacy protection in the analytics scenario. We use mutual information [51] as our evaluation metric, which measures

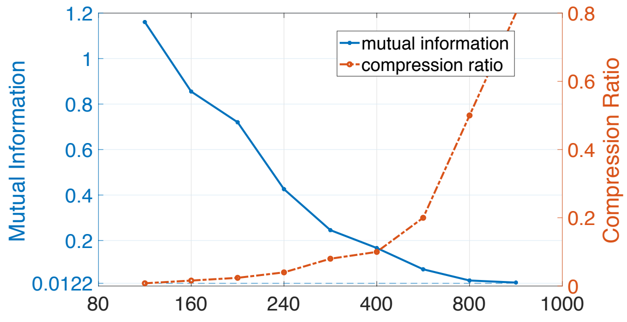


Fig. 15: Mutual information on various compression ratio. x-axis represents for the dimension after compression.

the distance between the original training instance and the CS-encrtyped instance stored on the Cloud. The mutual information of two variables A and B can be computed using the probability distributions,

$$I(A,B) = \sum_{a,b} p_{AB}(a,b) log \frac{p_{AB}(a,b)}{p_A(a) \cdot p_B(b)} \tag{17}$$

where $p_A(a)$ and $p_B(b)$ are marginal probability distribution and joint probability distribution $p_{AB}(a,b)$ are statistically independent if $p_{AB}(a,b) = p_A(a) \cdot p_B(b)$. When mutual information of two variables $I(A,B) = 0$, it implies that A and B are absolute independent. In the context of this paper, smaller $I(x,y')$ implies that there is less information can be referenced from y' about x, which implies attacker is hard to reconstruct the exact raw data x from y' (x is the original data and y' is the CS-encrypted data stored on the Cloud).

Fig. 15 illustrates the result about the privacy protection status of Kryptein with mutual information metric. We use 320 bins in our mutual information calculation as there are 320 training images stored in the server. It can be seen from the figure that the mutual information drops as the compression ratio increases. This is due to the fact that Kryptein's perturbation process will involve random noise by multiplying the random matrix. The perturbation matrix has the same dimension as the compressed data, so that the higher compression ratio, the more noise is inserted, which shows a decreasing trend for mutual information. The dot line in Fig. 15 represents for the safest case, using the dimension of raw face data (10,304) for random perturbation matrix, and the mutual information is the lowest at 0.0122, which implies there is very little correlated information between the raw face data and CS-encrypted face data. However, higher compression ratio results in more energy consumption and longer computation time in IoT devices. Overall speaking, it is suggested to choose the compression dimension balancing the efficiency, utility and privacy trade-off for various IoT applications. In our face (Orl dataset) recognition system, 240 is the reasonable dimension.

13 Conclusion

We presented Kryptein, a system that provides an efficient and practical level of confidentiality in the face of three significant threats confronting database applications: curious DBAs, arbitrary compromises of the application server and the DMBS and eavesdropping from the communication tunnel. Unlike other encryption systems, Kryptein takes compressive sensing as the underlying encryption mechanism and takes significantly less energy and computation consumption than the state-of-the-art. Kryptein meets its goals using three different layers: random compressed encryption, statistical computation over cipher and raw data decryption.

The evaluation results show Kryptein outperforms state-of-the-art systems such as CryptDB and Talos by two order-of-magnitudes in terms of computation time and energy consumption in embedded client devices, also outperforms state-of-the-art IoT ciphers (Simon and Speck) by extending about 35% IoT devices' (battery) lifetime. For the statistical data calculation, only a very simple calculation was required. For one segment of data (one matrix multiply), Kryptein achieved high accuracy (96%) on calculating the statistics without requiring to decrypt the raw data. The evaluation results show that it is difficult for attackers to reconstruct the original data from cipher without producing large errors. The experiments also illustrate IoT data variance will not affect Kryptein's accuracy in a long term usage, and the proposed system is also able to support the basic analytics tasks like classification on IoT.

Acknowledgments

The work has been supported by the Cyber Security Research Centre Limited whose activities are partially funded by the Australian Governments Cooperative Research Centres Programme.

