Secure Computing Enclaves Using FPGAs

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

Abstract—A new scheme for securing users' data and applications in public clouds and data centers using Field Programmable Gate Arrays (FPGAs) has been developed. This scheme incorporates all necessary protocols, hardware, and software components to provide protection against many known potential attacks including internal attacks. It achieves perfect forward secrecy, provides FPGA authentication and integrity checks, and securely establishes a symmetric session key between the user and the FPGA. A complete prototype has been implemented to show the feasibility of the proposed scheme with current FPGAs. Experimental results showed that an FPGA-based compute node can be set up in a cloud in 3.36s; 12.6 times faster than booting a medium-size conventional Virtual Machine (VM) on the same cloud. Based on the average global Internet speed, the time it takes to set up the FPGA-based machine from anywhere in the world was estimated to be 15s. Also, running an experimental secure image processing application on the FPGA took 50 percent less time than running the same application on a conventional state-of-the art processor (without a secure container).

Index Terms—Cryptographic protocols and algorithms, key management, hardware security, secure cloud computing, FPGAs

1 Introduction

Cloud computing has proven to be of immense benefit for individual users and enterprises. Reduction of capital costs, which is one of the essential benefits of cloud computing, makes cloud computing the ultimate choice for enterprises. However, protecting the confidentiality and integrity of the users' data and applications is a major concern that hinders the adoption of cloud computing for applications with sensitive data such as financial data processing and medical data [1], [2]. Current cloud infrastructures are not fully secured against attacks carried from inside the cloud where potential insider attackers have access to users' data and applications on the cloud resources. According to ESG Insider Threats Survey [3], insider attacks, which are carried by staffers in the cloud company, were ranked as the third most dangerous attack of the cloud. In addition, 66 percent of all organizations are very vulnerable to insider attacks methods [3]. Furthermore, 53 percent of respondents of US State of Cybercrime Survey confessed that damages caused by the insider attacks affect their business more than outsider attacks [4]. Researchers outlined how a malicious insider can extract RSA and AES keys in Amazon's cloud by exploiting shared caches [5]. A recent incident highlighted this problem where many companies' Twitter documents were revealed when an administrator's account was hacked by a malicious insider [6].

Hence, securing the clients' data and applications in the cloud would be beneficial for all parties involved. There is a need for a new scheme that provides cloud users with a secure execution environment and a scheme that allows them to; 1) Authenticate this execution environment, 2) Ensure their application's integrity, and 3) protect the integrity and confidentiality of their data. Such a scheme should not place trust on any of the involved entities (cloud infrastructure, cloud operator(s) and ot her clients).

Field Programmable Gate Arrays (FPGAs) are prefabricated integrated circuits (ICs) that contain up to millions of logic circuits and configurable interconnects that can be configured, by the users in the field, to implement any application. Configuration bitstreams are generated by the FPGA vendor's tools and are applied to the FPGA through dedicated configuration ports. FPGAs provide a performance (speed and energy consumption) closer to custom ASICs at a flexibility (configuration and re-configuration) similar to software. Designers can integrate their own designs with pre-designed circuits on the FPGA (called macros) and other circuits provided by a 3rd party (called circuit Intellectual Properties, IPs). FPGAs can be integrated in the cloud to form flexible, scalable, independent and secure compute resources. This would provide the cloud clients with a fast, isolated, and trusted data processing environment for their sensitive data. Compared to conventional software-based systems, FPGAs' attack surface is substantially smaller and better defined. This is because the FPGA configuration does not require the involvement of operating systems, drivers or compilers, making FPGAs suitable to build security solutions under more robust attack models and stronger security guarantees. However, the protection mechanisms provided by FPGA vendors to protect 3rd party circuit IPs are not adequate to secure users' applications data in a public setting such as the cloud. These mechanisms usually utilize symmetric encryption to protect the configuration bitstreams using keys embedded in the FPGAs that are only made available to major clients. In addition, it is impractical to initialize large sets of sensitive data using the FPGA configuration mechanisms due to

