This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2864513, IEEE Internet of Things Journal

FPGA-BASED SYMMETRIC RE-ENCRYPTION SCHEME TO SECURE DATA PROCESSING FOR CLOUD-INTEGRATED INTERNET OF THINGS

M. Al-Asli, M. E. S. Elrabaa, and M. Abu-Amara

Abstract— *A new scheme using Field Programmable Gate Arrays (FPGAs) to secure IoT data processing in public clouds against various attacks (including attacks from insiders) is proposed. The proposed scheme supports various business models involving multiple parties and allow the data owner to give temporary access to IoT data to specific clients at a public market place (the cloud). The scheme achieves perfect forward secrecy, provides FPGA authentication, a secure way to establish a symmetric session key between the on-cloud FPGA, the IoT device and the client, and allows user's configuration integrity check while running in the cloud FPGA. A symmetric proxy re-encryption (PRE) scheme is used to support the publish/subscribe mode of operation of IoTs. A complete prototype has been implemented to show the feasibility of the proposed scheme. Formal verification of the proposed protocol verified that it does not have any vulnerabilities. Experimental results showed that an FPGA implementation of the proposed PRE was **6x** faster than the SW implementation in transforming a ciphertext of size 1 Gb.*

Keywords— Cryptographic protocols and algorithms, Hardware Security, Cloud Computing Security, FPGAs, Internet of Things Security

1 INTRODUCTION

Internet of things (IoT) is penetrating all physical fields, including homes, manufacturing and urban spaces, and is expected to continue to grow dramatically in the future, reaching around 21 billion by 2020 [1]. The massive amount of data that is generated by these devices need to be stored, aggregated, and processed if any value is to be made of these data. Data has proven to be the most valuable commodity of our times. As such, many business models have emerged and will continue to emerge to monetize the enormous volumes of data generated every instance. Within these business models several sovereign entities are involved; data generators (IoT devices), data owners, carriers, keepers, one time/regular users, etc. Researchers have already identified many security challenges with IoT data security [2], but this new multifaceted data usage and handling introduces many new ones.

The cloud has emerged as the natural place for storing and processing IoT-collected data. Leading cloud vendors already offer IoT platforms [3]-[4]. To this end, public clouds have become marketplaces for many services that share and handle data. As such, cloud operators/vendors need to provide adequate infrastructure for securing the data and maintain its integrity and privacy when it is stored or traded with other entities for different purposes. This includes protecting the data from insiders' attacks, something that is not fully achievable with current techniques. In addition to conventional encryption and key management techniques, the major thrust in data protection in public clouds is in developing trusted execution environments or processors. However, so far there exist no processor that can really provide a truly isolated execution environment for users' applications such that no information can leak. Field-programmable Gate Arrays (FPGAs) on the other hand can provide such an environment. FPGAs are pre-fabricated integrated circuits that contains up to millions of general logic blocks, interconnect primitives, static RAM blocks (BRAMs), clock generators, DSP blocks, etc. that can be configured by the user to perform any function (i.e. to realize a custom computing machine). FPGA configuration (the equivalent of a context switch in conventional processor-based computing machines) does not require the involvement of operating systems, drivers or compilers, nor any other system software. This reduces the FPGAs' attack surface substantially allowing the use of more robust attack models and stronger security guarantees. FPGAs can be integrated with other cloud resources to form flexible, scalable, independent and secure compute resources within the cloud infrastructure . Clients can use them to perform fast and secure computations on their sensitive IoT data while utilizing the other benefits of the cloud. FPGAs can also be used to build more sophisticated solutions for modern machine-to-machine communications and big data applications [16].

All authors are with the Computer Engineering Dept, King Fahd University of Petroleum and Minerals, Dhahran, 31261 Saudi Arabia.
E-mails: g200464120,elrabaa,marwan@kfupm.edu.sa

IoT-based systems usually follow the publish/subscribe model with data coming from multiple IoT devices (publishers), and is read and processed by one or more clients (subscribers). To secure such data processing in the on-cloud FPGAs even from the cloud providers, a symmetric proxy re-encryption is needed to convert a publisher's ciphertext to another ciphertext that can be decrypted by the subscriber(s) without revealing the original key to the subscriber. Thus, the re-encryption proxy acts like an on-cloud broker allowing data owners to give subscribers access to encrypted data without revealing their master keys. Symmetric proxy re-encryption allows the use of symmetric encryption, which is more efficient when implemented on FPGAs compared to asymmetric encryption [21].

There is a need for new schemes for securing IoT data that can support different business models with multiple entities in different premises involved in data generation, storage, distribution, and processing. In this work, such a scheme is proposed. Utilizing FPGAs and standard security primitives, it guarantees IoT data privacy and integrity against various types of attacks and provides the standard overall protection as outlined in [34] under different use scenarios. It also provides FPGA authentication and ensures the integrity of the user's on-FPGA application, data confidentiality and configuration integrity. The proposed scheme utilizes a specially adapted form of symmetric proxy re-encryption to: 1) allow processing IoT data in the cloud without the need for encrypting the IoT data to specific on-cloud FPGAs, 2) avoid decrypt-then-encrypt process in the cloud which exposes the IoT data to cloud provider, and 3) to enable data sharing among the cloud resources, which allows our scheme to be suitable also for Map-Reduce applications. A complete prototype of the proposed scheme with the proxy re-encryption have been materialized with FPGAs and its performance has been evaluated. The confidentiality and integrity security properties of our proposed protocols was verified using the ProVerif automatic cryptographic protocol verifier. The proposed FPGA structure was verified to be secure against man-in-the-middle attacks and FPGA impersonation.

Next, related works are reviewed in Section 2. In Section 3 an overview of the proposed scheme is introduced, including the protocol, its security analysis, the related framework (HW and SW components), and the proposed symmetric proxy re-encryption. Experimental results are presented in Section 4. This includes the complete implementation of the proposed scheme and the symmetric proxy re-encryption in FPGAs and the details of all components, their implementation details and performance figures. It also provides performance comparisons between our FPGA-based proxy re-encryption and a software version of it implemented in python. Finally, conclusions are presented in Section 5.

2 RELATED WORK

Intel's Software Guard Extensions (SGX) [5] utilizes a set of micro-coded instructions that extend Intel architecture to provide secure execution enclaves. It provides security-sensitive computations with integrity and confidentiality guarantees even from privileged software such as the operating system (OS), the kernel and hypervisors. This is accomplished by allowing the user-level code to allocate private regions of memory, called enclaves, which are protected from other processes (including those running at higher privilege levels). It also provides software attestation to assure the user that their code (and the data it uses) is running in the intended trusted enclave. SGX does not prevent Cache-timing attacks, Physical attacks and Microcode attacks [6]–[9]. Furthermore, using these enclaves to receive the code from one party (e.g. the client) and receive the data from another party (e.g. the IoT device) is not directly supported by SGX; eliminating the benefits of SGX for the IoT data protection.

Other secure processors have been proposed such as Aegis [10], Bastion [11], Sanctum [12], Ascend [13], and Phantom [14]. Again, they provide isolated memory containers for users' applications and address translation mechanisms. To protect against malicious OS that can learn the memory access patterns of the container and timing cache attacks, Sanctum [12] flushes a container's RAM on context switches and makes containers manage their own page tables and handle their page faults, hence the OS cannot learn the virtual address causing the page fault. Still, Sanctum design does not protect against any physical attacks nor does it prevent fault-injection attacks and timing attacks. Ascend [13] and Phantom [14] secure processors make use of oblivious RAMs (RAMs that hide the operation being performed and shuffles their contents periodically to obscure memory access patterns). This however resulted in huge slowdown compared to other processors.

Unlike SGX, which uses the Enhanced Privacy ID (EPID) [15] to preserve the privacy of users using SGX in remote hosts, these secure processors did not guarantee the privacy of the user. Users can be tracked by the identity of the processor they are using.

