POSTER: Computations on Encrypted Data in the Internet of Things Applications

Laurynas Riliskis, Hossein Shafagh*, Philip Levis
Computer Science Department, Stanford University
*Department of Computer Science, ETH Zurich, Switzerland
{lauril, pal}@cs.stanford.edu, *shafagh@inf.ethz.ch

ABSTRACT

We identify and address two primary challenges for computing on encrypted data in Internet of Things applications: synchronizing encrypted data across devices and selecting an appropriate encryption scheme. We propose a caching mechanism that operates across the three devices, enabling interactive order-preserving encryption schemes on resource-constrained devices. Additionally, the system can use a high-level description of an IoT application to select automatically appropriate encryption for the data on corresponding tiers and their mathematical operations. This assists in fine-tuning and choosing the core parameters for underlying data structures.

Categories and Subject Descriptors

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

Keywords

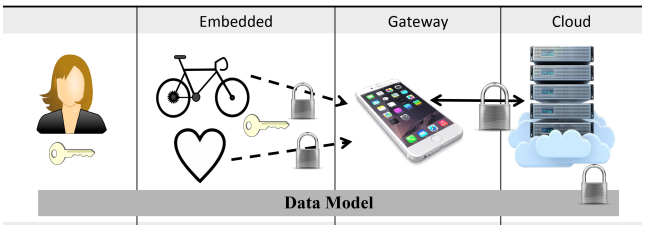
Internet of Things; System; Computing on Encrypted Data

1. INTRODUCTION

The Internet of Things (IoT) encompasses a huge variety of applications, ranging from fitness trackers and home automation to smart cars and smart cities. A common thread ties most of these applications together, however. At a system level, they operate across three tiers of devices as shown in Figure 1: embedded, gateway and cloud. Each tier has different storage, computation and energy resources. Embedded devices collect, compute on, and react to sensitive and private information such as vital signs and living habits. After collecting the information, embedded devices send the data to gateways, which compute, display, and/or further relay it to the cloud.

The tiers have different processors, run different operating systems, and use different programming languages. This diversity creates a huge attack surface. Ideally, to protect user data, an application should perform its computations on encrypted data, only decrypting when its owner views it.

CCS'15 October 12-16, 2015, Denver, CO, USA
ACM 978-1-4503-3832-5/15/10.
http://dx.doi.org/10.1145/2810103.2810111.



	Embedded	Gateway	Cloud
Computation	few 10s of MHz, few KB RAM	Snapdragon, Cortex 2 GB RAM	Xeon 10's of GB
Communications & OS	20 B (BLE), 127 B (ZigBee); embedded OS	1500 B (WiFi, LTE); Android, iOS	1500 B (Ethernet); Linux, Windows
Storage	few MB	few 10s of GB	Unlimited
Power supply	200 mAh Battery	3000 mAh Battery	Unlimited

Figure 1: An example of common IoT architecture.

IoT applications need new security architectures to support computing on encrypted data. Existing practical systems that process encrypted data, such as CryptDB [6], are designed for web services. For example, CryptDB selects and performs necessary cryptographic operations on a data flow between the app server and database, considering only the cloud as untrusted. In contrast, an IoT application stores and processes data on gateways as well as the cloud. This introduces an additional untrusted entity. Smartphones can be root-kitted, which would give an adversary access to personal data. Additionally, there are many IoT systems with a untrusted gateway, for example, home automation gateways can run arbitrary software.

IoT applications need new cryptographic algorithms and protocols. The resources available to IoT devices vary by up to six orders of magnitude (i.e., kB to GB of RAM and MHz to GHz of CPU power). Each tier has a different system architecture and communication interface (i.e., BLE and ZigBee, to WiFi, LTE, and Ethernet). Gateway and IoT devices are battery powered and so need careful power management. A traditional energy conservation technique is radio duty-cycling: the radio turned on only for brief periods, saving energy but increasing latency [2]. This approach works well for bursts of traffic (e.g., web browsing), but works poorly for interactive exchanges of small packets, such as used in *Order-Preserving Encryption*. mOPE [7] requires $O(log\ n)$ rounds for an insert in the worst case. In the IoT context, this results in increased energy consumption and higher communication latency due to unstable connectivity.

IoT applications must be able to support delay-tolerant operations [3], that is, networking when devices are not always connected. As in the case of an athlete viewing his

performance metrics and vital signs in the field, or a BLE enabled smart door lock, the user wants a functional app even if an associated smart phone gateway briefly does not have data coverage. Therefore, rather than relying on always-on Internet connectivity, each tier must locally store a subset of the data and synchronize when connectivity is available.