References

[1] M. Chui, M. Löffler, and R. Roberts, "The internet of things," *McKinsey Quarterly*, vol. 2, no. 2010, pp. 1–9, 2010.
[2] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun, "Fog computing: Focusing on mobile users at the edge," *arXiv preprint arXiv:1502.01815*, 2015.
[3] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proc. of SOSP*. ACM, 2011, pp. 85–100.
[4] H. Shafagh, A. Hithnawi, A. Dröscher, S. Duquennoy, and W. Hu, "Talos: Encrypted query processing for the internet of things," in *Proc. of SenSys*. ACM, 2015, pp. 197–210.
[5] D. L. Donoho, "Compressed sensing," *Information Theory, IEEE Transactions on*, vol. 52, no. 4, pp. 1289–1306, 2006.
[6] M. R. Mayiami, B. Seyfe, and H. G. Bafghi, "Perfect secrecy using compressed sensing," *arXiv preprint arXiv:1011.3985*, 2010.
[7] R. Rana, M. Yang, T. Wark, C. T. Chou, and W. Hu, "Simpletrack: Adaptive trajectory compression with deterministic projection matrix for mobile sensor networks," *Sensors Journal, IEEE*, vol. 15, no. 1, pp. 365–373, 2015.
[8] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *Proc. of ICML*. ACM, 2009, pp. 689–696.
[9] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani *et al.*, "Least angle regression," *The Annals of statistics*, vol. 32, no. 2, pp. 407–499, 2004.
[10] M. R. Osborne, B. Presnell, and B. Turlach, "A new approach to variable selection in least squares problems," 1999.
[11] M. Elad, "Optimized projections for compressed sensing," *Signal Processing, IEEE Transactions on*, vol. 55, no. 12, pp. 5695–5702, 2007.
[12] M. Zhou, H. Chen, J. Paisley, L. Ren, L. Li, Z. Xing, D. Dunson, G. Sapiro, and L. Carin, "Nonparametric bayesian dictionary learning for analysis of noisy and incomplete images," *Image Processing, IEEE Transactions on*, vol. 21, no. 1, pp. 130 –144, jan. 2012.
[13] A. Paverd, A. Martin, and I. Brown, "Modelling and automatically analysing privacy properties for honest-but-curious adversaries."

[14] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar, "On the privacy preserving properties of random data perturbation techniques," in *Proc. of ICDM*. IEEE, 2003, pp. 99–106.

[15] F. Inc. (2016) Fitbit. [Online]. Available: https://www.fitbit.com/au

[16] M. Inc. (2016) Microsoft band. [Online]. Available: http://www.microsoft.com/Microsoft-Band/

[17] R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin, "The johnson-lindenstrauss lemma meets compressed sensing," *Constructive Approximation*, 2007.

[18] S. Bajaj and R. Sion, "Trusteddb: A trusted hardware-based database with privacy and data confidentiality," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 3, pp. 752–765, 2014.

[19] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," in *Proc. of the VLDB Endowment*, vol. 6. VLDB Endowment, 2013, pp. 289–300.

[20] C. Stergiou, K. E. Psannis, B.-G. Kim, and B. Gupta, "Secure integration of iot and cloud computing," *Future Generation Computer Systems*, vol. 78, pp. 964–975, 2018.

[21] P. Li, J. Li, Z. Huang, C.-Z. Gao, W.-B. Chen, and K. Chen, "Privacy-preserving outsourced classification in cloud computing," *Cluster Computing*, vol. 21, no. 1, pp. 277–286, 2018.

[22] W. Xue, Y. Shen, C. Luo, W. Hu, and A. Seneviratne, "Acies: A privacy-preserving system for edge-based classification," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 914–919.

[23] M. S. de Brito, S. Hoque, R. Steinke, A. Willner, and T. Magedanz, "Application of the fog computing paradigm to smart factories and cyber-physical systems," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 4, p. e3184, 2018.

[24] C. E. Shannon, "Communication theory of secrecy systems*," *Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[25] Y. Rachlin and D. Baron, "The secrecy of compressed sensing measurements," in *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*. IEEE, 2008, pp. 813–817.

[26] S. Agrawal and S. Vishwanath, "Secrecy using compressive sensing," in *Information Theory Workshop (ITW), 2011 IEEE*. IEEE, 2011, pp. 563–567.

[27] C. Wang, B. Zhang, K. Ren, and J. M. Roveda, "Privacy-assured outsourcing of image reconstruction service in cloud," *IEEE Transactions on Emerging Topics in Computing*, vol. 1, no. 1, pp. 166–177, 2013.

[28] S.-Y. Chiu, H. H. Nguyen, R. Tan, D. K. Yau, and D. Jung, "Jice: Joint data compression and encryption for wireless energy auditing networks," in *Sensing, Communication, and Networking (SECON), 2015 12th Annual IEEE International Conference on*. IEEE, 2015, pp. 453–461.