FPGA-based schemes for securing clients' data in the cloud were proposed in [17,18]. In [17] an FPGA-based security approach for cloud computing that makes use of RSA and its private key to form a root of trust (ROT) inside the FPGA was proposed. Augmenting the cloud's servers with two types of chips that are paired cryptographically by the manufacturer was proposed in [18]. The FPGA would be the processing chip and a custom chip that holds the state between power cycles using none-volatile memory. These solutions however cannot support data aggregated from multiple resources, hence are not suitable for IoT data protection in the cloud. A framework for users' data privacy for Map-Reduce applications in the cloud using the security features of current FPGAs and proxy re-encryption was proposed in [19]. Public/private keys are used for encrypting/decrypting a symmetric key to be shared between the user and the FPGAs in the cloud for data encryption/decryption. The proxy re-encryption

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2864513, IEEE Internet of Things Journal

scheme allows multiple access to the encrypted data from different users with different assigned keys. In addition, they propose using the FPGA's embedded symmetric keys for protecting the configuration bit stream. These embedded keys however are not only unavailable for general users, but were proven to be insecure against known plaintext attacks (KPA) [20]. The proposed scheme also requires a certificate authority (CA) to certify the FPGA's public keys as well as a proxy server to manage key re-encryption. The scheme assumes full trust in the cloud user who will get access to the FPGA's symmetric keys and semi-trust in the proxy and cloud operator.

Research on IoT security mainly falls into efficiently authenticating and securing the IoT devices themselves and/or securing the end-to-end communication between these devices and the cloud. Lightweight authentication methods were proposed to cope with IoT devices' limitations. These methods include homomorphism [22], Elliptic Curve Cryptography (ECC) [23] and Datagram Transport Layer Security (DTLS) protocol based authentication [24]. Commercial cloud-based IoT platforms use industry-standard protocols, such as Transport Layer Security (TLS) and X.509, to secure communication between the IoT devices and the cloud [25]. In addition, several lightweight communication protocols were proposed based on public key infrastructure [26], IPv6/ Low-Power Wireless Personal Area Networks [27], integrated DTLS and Constrained Application Protocol (CoAP) [28] and Secure Sockets Layer (SSL) [29]. Researchers have also proposed various secure architectures and supporting technologies for IoT devices security. Layered security architectures were proposed [30-31] for IoT security and verification, conceptually covering various attacks and mitigation techniques in each layer. Layers cover various techniques related to IoT security such as key management, encryption oracles and access control. A middleware was proposed in [32] to meet the scalability and the high number of heterogeneous devices of the IoT system. The middleware mainly targeted developing a security algorithm to tackle packet sniffing, man-in-the-middle attack and identity spoofing in the IoT environment. An architecture on lightweight identity-based cryptography (LIBC) with ECC was proposed in [33] to solve security issues related to cloud-integrated IoT environment. A dynamic permutation method for obfuscating data encryption in a data processing unit connected to multiple sensors/actuators was proposed in [35]. In [36], an FPGA-based secure architecture for IoT devices is proposed. It utilizes a PUF-based true random number generator and cryptographic cores to protect the IoT device against Trojans/tampering in the field and provide safe communications with other (pre-defined) modules. Another effort for securing FPGA-based IoT devices was reported in [37]. Using a platform FPGA (with an embedded CPU), both the boot image and the configuration bit stream are encrypted and authenticated to protect the design and avoid spoofing and Trojan attacks.

All the above techniques either are concerned with securing the IoT device itself, as a stand-alone or part of a network, or its communication with another entity (the owner of the data). They do not support emerging cloud-based business models that involve multiple entities accessing, analyzing, processing, and using data in public clouds that came from multiple IoT devices. In addition, they neither provide secure and safe methods for IoT owners to license their devices' data, nor did they consider attacks from cloud insiders. I.e. none provided a complete solution that can be practically applied within a public cloud environment. Fig. 1 illustrates a typical application of our proposed scheme in the context of smart cities. IoT devices installed by some entity(s) in many locations collect various information (traffic, pollution, noise, etc.), encrypt it with their private keys and send it for storage in a public cloud. Clients pay the data owners to get temporary re-encryption keys to use this data to obtain useful information for their customers. The re-encryption proxy (PRE) re-encrypt the data with the key established with the FPGA in the cloud, which in turn decrypt the data and process it using the clients' applications and send them the results.

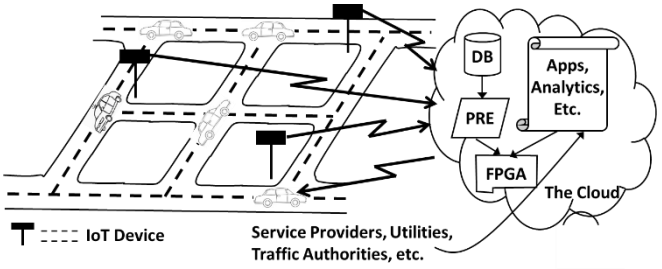

Fig. 1. A typical use case of the proposed scheme in smart cities IoT applications.

3 Overview of the Proposed Scheme

3.1 The In-Cloud FPGA

The in-cloud FPGA devices in the proposed scheme should have the following capabilities (most of which already exist in current FPGAs):

- A unique identification number; a nonvolatile, unchangeable and permanently programmed value that can be used to authenticate the FPGA similar to that found in Xilinx FPGAs [38]. This identifier alone, however is not enough for device authentication as it is illustrated in [39].
- Internal Configuration Access Port (ICAP) such as in Xilinx devices [40]. A specially developed circuitry on the FPGA (dubbed *Static Logic*) would configure the FPGA through the ICAP. External reconfiguration and readback ports should be disabled [40].
- Have partial reconfiguration capabilities; i.e. parts of the FPGA can be reconfigured while other parts remain the same.
- Supports the readback of static configuration contents only (Look-Up-Tables, interconnects, and I/Os only), but cannot readback dynamic data such as RAM or

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2864513, IEEE Internet of Things Journal

Flip-Flop contents.

Fig. 2 shows the proposed on-FPGA infrastructure to implement secure IoT data processing in clouds or data-centers. Many of the components of the static or fixed logic blocks (blocks that remain fixed and are not re-configured) already exist in many FPGAs as hard macros (custom circuits pre-fabricated on the FPGA with optimized performance/area/power). The proposed fixed logic includes:

- Key generation circuitry that consists of a PUF (Physically Unclonable Function) and a *b*-generation circuitry to generate random numbers that are used by the modular exponentiation circuit (mexp),
- Enc/Dec block for Encryption/Decryption,
- A hash function such as SHA3 [46],
- Controllers to coordinate the different activities between the static logic blocks and to control the Ethernet communication etc.

PUFs are functions that make use of the manufacturing process variations to generate different (and unique) long random numbers in response to different input stimuli [41]. These responses are not only stable per device, but they are also unique and unpredictable since manufacturing variations cannot be controlled. Hence, PUFs can be used to authenticate devices (such as FPGAs) and provide long random numbers that can be used for key generation. Some of Altera's FPGA are already equipped with SRAM-based PUFs [42]. We believe that in the near future all FPGA devices will be equipped with many standard security primitives (as hardware macros) such as PUFs, Hash, RSA, etc. Even with current FPGAs, the *Static Logic,* can be provided by the FPGA or board manufacturer as pre-configured circuitry on the FPGAs on tamper-proof boards and packages. Boards will have to be shipped with batteries and be powered constantly to maintain the *Static Logic*'s configuration. The entity that does this will be the Trusted Party (**TP**) in the proposed scheme. Users' circuits are placed into specific FPGA regions via partial reconfiguration. Only the input/output of the Encryption/Decryption are available to the users.

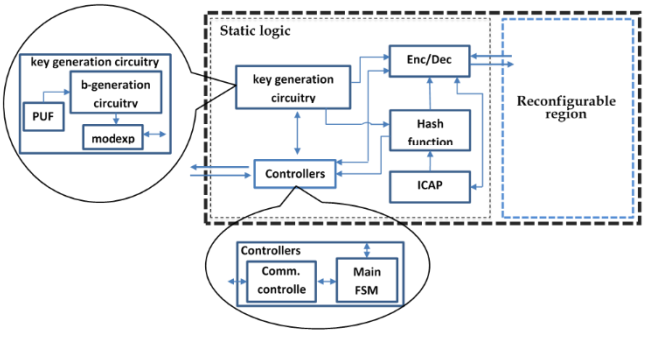

Fig. 2. The proposed structure of on-cloud FPGAs (control signals are not shown).