2. PROBLEM STATEMENT AND CONTRIBUTION

We identify and address two primary challenges to facilitate computation on encrypted data in IoT applications: *(i)* synchronizing encrypted data between devices for order-preserving schemes, and *(ii)* automatically selecting an appropriate encryption for the corresponding tier and mathematical operation.

As a further contribution, we describe a programming approach that analyzes the high-level description of an IoT application and subsequently synthesizes protocols from the available cryptographic primitives. The program investigates relations between data computation and distribution, and underlying link layer properties. Finally, the program generates the necessary code to adapt the appropriate encryption scheme to the three-tier architecture of the IoT application.

3. RELATED WORK

Homomorphic encryption [1] allows arbitrary mathematical operations on encrypted data. Such operations are yet to become practical. Currently, they require quixotic computational resources. Thus, many alternative paths have been explored.

Encryption Mechanisms. The foundation of most SQL-like databases rests on simple mathematical operations such as equality, order, sum and mean. Consequently, much of the research has focused on encryption models that allow these mathematical operations at a more practical computational cost. Given the constraints of IoT applications, we consider four classes of encryption schemes as economical to use. These encryption schemes based on complexity and required resources are divided into two categories.

The first category includes *random* and *deterministic* encryptions, which use efficient AES-based schemes. *Random* is a probabilistic encryption with the highest degree of security, but does not allow any computation. *Deterministic* encryption enables equality check but has a lower level of security. The second category includes high complexity and high overhead schemes: *order-preserving* and *additive homomorphic* encryptions. The *order-preserving* encryption allows ordering of the encrypted data. *Additive homomorphic* homomorphic encryption enables addition while preserving high security. Both perform the computation of the plaintext values in the ciphertext space.

Existing Implementations of the Mechanisms:

- CryptDB, an encrypted query processing system for web services, uses a proxy to intercept communications between an application server and the database. CryptDB also applies the cryptographic schemes, translates queries, performs necessary crypto operations and handles the key management. In contrast, an IoT application has a second untrusted party: the gateway, that demands a more sophisticated approach. Additionally, the system needs to handle disconnects and operates on both locally stored data as well as in the Cloud.

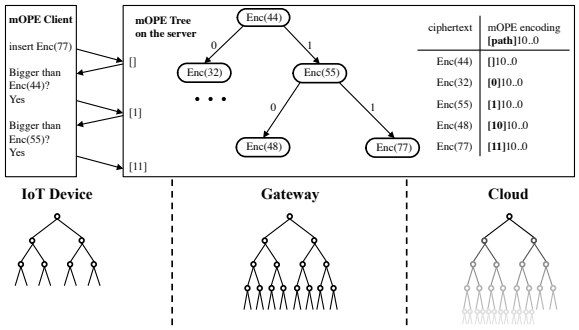


Figure 2: mOPE is an interactive order-preserving encryption approach relying on Binary Search Tree (BST) to order encrypted data.

- mOPE [7] is an `interactive` encryption mechanism with mutable ciphertext. That introduces a challenge: *how to maintain the consistency of data among different tiers of IoT?* mOPE assumes a client-server approach (see the upper part of Figure 2), where the server that stores the data is honest-but-curious [5]. The server maintains a balanced binary search tree (BST) whose nodes are encrypted values. The path from the root to the node containing the value depicts the order-preserving encoding, resulting in the server learning only the order information and nothing more. Such a solution is not applicable in a three-tier architecture because the different devices cannot store same size trees. Moreover, each insert takes $O(log\ n)$ round trips that would have a significant performance impact. The authors suggest usage of a local cache table to reduce insert overhead, however, that does not solve the three tier architectural challenge. In an attempt to address the communication overhead of mOPE, the authors in [4] maintain the tree and encodings locally and only insert the encoding in the database. Although this reduces the interaction for inserts to $O(1)$ time the required storage to maintain the local tree is too large for the IoT devices. Moreover, this approach implies extensive communication after tree rebalances, (i.e., mutation), which renders it as not suitable for our scenario.
- AutoCrypt [8] is a compiler that transforms a plain C application to one that computes on encrypted data. AutoCrypt combines and converts between partially-homomorphic encryption (PHE) schemes. Similarly as AutoCrypt, we generate code for the user but across all tiers of the IoT architecture. AutoCrypt requires extensive computational resources, for example, one MB computation needs 2 GB of RAM. In our work, we perform reasoning on which algorithm is feasible for the corresponding class of device.

4. INITIAL RESULTS

Attacker Model. Our system addresses the threat model of database/system compromise. In this threat model, the attacker is interested in gaining access to the data stored on either of the tiers. In case of the Cloud, the attack can be launched internally by a curious database administrator of the Cloud, or externally by malicious entities. On the gateway or the IoT device, the attack could be initiated by malware or physical capture. Our system addresses this threat model by storing the data in encrypted form on each tier and facilitating computation on all encrypted data.