[29] Z. Huang, W. Du, and B. Chen, "Deriving private information from randomized data," in *Proc. of SIGMOD*. ACM, 2005, pp. 37–48.

[30] F. Li, J. Sun, S. Papadimitriou, G. A. Mihaila, and I. Stanoi, "Hiding in the crowd: Privacy preservation on evolving streams through correlation tracking," in *Proc. of ICDE*. IEEE, 2007, pp. 686–695.

[31] S. Papadimitriou, F. Li, G. Kollios, and P. S. Yu, "Time series compressibility and privacy," in *Proc. of VLDB*. VLDB Endowment, 2007, pp. 459–470.

[32] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *Proc. of WWW*. ACM, 2009, pp. 791–800.

[33] S. Gambs, M.-O. Killijian, and M. N. del Prado Cortez, "Show me how you move and i will tell you who you are," in *Proc. of SPRINGL*. ACM, 2010, pp. 34–41.

[34] J. Liu, B. Priyantha, T. Hart, H. S. Ramos, A. A. Loureiro, and Q. Wang, "Energy efficient gps sensing with cloud offloading," in *Proc. of SenSys*. ACM, 2012, pp. 85–98.

[35] S.-Y. Juan, Y.-F. Chung, C.-T. King, and C.-H. Hsu, "Cegf: corner extraction by gps filtering for power-efficient location uploading," in *Proc. of MobiSys*. ACM, 2013, pp. 537–538.

[36] Q. T. Inc. (2016) Tren power profiler. [Online]. Available: https://developer.qualcomm.com/software/trepn-power-profiler

[37] zip. (2014) Zip4j:java library to handle zip files. [Online]. Available: http://www.lingala.net/zip4j/

[38] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *Proc. of LCN*. IEEE, 2004, pp. 455–462.

[39] 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 *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2017, pp. 169–180.

[40] R. Rana, M. Yang, T. Wark, C. T. Chou, and W. Hu, "Simpletrack: Adaptive trajectory compression with deterministic projection matrix for mobile sensor networks," *IEEE Sensors Journal*, vol. 15, no. 1, pp. 365–373, 2014.

[41] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "Notes on the design and analysis of simon and speck." *IACR Cryptology ePrint Archive*, vol. 2017, p. 560, 2017.

[42] Z. Liu, Y. Li, and M. Wang, "Optimal differential trails in simon-like ciphers," *IACR Transactions on Symmetric Cryptology*, pp. 358–379, 2017.

[43] W. Xu, Y. Shen, Y. Zhang, N. Bergmann, and W. Hu, "Gait-watch: A context-aware authentication system for smart watch based on gait recognition," in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*. ACM, 2017, pp. 59–70.

[44] Z. Yang, W. Yan, and Y. Xiang, "On the security of compressed sensing-based signal cryptosystem," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 3, pp. 363–371, 2014.

[45] A. D. Wyner, "The wire-tap channel," *Bell system technical journal*, vol. 54, no. 8, pp. 1355–1387, 1975.

[46] M. Johnson, P. Ishwar, V. Prabhakaran, D. Schonberg, and K. Ramchandran, "On compressing encrypted data," *IEEE Transactions on Signal Processing*, vol. 52, no. 10, pp. 2992–3006, 2004.

[47] M. Weinstein, "What your fitbit doesn't want you to know," https://www.huffpost.com/entry/what-your-fitbit-doesnt-w_b_8851664?guccounter=1.

[48] M. Burgess, "Stravas data lets anyone see the names (and heart rates) of people exercising on military bases," https://www.wired.co.uk/article/strava-military-bases-area-51-map-afghanistan-gchq-military.

[49] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," *arXiv preprint arXiv:1610.05755*, 2016.

[50] F. S. Samaria and A. C. Harter, "Parameterisation of a stochastic model for human face identification," in *Proceedings of 1994 IEEE Workshop on Applications of Computer Vision*. IEEE, 1994, pp. 138–142.

[51] D. Agrawal and C. C. Aggarwal, "On the design and quantification of privacy preserving data mining algorithms," in *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2001, pp. 247–255.

PLACE PHOTO HERE

Wanli Xue Biography text here.

Chengwen Luo Biography text here.

Yiran Shen Biography text here.

Rajib Rana Biography text here.

Guohao Lan Biography text here.

Sanjay Jha Biography text here.

Aruna Seneviratne Biography text here.

Wen Hu Biography text here.