Notation

RND_1 is the on-FPGA, *m*-bit, PUF-generated random number that is read once by the **TP** (at the FPGA/board manufacturing time) and cannot be read again or altered. The mask number *RND_2,* is also an *m*-bit random number generated by the **TP**. *RND_2* is used to generate an *n*-bit random number *b* from the *FPGA-RND*. *RND* is a secure random number generated by the client. The parameter *config* represents the partial bit stream of a client's design. *Config_RB* is the actual FPGA configuration read back using ICAP. Encryption of a data using a key *k* is denoted as *CT := E(data,k)* and the corresponding decryption as *PT: =D(CT,k)*. An FPGA device F is identified by the unique identifier, denoted as *ID(F)*.

We use the term 'IoT devices' to refer to constrained internet-enabled devices (directly or through gateways). Client application and client are used interchangeably.

The *b*-generation circuitry can be implemented as in Fig. 3. This simple implementation utilizes differential circuitry to protect against differential power attacks during the *b*-generation phase. It consists of two *m*-bit shift (rotate) registers (*RND_1* and *RND_2*) and an *n*-bit register for the produced *b*. The *n*-bits of the *RND_1* with corresponding 1s (i.e. high bits) in *RND_2* are shifted into the *b* register in *m* or less cycles.

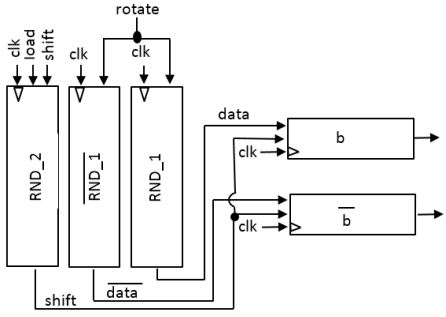

Fig. 3. The *b*-generation circuitry that generates *b* used for the session key. To protect against differential power attacks, differential circuitry is used for the *RND_1* and *b* registers. The *RND_2* register's value is already known and needs no protection.

Involved parties

Fig. 4 shows the parties that are involved in our proposed scheme. The client, who is not necessarily the IoT device or data owner, requests an FPGA to process the IoT devices' data from the Cloud Provider (**CP**) who is providing FPGA-based processing as a service. The FPGA Vendor (**FV**) sells FPGA devices to the **CP** and acts as a Trusted Party (**TP**). **FV** could be the FPGA manufacturer or the board manufacturer. The cloud is used for IoT data storage and processing. The cloud is also used for forwarding the control commands as we consider that there is no direct communication between the client application and the IoT devices (similarly to the cloud-based IoT business models of Microsoft [43] and IBM [4]). Therefore, IoT devices receive control commands from the cloud and send their data to the cloud to be stored and processed. The **TP** facilitates authentication and secret sharing among the different entities involved.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2864513, IEEE Internet of Things Journal

3.2 The Proposed Symmetric Proxy Re-Encryption

Data collected from different IoT devices are usually aggregated and stored in the cloud before clients process it. Different IoT devices use different keys, which means that the on-cloud FPGAs need to use different keys for each data they processes, and similarly, the IoT device needs to send the encrypted data to a specific FPGA. This does not suit the publish/subscribe model used in such scenarios. Therefore, we propose a symmetric proxy re-encryption (PRE) that would securely transform data encrypted by the IoT devices to a data encrypted by the session key established between the IoT device(s) and the on-cloud FPGA. This would make it possible for any IoT device's data to be processed by any authenticated FPGA in the cloud.

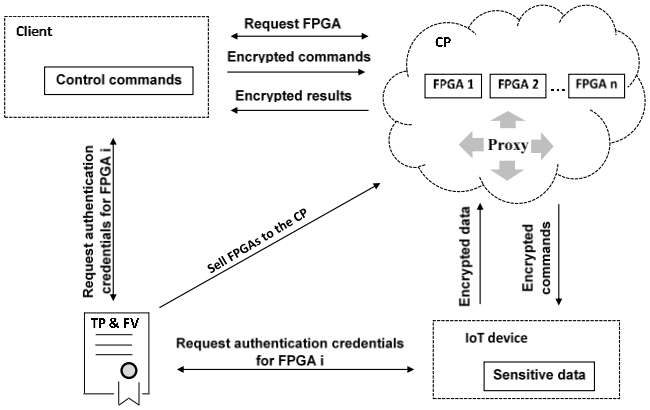

Fig. 4. Overall framework of the proposed scheme.

The symmetric proxy re-encryption scheme is illustrated in Fig. 5. The IoT device authorizes an FPGA to decrypt the data that is stored or going to be stored in the cloud in the format ($data * K \bmod p_i$), where $K = g^r \bmod p$, is the IoT device's private key. The parameters p and p_i are primes, *data* are chunks of data, both *data* and K must be ≥ 1 Kbits long, and *data* cannot be larger than p_i (to be able to recover the data with an inverse modular multiplication). With operands larger than 1 Kb long, there is no known efficient, non-quantum algorithm that solves the integer factorization required to break such a secret [57]. Initially, the IoT device and the FPGA share a session key ($g^{ab} \bmod p$) using Diffie-Hellman (DH) key exchange [44]. The IoT device then sends the re-encryption key (rK) to a proxy residing in the cloud. The rK is computed by multiplying the session key by the multiplicative modular inverse of the IoT device private key ($g^r \bmod p$). The proxy converts the data by modular multipilication by rK and sends the result to the FPGA. The data is now converted to the format $data * g^{ab} \bmod p$ and the FPGA can decrypt the data by inverse modular multiplication by the shared session key ($g^{ab} \bmod p$). Proof of the security of this scheme is straight forward and follows directly the proof of [45].

It is assumed that the proxy and the FPGA are in the same location to make the transformation fast. The only operations that need communication outside the cloud are the FPGA authentication by the IoT device and the sharing of a session key with it. Using a shared key to authenticate the FPGA and allow the FPGA to decrypt the data is only made possible in this case due to the use of symmetric PRE. Had the PRE not being symmetric, public key cryptography would have been required, which requires a certificate authority to certify the public keys for every FPGA in the cloud and limits the scalability of the publish/subscribe system [21].

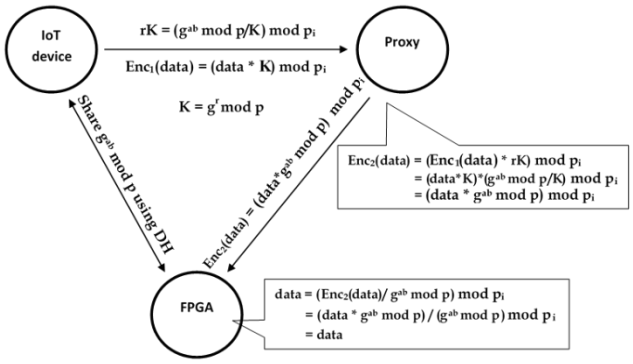

Fig. 5. The proposed symmetric proxy re-encryption.

3.3 Proposed Security Protocol

The proposed protocol for securing the communication between the clients' applications, the IoT devices, and the on-cloud FPGAs is illustrated using the sequence diagram in Fig. 6. The client application is responsible for authenticating the FPGA, securely sharing keys with the FPGA, securely sending configuration bitstream and checking the configuration integrity while the configuration bitstream is running in the FPGA. On the IoT device side, the IoT device also needs to authenticate the FPGA and share a key with it. The IoT data is stored in the cloud using the IoT device's private key ($g^r \bmod p$) and the IoT device gives delegation for the authenticated FPGA to decrypt and process the data:

- A client sends a request for a physical resource (i.e. the FPGA) to the **CP**. An FPGA is assigned to the client. It can receive data from multiple IoT devices. The CP sends the FPGA's identifier (*ID(Fi)*) to the client (step 1 and step 2 in Fig. 6).
- The client forwards the *ID(Fi)* to the **TP** which responds with FPGA authentication credentials; an m-bit random number *RND_2* that has exactly n number of high bits (used for the b-generation as in Fig. 3), hash of the corresponding n-bit number b concatenated with *ID(Fi)*, and the FPGA's session key portion ($g^b \bmod p$) (step 3 and step 4 in Fig. 6). Both g and p are public values with g usually being a small integer such as 2 and p being a prime number satisfying the condition $g^b \geq p$. Similarly, g^a must be $\geq p$.
- The client forwards *RND_2* and its own portion of the session key, $g^a \bmod p$, to the CP and requests the FPGA's authentication credentials. The FPGA uses *RND_2* to generate b using the b-generation circuitry (step 5 in Fig. 6), then uses it to generate its portion of the session key ($g^b \bmod p$), computes *Hash(b+ID(Fi))*, and sends the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2864513, IEEE Internet of Things Journal