Access time	Value	Path	sync
0	44	[]	yes
1	33	0	yes
2	55	1	yes
3	48	10	yes
4	77	11	yes
current	**20**	00[0?1]	

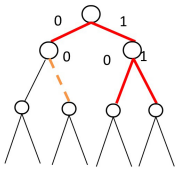


Figure 3: Example of caching strategy for distributed mOPE.

Synchronizing encrypted data between tiers. The three-tiered architecture complicates maintaining the consistency of interactive mechanisms, such as mOPE (see the lower part of Figure 2). Consider an IoT device that collects and encrypts 1024 data samples locally with mOPE. Once the IoT device connects, it uploads its data to the gateway. The gateway stores 8192 values locally and synchronizes its data when connected with the Cloud. When the embedded device or gateway reach storage limit, they start discarding local data according to a policy, like FIFO or LIFO. Thus, the challenge boils down to the synchronization of the increasingly smaller, balanced BST's of the Cloud, gateway, and embedded device.

Approach. Rather than synchronizing trees, we use a caching mechanism that locally stores most frequently used pathways of the global tree. The mechanism operates on a similar idea as $L1$ and $L2$ caches in a modern CPU. The embedded device (with the smallest amount of storage) maintains the smallest cache; the gateway keeps a larger cache and the Cloud stores a full tree. The cache table consists of access time, value, and a path in the tree, and a synchronization flag. Additionally, the cache maintains the information about leaves. Consider the tree depicted in Figure 2 and shown in a table in Figure 3. The embedded device stores red colored pathways in the cache. A new value, 20, has to be inserted in the tree, but the pathway ends at 00. The cache has to retrieve the next node from the gateway. If there is no connection, the local view of the data still signifies valid information so that the last value is the smallest in the local tree. When connected to the gateway, the embedded device will ask for a cache lookup and may or may not find the leaf to place 20. The gateway responds directly with the discovered pathway in the local cache, or after requesting it from the Cloud.

On successful lookup, an additional value will be added to the local cache. Given the table size of 6 entries, the last accessed value must be deleted due to space limitations resulting in a "broken" tree. Because we are caching the path and values, we can insert the value – let say 60 – without knowing the root of the tree but by looking up the path and comparing the next value.

While, cache misses result in communication overhead, the lookups can piggyback on the reliable data delivery mechanisms. On the other hand, cache hits containing exactly the same value; thus we need only to transmit the metadata hence saving energy on the data transmission.

In this initial work, we are in the process of evaluating a variety of caching mechanisms and possible approaches minimizing cache misses. It is our belief that the appropriate caching mechanism will depend on the type of application (e.g sensory data arrival model).

Table 1: Our system creates an access table from the computations performed on the fields of the data model. This knowledge allows selection of a suitable encryption mechanism.

	Device	Gateway	Cloud
Time	None → RAND	Order → OPE	Order → OPE
HR	None → RAND	Comp →DET	Comp → DET Addition → HOM
ID	None → RAND	None → RAND	None → RAND

Selection of the encryption mechanisms. There are frameworks where an IoT application is developed by describing it in a high level of abstraction. These approaches emphasize that IoT applications have a holistic data model spanning all three tiers, even though it is being implemented in several different languages across various types of devices. We depend on this fact to construct the access table of the data model as shown in Table 1. The encryption mechanisms are selected as follows: first we assume the base-case of encrypting all data with the highest level of security (random). Then we walk through the data access, and we determine how a particular field needs to be encrypted. The selection of encryption mechanism depends on the mathematical operations executed on the fields. For example, a gateway comparing heart rate values would imply deterministic encryption. Similarly, in the Cloud, when accessing the data with regards to the order and computing averages, order-preserving and additive homomorphic encryption are needed. Note, that data is never decrypted outside a trusted space across all tiers. As a result, the view from the Cloud will return the encrypted results that are decrypted on the owners device when viewed.

5. FUTURE WORK

This poster presents an initial result of the mechanisms that enable processing of encrypted data in the Internet of Things. We are in the process of implementing prototypes of the described mechanisms. We will evaluate the usability and performance of them by assessing the overhead and quantifying all possible computation, latency, and energy costs.

6. REFERENCES

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

[2] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *ACM MobiSys*, 2012.

[3] S. Jain, K. Fall, and R. Patra. Routing in a Delay Tolerant Network. In *SIGCOMM*, 2004.

[4] F. Kerschbaum and A. Schroepfer. Optimal average-complexity ideal-security order-preserving encryption. In *ACM CCS*, 2014.

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

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

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

[8] S. Tople, S. Shinde, Z. Chen, and P. Saxena. AUTOCRYPT: Enabling Homomorphic Computation on Servers to Protect Sensitive Web Content. In *CCS*, 2013.