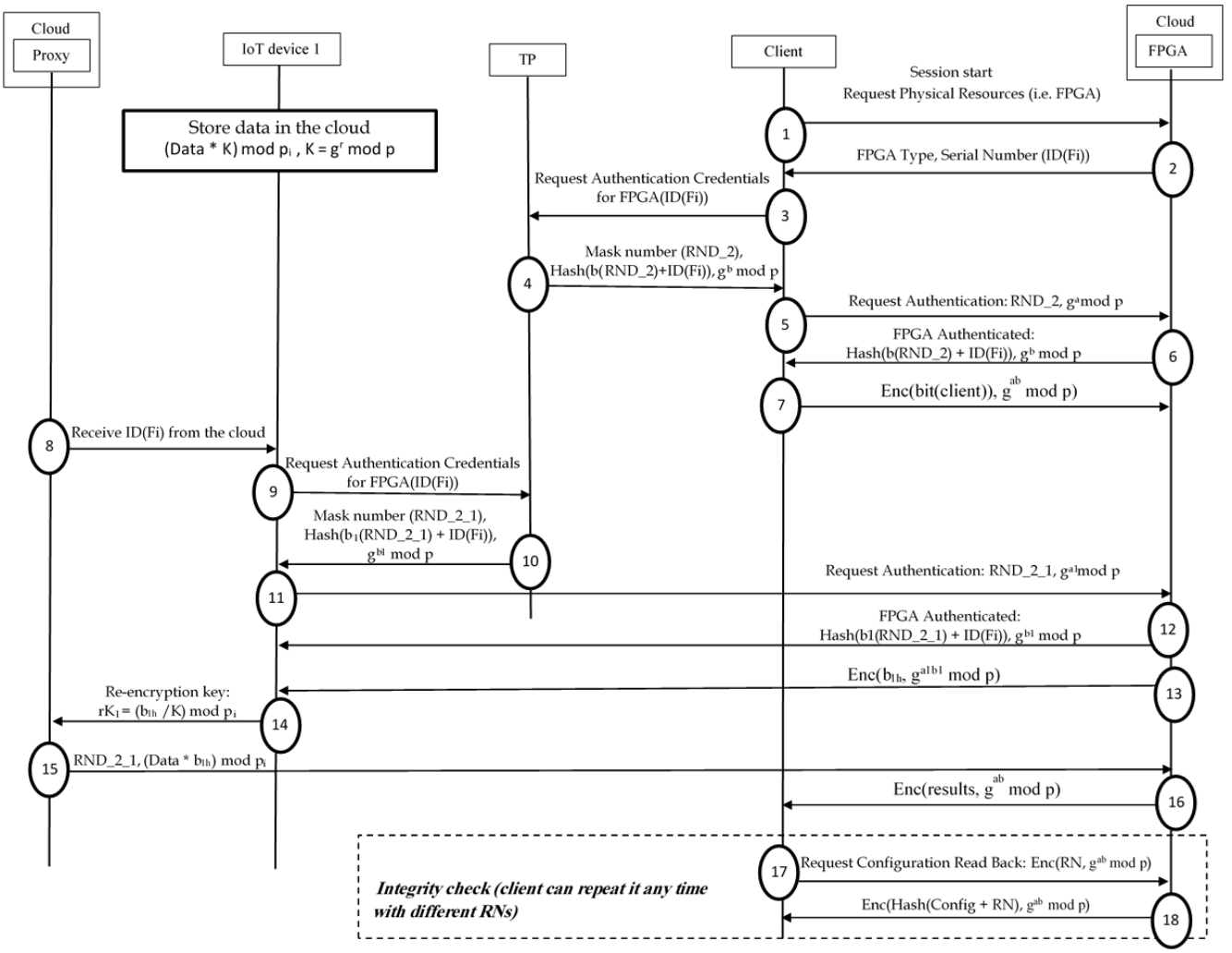

Fig. 6 The protocol sequence diagram. The dotted box indicates the steps of performing integrity check to the configuration running in the FPGA.

result back to the client (step 6 in Fig. 6). The client can now authenticate the FPGA by comparing the values of *Hash (b+ID(Fi))* and (g^b *mod p*) that have been received from the **TP** and **CP**. Both parties now share the symmetric session key (g^{ab} *mod p*), completing the Ephemeral Diffie–Hellman key exchange [46]. At this point, *a* and *b* are destroyed by the client and the FPGA, respectively. In addition, the session key will be destroyed at the end of the session to achieve Perfect Forward Secrecy (PFS).

– The client sends his/her circuit's configuration bitstream *config* encrypted using g^{ab} *mod p*. The static logic on the FPGA will then decrypt it and use it to configure the FPGA through the ICAP (step 7 in Fig. 6).
– The CP broadcasts the ID of the FPGA to the client-specified IoT devices. The IoT device then sends the FPGA ID to the TP and the TP responds with a new random number (*RND_2_1*) along with the hash and the key portion of the Diffie-Hellman key exchange (g^{b1} *mod p*). The IoT device requests FPGA authentication by sending *RND_2_1* and its Diffie-Hellman key exchange portion (g^{a1} *mod p*) to the FPGA. The FPGA responds by providing the hash and the g^{b1} *mod p*. The IoT device can then compare the hashes and keys portions that it had received from both the **TP** and the FPGA (steps 8-12 in Fig. 6). If there is a match, the session will be established. Otherwise, it will be terminated.
– The FPGA hashes the key (b_1), which is generated by the PUF circuitry, and encrypt it using the established session key (g^{a1b1} *mod p*) and sends the encrypted hash (Enc(b_{1h} , g^{a1b1} *mod p*)) to the IoT device (step 13 in Fig. 5) The hash of b_1 is used instead of b_1 to avoid exposing it outside the FPGA and to satisfy the PFS. The IoT device sends the re-encryption key ($rK1 = b_{1h}/g^r$ *mod p) mod* p_i to the on-cloud proxy which in turn transforms the IoT device data as discussed by the scheme in Fig. 4 and sends the re-encrypted data to the FPGA for processing (step 14 and step 15 in Fig. 6). The FPGA decrypts the data, run the application, and sends the result to the client after encrypting it using the session key (g^{ab} *mod p*). The same circuit could process data from different IoT devices, each with its own *rK*.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2864513, IEEE Internet of Things Journal

– To protect against any circuit tampering (e.g. HW Trojans or sniffing circuitry inserted on the FPGA), the client chooses a secure random value *RND*, encrypts it with the g^{ab} *mod p* and sends it to the FPGA requesting configuration readback. The *Static Logic* decrypts *RND*, reads back the FPGA configuration, hashes it with *RND*, encrypts with the session key, and sends it back to the client (step 16 and step 17 in Fig. 6). The client can use this to validate the integrity of the FPGA. This check can be repeated any number of times (with a new *RND* every time to prevent replay attacks), during the operation of the client's circuit on the FPGA.

The steps above are repeated for every session and *RND_2* is never repeated. It should be noted that this scheme also supports 3rd-party provided circuit IPs (i.e. the circuit is provided by an IP vendor). In this case, to protect the circuit IPs, the IP vendor will encrypt the circuit IP(s) using a different Mask and a key obtained through similar steps, and performs the integrity checks.

3.4 Security Analysis

The proposed protocol illustrated in Fig. 5 provides FPGA authentication, configuration integrity check and data confidentiality in the cloud environment. It provides one-way authentication of the FPGA to both, the client's application in the cloud and the IoT device. I.e. the protocol does not authenticate the client's application and the IoT device in the other direction (to the FPGA). The CP usually does that. It also minimizes communication between the client application or the IoT device and the TP.

Table 1 provides a detailed security analysis of the proposed scheme against relevant types of attacks. The attacks are classified as cryptographic, network, and physical attacks. Cryptographic attacks include Known-plaintext attack (KPA), Chosen-plaintext attack (CPA), Ciphertext-only attack (COA), Chosen-ciphertext attack (CCA) and open key attack model, which includes Related-key and Known-key distinguishing attacks. Network attacks include man-in-the-middle (MiM), impersonation and replay attacks [46].

Physical attacks can be invasive, non-invasive, or semi-invasive. The most relevant of such attacks to an FPGA-based secure computing is the side-channel attack; an attack that uses information leaked about the operation being performed. Information leaks out as timing, power, and/or electromagnetic radiation (e.g. differences observed when processing binary 1s and 0s). These attacks are mitigated as shown in Table 1 by using the differential circuitry for generating *b* (Fig. 2) to prevent leaking *RND_1* out and similar techniques for other components such as the RSA [47]. Other non-invasive attacks that are particular to FPGAs include Reverse Engineering and Tampering, Cloning, and Counterfeiting attacks. In Reverse Engineering attacks, an adversary studies the configuration blocks and replaces (or tamper with) the security components by his own malicious components (e.g. Trojans) in order to disclose secrets and sensitive data [48]. Cloning attack occurs when an adversary creates an exact copy of the FPGA configuration. Counterfeiting attack is an extension to the cloning attack and it occurs when identical FPGAs are used. Hence, a configuration made for one device can be used with another. The details of the design do not need to be known by the attacker and the configuration is treated as a black box. Compromising the FPGA would only require inserting a snooping circuitry to disclose FPGA secrets and consequently, the users' data. These attacks are prevented as described in Table 1.

Verification

ProVerif [49] was used for the formal verification of the security of the proposed protocol and to ensure that the protocol does not suffer from any vulnerabilities. ProVerif can verify the secrecy (the attacker cannot obtain the secret), authentication and strong secrecy (the attacker cannot learn the changes made to the secret) of security protocols. Fig. 7 illustrates the formal verification process with ProVerif to test each security poperty.

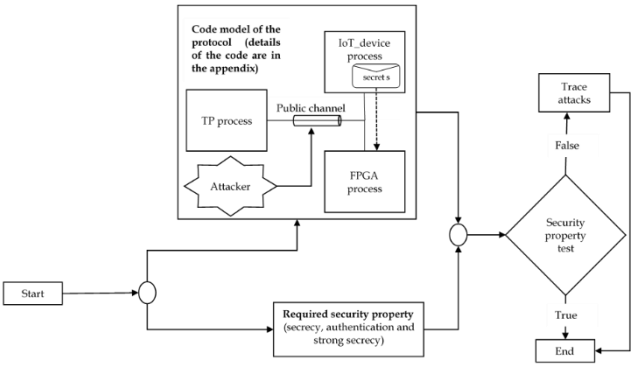

Fig. 7 The formal verification process of the proposed protocol with ProVerif [49].

The following assumptions were made:

– We modeled the interactions between the IoT-device and the FPGA as this also models the interactions between the client and the FPGA.
– The attacker has access to all communication channels except for private channels.
– To verify the match of the hash values received from the **TP** and the FPGA in the IoT device side, these values are sent to the IoT device and the FPGA. The FPGA then send the value received from the TP to the IoT device to ***emulate*** the operations of the *b*-generation and its corresponding hash value.
– The channels between the IoT device/FPGA and the TP are set as private. For the case of the FPGA this is to model the initial process of installing the FPGA secrets by the TP. After the FPGA installation in the cloud, it does not communicate with the TP.
– All communications between entities that go through the cloud are not considered private. Communications between the IoT device and the clients, on one side, and the TP on the other side (i.e. steps 3, 4, 9, and 10 in Fig.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2864513, IEEE Internet of Things Journal

Table 1: Summary of the attacks prevented by the proposed scheme.

Attack category	Attacker	Countermeasures	
Cryptographic attacks	Assumed to be a malicious insider who tries to break the cryptographic oracle and obtains the session key established between the client/IoT device and the FPGA.	Any data outside the FPGA is encrypted. Open key attack model [46] is prevented by producing random uncorrelated numbers using the PUF.	
Network attacks	Assumed to be a malicious insider/outsider attempting to impersonate the FPGA and/or obtain sensitive data.	Impersonation	The TP sends the hash *Hash(b(RND_2) + ID(Fi))* to the client which has to match the hash received from attacker. The attacker can send *Hash(b(RND_2)* + ID(Fi))* and $g^t \bmod p$ which do not equal *Hash(b(RND_2) + ID(Fi)* and $g^b \bmod p$. Replaying the hash to be sent by the *Fi* is also prevented because *RND_2* is never repeated.
		MiM	MiM is unable to re-compute the hash sent by the FPGA while providing the correct $g^b \bmod p$. Exchanges between the client and the TP outside the cloud (messages 3 and 4 in Fig. 6) can be protected by standard protocols such as SSL,TLS etc.
		Replay	Replaying the values to be sent by the FPGA is prevented because *RND_2* is never repeated. Integrity checking replay is prevented using the newly client-generated random number (*RND*).
Physical and FPGA attacks	Assumed to be a malicious insider that has access to the FPGA devices in the datacenter and is trying to obtain the device secrets and the IoT sensitive data.	Invasive	Damage the FPGA and any divulged secrets such as the *RND_1* are useless because it is only unique to that FPGA.
		Non-invasive	All blocks of the *static logic* have constant processing time (cycles). Similarly, Power and Electromagnetic Radiation analysis attacks are mitigated due to the use of differential *RND_1* circuitry and similar techniques for the security components such as the RSA [47]; the power/electromagnetic profiles do not depend on the value of *b* or the shared key.
		Semi-invasive	The required knowledge and equipment are beyond a malicious insider.
		Reverse Engineering and Tampering	*Static Logic* is installed by the TA and cannot be read back. Furthermore, repeated integrity checks would expose any changes made to the FPGA Configuration.
		Cloning and counterfeiting	The PUF produces a unique *RND_1* for every FPGA.

6) take place outside the cloud and are considered private (i.e. secured via SSL/TLS).
– The attacker is active which means that the attacker has full access to all messages and can send or replay messages in the communication channels.

The ProVerif code of the proposed protocol is provided in the appendix. It consists of the following parts:
– Channels involved and adversary model (lines 3-8),
– Encryption/decryption and hash functions models (lines 9-16).
– The TP's, the IoT-device's, and the FPGA's operations (lines 19-22, 23-29, 30-34, respectively).

The results of running the ProVerif code shows that the query is true indicating that the protocol is free from vulnerabilities (all the required security properties are met).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2864513, IEEE Internet of Things Journal

4 Experimental Results

4.1 Static Logic Implementation

To evaluate the practicality and performance of the proposed scheme, a complete proof-of-concept prototype of an FPGA system has been implemented. A Xilinx Virtex-6 LX 550T FPGA [38] prototyping board (with 1 Gbps Ethernet ports) was used for the prototype. For prototyping purposes, the *Static Logic* blocks were implemented using the FPGA's reconfigurable logic blocks. The *Static Logic* is made of the following components:

- A 256-bit SHA3 hashing block based on the Keccak sponge function [50]. The design required major changes to make it routable and to pipeline it (mainly the rounds steps).
- A 1 Kb modular multiplier based on the interleaved modular multiplication algorithm [51].
- A modular exponentiation block (mexp) based on the Square-and-Multiply algorithm [52]. To protect against Differential Power Analysis (DPA) attacks, the implemented modexp performs the multiplication at each step but the result will not be written to the appropriate register except when the corresponding exponent bit is 1 (in case the bit is 0, the result is written to a dummy register). This masks the power dependency on the exponent bits.
- The PUF as 2 Kb differential registers containing a random number, and the *b*-generation circuitry (as shown in Fig. 3).
- An Ethernet controller and a state machine to handle the data flow between the components.

The FPGA's logic and memory utilization of the different *Static Logic* blocks are shown in Table 2 along with their maximum possible frequencies. These results show that even if the *Static Logic* components were to be implemented using the FPGA's configurable resources they would consume relatively very low resources (~8% of LUTs, ~2.5% of flip-flops, ~1.9% of the available block RAMs, and ~3.4% of the available DSP multipliers). Prior works ([52-54]) reported similar results indicating that these types of functions can be implemented very efficiently on FPGAs.

The Static Logic was also synthesized as a custom circuit to estimate its area if it was made as hard macros on the FPGA. The total gate count was 164,223 gates (total RAM and FFs count remain the same as the FPGA implementation). Based on that, and to put this into perspective, the total area of the Static Logic as custom HW macros is estimated to be ~0.06 mm^2 in a state-of-the-art 16/14 nm fabrication technology based on the International Technology Roadmap for Semiconductors (ITRS) [55]. A typical state-of-the-art FPGA would have a die area from few hundred mm^2 to around 2,000 mm^2 [56].

4.2 PRE Implementation

The PRE was also implemented in a second Virtex 6 LX 550T FPGA. It can be installed once in an untrusted manner since the data is not being decrypted in the PRE. Our FPGA-based PRE consists of an Ethernet controller, 1 Kb modular multiplier, and a FSM to receive the *rK* key and accumulate the data into 1 Kb chunks to be multiplied by the *rK*. The components of the FPGA along with their maximum frequency are shown in Table 3.

Table 2. FPGA resource usage of the *Static Logic*.

Static Logic		*LUTs*	*FFs*	*BRAMs*	*DSP*	F_{Max} *(MHz)*
Full System		25,981 (7.56%)	15,599 (2.27%)	12* (0.95%)	29 (3.36%)	234.9
SHA3-256		3,324 (1.13%)	1,117 (0.17%)	0 (0.00%)	3 (0.35%)	273.9
Enc-Dec		8,711 (2.31%)	3,033 (0.41%)	0 (0.00%)	0 (0.00%)	134.7
Key generation	***mexp***	6,816 (1.98%)	3,595 (0.52%)	0 (0.00%)	0 (0.00%)	130.6
	b-generation circuitry	3,340 (0.68%)	4,349 (0.68%)	0 (0.00%)	2 (0.23%)	430.3
Controllers	***Ethernet Controller***	1,302 (0.38%)	1,045 (0.15%)	12 (1.89%)	19 (2.20%)	234.6
	Main FSM	2,488 (0.72%)	2,460 (0.36%)	0 (0.00%)	2 (0.23%)	413.6

* Out of 1,264 available BRAMs (~ 226 Kb out of 22,752 Kb total).

Table 3: FPGA resource usage of the PRE Logic.

PRE Logic	*LUTs*	*FFs*	*BRAMs*	*DSP*	F_{Max} *(MHz)*
Full System	21,068 (4.81%)	8534 (1.24%)	12 (1.89%)	21 (2.43%)	135
ModMult x 10	2,107 (0.61%)	773 (0.11%)	0 (0.00%)	0 (0.00%)	134.7
Ethernet Controller	1,302 (0.38%)	1,045 (0.15%)	12 (1.89%)	19 (2.20%)	234.6
FSM	266 (0.07%)	260 (0.03%)	0 (0.00%)	2 (0.23%)	274.6

The modular multiplication was parallelized into 10 multipliers to match the Ethernet communication capacity. For a 1 Gbps Ethernet link, 8 bits can be processed every cycle. For this reason, BRAMs were used to buffer the data that is received/sent from/to Ethernet link and 10 (serial) multipliers to work in parallel such that 250 bits of the product results are outputted every cycle, ignoring the latency of filling the BRAMs and computing the first product result at the beginning. The synthesis results show that the PRE used less than 5% of the FPGA resource. This means that it can be implemented in low-cost FPGAs.

4.3 Performance Evaluation

Static Logic Speed

As shown in Table 2, the *Static Logic* synthesized on the FPGA was relatively fast. All components used the 100 MHz FPGA board clock since that was more than enough to handle the board's 1 Gbps Ethernet traffic. The SHA3-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2864513, IEEE Internet of Things Journal

256 achieved a throughput of 237MB/s and a latency of 27 cycles to process 64 B of data. Similarly, the 1 Kb- modular multiplier takes 1,024 cycles to process 1 Kb of data and can be enhanced by making multiple copies of it to work in parallel as discussed in subsection 4.2. The Modexp component is used at the beginning of the session and when calculating the modular multiplicative inverse as in Fig. 6. The latency of the modular exponentiation was just under 0.7ms. The latency of our basic *b*-generation circuitry is 2,048 cycles for the 2 Kb *RND_1*. These components can be easily operated at higher frequencies to handle higher bandwidth Ethernet links.

PRE Performance Evaluation

Our proxy re-encryption can be seen as a packet processor; it receives the data and output the re-encrypted result. The performance of our FPGA implementation of the proxy was compared to a software implementation and other published HW implementations [17, 19]. The SW version was implemented in python 2.7 on an Intel Xeon CPU with eight core 3.20 GHz, 23.5 GB of memory, and 2 TB of disk. The input data was kept entirely in the RAM (using arrays). For precise measurement of the time of the SW version, the measurements were repeated 10,000 times and the average of 10 runs was taken producing the 1,000 measurements in Fig. 8 for 1 Mb of data. The minimum time was set to be the actual value, the maximum time to the experimental value, and the percentage of error was calculated and found to be 2.7639% only. The FPGA implementation makes use of the 1 Gb Ethernet link to fill the BRAMs. Based on these setups, Fig. 9 shows the time it takes in seconds for the two implementations. The FPGA implementation is, on average, 5.8 times faster than the SW implementation. Given the speedup obtained, if the Ethernet link is 10 Gb, the speedup will be about 58x.

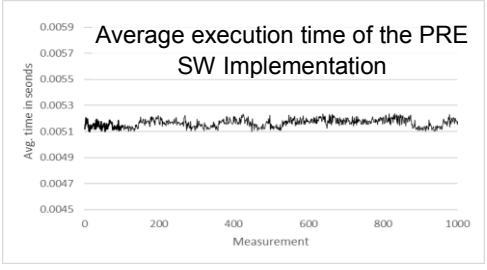

Fig. 8. Execution time of the Python PRE implementation over 1,000 runs.

Table 4 shows a comparison with published FPGA-based secure platforms [17, 19, and 36]. As this table shows, our basic block implementations provide an excellent combination of speed and resource utilization. This is especially true for our PRE implementation, which at half the resource usage, provides over 155x speed improvement over the implementation in [19]. This is due to the use of modular multiplication instead of conventional modular exponentiation-based re-encryption.

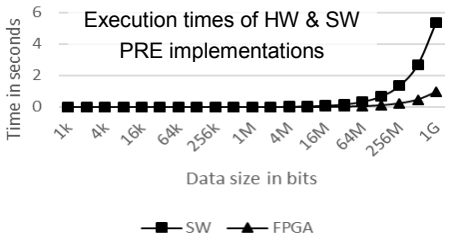

Fig. 9. Execution time comparison of a Python implementation and the PRE FPGA implementation.

Table 4: Performance comparison with published FPGA-based security implementations.

Implementation	*PRE Implementation*		*Hash (SHA3-256)*		*Encryption (AES)*	
	Slices	*Speed* (µs/128-bit re-encryption)	*Slices*	*Speed* (Mbps)	*Slices*	*Speed* (Gbps)
FPGA Trusted Computing [17]	N/A		204	12	456	0.572
Privacy Preserving FPGA [19]	10,854	1,843.71	2,300	540	14,000	40
FPGA-based IoT [36]	N/A		N/A		1,576	12.8
This work*	5,267	11.85	831	237	2,178	34.5

* Including the network controller

Evaluation of Total Latency

The overall end-to-end delay for one transaction as depicted in Figure 6 has been evaluated using the global average internet speed of 6.3 Mb/s [58] and the actual speed of the FPGA static logic and PRE and assuming a 32Kb IoT data size. The detailed times (in milliseconds) of each step along with the exchanged data size(s) and the total time are shown in Table 5. The total transaction time of 230 ms obtained with the conservative estimate of the network bandwidth (inside and outside the cloud) would satisfy typical IoT applications' requirements.

Table 5: End-to-End evaluation of the proposed scheme's delay.

Protocol step #	*Time (ms)*
1- Request (256b)	**3.94E-02**
2- FPGA serial & ID (512b)	**7.88E-02**
3- Request (256b)	**3.94E-02**
4- Mask (2,048b), hash (256b), g^a mod p (2048b)	**0.670**
5- *RND_2 (2,048b), g^a mod p (2,048b)*	**0.630**
6- Hash (256b), g^b mod p (2,048b)	**3.94E-02**
7- Not included	
8- ID (256b)	**3.94E-02**
9- Request (256b)	**0.670**
10- Mask (2,048b), hash (256b), g^{ab} mod p (2,048b)	**0.670**
11- *RND_2_1 (2,048b), g^{ab} mod p (2,048b)*	**0.630**
12- Hash (256b), g^{ab} mod p (2,048b)	**0.354**
13- Encrypted b (2,048b)	**0.315**
14- rK (2,048b)	**0.315**
15- *RND_2_1 (2,048b), encrypted data (32Kb)**	**112**
16- Encrypted results *(32Kb)**	**112**
Total End-to-End delay	**230.1**

***** For both communication and computation, assuming 6.3 Mbs communication speed within the cloud in case the PRE is not in the same geolocation as the FPGA. Computations time is only 32 ms.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2864513, IEEE Internet of Things Journal

5 Conclusions & Discussion

A new FPGA-based scheme for securing IoT data in public clouds that supports emerging business models with several involved parties is proposed. It includes a protocol for establishing a secure session between a client and an on-the-cloud FPGA to process IoT data. Utilizing a symmetric proxy re-encryption, data owners can give temporary access to IoT data in the cloud without divulging the keys used by the IoT devices to encrypt the data. The proposed provides strong protection against various attacks. The use of modular multiplication (with >1Kb long keys) greatly reduces the computational complexity compared to conventional techniques (e.g. RSA) while providing similar security levels. This, and the support for secure dealings between distributed entities (IoT device, Client, re-encryption proxy, and the FPGA), make the proposed scheme well suited for emerging cloud paradigms such as Fog and Edge computing.

A complete proof-of-concept prototype implementation of the scheme showed that it is feasible even with existing FPGAs, simple to implement, efficient in terms of resource utilization and suites the publish/subscribe model used with IoT devices. It also achieves PFS, provides authentication of the on-cloud FPGAs to the clients and IoT devices, and checks the integrity of clients' configurations to prevent any modifications and/or other FPGA related attacks such as reverse engineering and cloning. An FPGA PRE implementation outperformed a SW one by **6x** for a 1GB of transformed text. Our HW implementations provided an excellent combination of speed and HW efficiency compared to other published FPGA-based secure platforms. Even with conservative estimate of the network bandwidth (inside and outside the cloud), the total end-to-end delay of the proposed scheme would satisfy typical IoT applications' requirements. Formal verification with a realistic attacker model was performed on the proposed protocol that proved that it has no vulnerabilities.

Acknowledgment

The authors wish to acknowledge facility support provided by King Fahd University of Petroleum and Minerals.

Appendix: ProVerif code (in pi calculus)

```
(* Diffie-Hellman representation*)
A -> B : e^n0
                B -> A : e^n1
                A and B compute the key as k = (e^n0)^n1 = (e^n1)^n0
A -> B : {s}k *)
free c.(*a channel used to send/receive messages between parties *)
free c1. (*a channel used to send/receive messages between the TP and the IoT_device/FPGA *)
private free s. (*a message to be send securely upon executing the protocol *)
(* active adversary *)
param attacker = active. (*Active means that the attacker can intercept messages send ,receive or modify messages *)
(* Shared key cryptography *)
fun enc/2. (*encryption function with 2 inputs *)
reduc dec(enc(x,y),y) = x. (*the corresponding decryption*)
fun hash/1. (* the hash function with 1 input *)
(* Diffie-Hellman functions *)
fun f/2. (*a function used to represent gab=gba *)
fun g/1. (*the exponent ion function *)
equation f(x,g(y)) = f(y,g(x)). (*the corresponding equation of the function f *)
(* Test whether message s is secret *)
query attacker:s.
(* The TP process *)
let TP = new n00; new b;
(*using channel c1 to share a key with the IoT_device and the FPGA*)
(out(c1,g(n00))  | in(c1,x11); let k = f(n00,x11) in out(c1,
enc(g(n1),k));out(c1,enc(hash(b),k))).
(*The IoT_device process *)
let IoT_device = new n0; new n11;
(*sharing a key with the TP *)
(*receiving gb and hash(b) from the TP *)
        let gb_TP = in (c1,m);
        let hash_b = in (c1,h) in
(*receiving gb and hash(b) from the FPGA *)
        out(c,g(n0)) ; in(c,gb_FPGA)| in(c,hash_FPGA);
(*authenticating the FPGA *)
        if gb_TP=gb_FPGA then
        ( if hash_b=hash_FPGA then
 (*if authentication done, send the message s *)
        let k1 = f(n0,gb_TP) in out(c, enc(s,k1)))
        else (0)).
 (*The FPGA process *)
 let FPGA =  new n01;
 (*receiving gb and hash(b) from the TP and receiving ga from the client *)
                let gb = in (c1,m);
                let hash_b1 = in (c1,h); in
                in(c,x0);
 |                      et k1 = f(gb,x0) in
 (*sending gb and hash(b) to the IoT_device*)
                        out(c,gb);
                        out(c,hash_b1);
                        in (c,m3);
 (*receiving the message s  from the IoT_device *)
                        let s3 = dec(m3,k1) in 0).
process TP | IoT_device | FPGA
```

References

[1] J. Rivera and R. Van der Muelen. (2013, December). Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020. [Online]. Available: http://www.gartner.com/newsroom/id/2636073

[2] R. van Kranenburg and A. Bassi, "IoT Challenges," Commun. Mob. Comput., vol. 1, no. 1, p. 9, 2012.

[3] Amazon. (2016). AWS IoT. [Online]. Available: https://aws.amazon.com/iot/

[4] M. J. Yuan. (2011). IBM Watson. [Online]. Available: https://www.ibm.com/watson/developercloud/services-catalog.html

[5] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative Technology for CPU Based Attestation and Sealing," Proc. 2nd international workshop on hardware and architectural support for security and privacy. Vol. 13. 2013.

[6] S. S. and T. M. J. Götzfried, M. Eckert, "Cache Attacks on Intel SGX." Proc. of the 10th European Workshop on Systems Security, 2017.

[7] C. M. and S. M. M. Schwarz, S. Weiser, D. Gruss, "Malware Guard Extension: Using SGX to Conceal Cache Attacks." arXiv preprint arXiv:1702.08719, 2017.

[8] S. C. and A. S. F. Brasser, U. Muller, A. Dmitrienko, K. Kostiainen, "Software Grand Exposure: SGX Cache Attacks Are Practical," arXiv preprint arXiv:1702.07521, 2017.

[9] M. P. Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," arXiv preprint arXiv:1611.06952, 2016.

[10] G. E. Suh, D. Clarke, B. Gassend, M. Van Dijk, and S. Devadas, "AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing," *Proc. Int. Conf. Supercomput.*, pp. 160–171, 2003.

[11] D. Champagne and R. B. Lee, "Scalable architectural support for trusted software," *High Perform. Comput. Archit. (HPCA), 2010 IEEE 16th Int. Symp.*, pp. 1–12, 2010.

[12] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal Hardware Extensions for Strong Software Isolation," *Proc. 25th USENIX Secur. Symp.*, 2016.

[13] C. W. Fletcher, M. Van Dijk, and S. Devadas, "A secure processor architecture for encrypted computation on untrusted programs," *Scalable Trust. Comput. STC '12. Proc. Seventh ACM Work.*, p. 3, 2012.

[14] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song, "PHANTOM: Practical Oblivious Computation in a Secure Processor," in *CCS'13*, pp. 311–324, 2013.

[15] E. Brickell and J. Li, "Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities," *IEEE Trans. Dependable Secur. Comput.*, vol. 9, no. 3, pp. 345–360, 2012.

[16] Voss, Sven-Hendrik. "Towards unique performance using FPGAs in modern communication, data processing and sensor systems," Signal Processing and Integrated Networks (SPIN), 2016 3rd International Conference on. IEEE, 2016.

[17] K. Eguro and R. Venkatesan, "FPGAs for trusted cloud computing," Proc. 22nd Int. Conf. on Field Programmable Logic and Applications, FPL 2012, pp. 63–70, 2012.

[18] V. Costan and S. Devadas, "Security challenges and opportunities in adaptive and reconfigurable hardware," *2011 IEEE Int. Symp. Hardware-Oriented Secur. Trust*, pp. 1–5, 2011.

[19] L. Xu, W. Shi, and T. Suh, "PFC: Privacy preserving FPGA cloud - A case study of MapReduce," *IEEE International Conference on Cloud Computing, CLOUD*, pp. 280–287, 2014.

[20] P. Swierczynski, M. Fyrbiak, C. Paar, C. Huriaux, and R. Tessier, "Protecting against Cryptographic Trojans in FPGAs," Proc. 23rd IEEE Int. Sym. Field-Programmable Custom Computing Machines (FCCM), pp. 151-154, 2015.

[21] M. A. Tariq, B. Koldehofe, and K. Rothermel, "Securing Broker-Less Publish / Subscribe Systems Using Identity-Based Encryption," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 518–528, 2014.

[22] H. Ning, H. Liu, and L. T. Yang, "Aggregated-proof based hierarchical authentication scheme for the internet of things," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 657–667, 2015.

[23] N. Ye, Y. Zhu, R. C. Wang, R. Malekian, and Q. M. Lin, "An efficient authentication and access control scheme for perception layer of internet of things," *Appl. Math. Inf. Sci.*, vol. 8, no. 4, pp. 1617–1624, 2014.

[24] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, "Towards viable certificate-based authentication for the internet of things," *Proc. 2nd ACM Work. Hot Top. Wirel. Netw. Secur. Priv. - HotWiSec '13*, p. 37, 2013.

[25] Microsoft. (2012). Azure IoT Suite. [Online]. Available: https://www.microsoft.com/en-us/internet-of-things/azure-iot-suite.

[26] F. Li and P. Xiong, "Practical secure communication for integrating wireless sensor networks into the internet of things," *IEEE Sens. J.*, vol. 13, no. 10, pp. 3677–3684, 2013.

[27] I. E. Bagci, S. Raza, T. Chung, U. Roedig, and T. Voigt, "Combined secure storage and communication for the internet of things," *IEEE International Conference on Sensing, Communications and Networking, SECON 2013*, pp. 523–531, 2013.

[28] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight secure CoAP for the internet of things," *IEEE Sens. J.*, vol. 13, no. 10, pp. 3711–3720, 2013.

[29] H. Zhang and T. Zhang, "Short Paper: "A peer to peer security protocol for the internet of things: Secure communication for the sensiblethings platform," *18th International Conference on Intelligence in Next Generation Networks, ICIN 2015*, pp. 154–156, 2015.

[30] J. Qian, H. Xu, and P. Li, "A novel secure architecture for the internet of things," *Proc. International Conference on Intelligent Networking and Collaborative Systems, IEEE INCoS 2016*, pp. 398–401, 2016.

[31] D. Singh, G. Tripathi, and A. Jara, "Secure layers based architecture for internet of things," *IEEE World Forum Internet Things, WF-IoT 2015 - Proc.*, pp. 321–326, 2015.

[32] S. Sicari, A. Rizzardi, D. Miorandi, C. Cappiello, and A. Coen-Porisini, "A secure and quality-aware prototypical architecture for the internet of things," *Inf. Syst.*, vol. 58, pp. 43–55, 2016.

[33] T. Bhattasali, R. Chaki, and N. Chaki, "Secure and trusted cloud of things," *Annual IEEE India Conference, INDICON 2013*, 2013.

[34] B. Guttman and E. Roback, "An Introduction to Computer Security : The NIST Handbook," Natl. Inst. Stand. Technol. Technol, 1995.

[35] J. Dofe, J. Frey, and Q. Yu, "Hardware Security Assurance in Emerging IoT Applications," Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS), pp. 2050-2053, 2016.

[36] A. Johnson, R. Chakraborty, and D. Mukhopadhyay, "A PUF-Enabled Secure Architecture for FPGA-Based IoT Applications," IEEE Trans. on Multi-Scale Computing Systems, Vol. 1, No. 2, pp. 110-122, 2015.

[37] Y. Liu, J. Briones, R. Zhou, and N. Magotra, "Study of Secure Boot with a FPGA-based IoT Device," Proc. IEEE Int. Midwest Symp. on Circuits and Systems (MWSCAS), pp. 1053-1056, 2017.

[38] Xilinx Inc., http://www.xilinx.com, "XAPP1084(v1.3): Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs," *Xilinx, Inc.*, vol. 1084, pp. 1–20, 2013.

[39] S. Goren, O. Ozkurt, A. Yildiz, and H. F. Ugurdag, "FPGA bitstream protection with PUFs, obfuscation, and multi-boot," *6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip, ReCoSoC 2011 - Proceedings*, 2011.

[40] Xilinx Inc., http://www.xilinx.com, "Xilinx Partial Reconfiguration User Guide," vol. UG702, pp. 1–124, 2012.

[41] C. Böhm and M. Hofer, *Physical unclonable functions in theory and practice*, vol. 9781461450. 2013.

[42] Intel, (2010). Stratix V Device Overview. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/hb/stratix-v/stx5_51001.pdf

[43] TechTarget, (2012). Reference Architecture. [Online]. Available: http://internetofthingsagenda.techtarget.com/definition/reference-architecture

[44] R. C. Merkle, "Secure communications over insecure channels," *Communications of the ACM*, vol. 21, no. 4. pp. 294–299, 1978.

[45] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," *Lect. Notes in Computer Science*, vol. 1403, pp. 127–144, 1998.

[46] Mark Stamp, *Information Security Principles and Practice*, 2nd ed., A JOHN WILEY & SONS, INC., PUBLICATION, p. 608, 2011.

[47] P. Maistri, S. Tiran, P. Maurine, I. Koren, and R. Leveugle, "Countermeasures against em analysis for a secured FPGA-based AES implementation," *Int. Conf. Reconfigurable Comput. FPGAs, ReConFig 2013*, 2013.

[48] R. S. Chakraborty, I. Saha, A. Palchaudhuri, and G. K. Naik, "Hardware trojan insertion by direct modification of FPGA configuration bitstream," *IEEE Des. Test*, vol. 30, no. 2, pp. 45–54, 2013.

[49] B. S. Xavier Allamigeon, Vincent Cheval. (2014). ProVerif: Cryptographic protocol verifier in the formal model. [Online]. Available: http://prosecco.gforge.inria.fr/personal/bblanche/proverif

[50] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "On the indifferentiability of the sponge construction," *Lect. Notes in Computer Science*, vol. 4965 LNCS, pp. 181–197, 2008.

[51] D. N. Amanor, C. Paar, J. Pelzl, V. Bunimov, and M. Schimmler,

"Efficient hardware architectures for modular multiplication on FPGAs," *Proc. International Conference on Field Programmable Logic and Applications, FPL*, vol. 2005, pp. 539–542, 2005.

[52] D. Suzuki, "How to maximize the potential of FPGA resources for modular exponentiation," *Advances in Cryptology - Cryptographic Hardware and Embedded Systems - CHES 2007*, pp. 272–288, 2007.

[53] T. Good and M. Benaissa, "AES on FPGA from the Fastest to the Smallest," *Lect. Notes Comput. Sci. Adv. Cryptol. - Cryptogr. Hardw. Embed. Syst. - CHES 2005*, pp. 427–440, 2005.

[54] D. S. Kumar, "Compact Implementation of SHA3-1024 on FPGA," vol. 3, no. 7, pp. 79–86, 2015.

[55] ITRS. (2012). International Technology Roadmap for Semicon ductors. [Online]. Available: http://www.semiconductors.org/clientuploads/Research_Technology/ITRS/2015/0_2015 ITRS 2.0 Executive Report (1).pdf

[56] Xilinx Inc.. 7 Series FPGAs Packaging and Pinout, Product Specification Xilinx. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug475_7Series_Pkg_Pinout.pdf

[57] Thorsten Kleinjung, et. al., "Factorization of a 768-bit RSA modulus," IACR Cryptology ePrint Archive 2010: 006 (2010).

[58] M. Jackson, "Global Average Internet Speeds Hit 6.3Mbps vs 14.9Mbps in UK," 2016. www.ispreview.co.uk/index.php/2016/06/q1-2016-akamai-uk-internet-speeds-reach-15-mbps-vs-6-3-mbps-globally.html.