• *The authors are with the Computer Engineering Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia. E-mail: {elrabaa, g200464120, marwan}@kfupm.edu.sa.*

Manuscript received 3 Jan. 2018; revised 7 July 2019; accepted 1 Aug. 2019. Date of publication 6 Aug. 2019; date of current version 12 Mar. 2021. (Corresponding author: M. E. S. Elrabaa.) Digital Object Identifier no. 10.1109/TDSC.2019.2933214

1545-5971 © 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.

limited space resources and the slowness of the configuration process. Finally, many successful attacks against such a method have been reported, either using Known-Plaintext Attacks (KPA) [24] or other attacks [25], [26], [27], [28], [29], [30], [31], [32], [33].

In this work, a new scheme is proposed that utilizes FPGAs and standard (proven) security primitives to provide cloud users with a secure environment (i.e., enclaves) to run their applications and data in public clouds and datacenters. It provides confidentiality and integrity for users' data and applications without the pitfalls of previous schemes, namely:

- No secure channel is required between the users and the FPGA they use in the cloud as a computing platform.
- No trust is placed on any entity other than a trusted authority outside the cloud (no parties need to share their secrets with other parties).
- Each session key is used only once and is never reused even for the same user. In addition, none of the parameters used to create the session key are preserved (to ensure perfect forward secrecy).
- No plain text data or configuration bitstreams are ever made accessible to any party (other than their creator).
- Ability to detect most forms of HW tampering with the design.

A complete proof-of-concept prototype has been implemented to evaluate the performance of the proposed scheme and to illustrate how it can be integrated into a cloud/datacenter. In addition, a complete analysis and formal verification of the proposed protocols have been performed to verify its security against various attacks.

Related work is reviewed and summarized in Section 2. In Section 3, an overview of the proposed scheme is introduced including the implemented framework. Detailed security analysis and verification of the proposed protocol are presented in Section 4. Experimental results are presented in Section 5. They include details of all components and their performance, and comparisons to conventional virtual machines and SW implementations. Finally, conclusions and relevant discussion are presented in Section 6.

2 Related Work

Many research and development efforts went into developing trusted execution environments for protecting the users' data and computations in a public cloud or data centers. The most prominent of those is Intel's Software Guard Extensions (SGX) [7]; a set of micro-coded instructions that extend Intel's architecture to provide secure execution enclaves. SGX instructions provide security-sensitive computations with integrity and confidentiality guarantees even against malicious privileged software such as the Operating System (OS), the kernel and hypervisors. SGX allows user-level code to allocate private regions of memory, called enclaves that are protected from other processes (including those running at higher privilege levels). SGX also provides software attestation to assure the user that their code is running in the intended trusted enclave. This attestation covers only the private code and data inside an enclave. Attestation based on a physical Trusted Platform Module (TPM) covers all the software in the platform as in Intel's prior Trusted eXecution Technology (TXT) [8]. SGX does not prevent attacks such as cache-timing attacks, physical attacks and microcode attacks [9]. Many researchers [9], [10], [11], [12] reported successful cache attacks on SGX enclaves.

Prior to SXG, Intel's TXT provided attestation of a platform and the operating system running on that platform to ensure that the OS starts in trusted environment. TXT's aim was to provide an environment for VMs that is isolated from untrusted software and provide the VM control over the platform while it is active. Similar to the conventional TPMs, TXT is also vulnerable to physical DRAM attacks since DRAM encryption is not implemented. In addition, several attacks on the System Management Mode (SMM) were reported (e.g., [13], [14], [15], [16], [17]). Since SMM is the most privileged execution mode, such attacks grant the attacker access to all computer software including accessing the TXT memory.

Several other secure processors have been proposed such as Aegis [18], Bastion [19], Sanctum [20], Ascend [21], and Phantom [22]. 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 [20] 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 does not protect against physical attacks nor does it prevent fault-injection and timing attacks. Ascend [21] and Phantom [22] secure processors make use of oblivious RAMs; RAMs that perform the reads and writes simultaneously to hide the operation being performed and shuffle the RAM contents from time to time to obscure memory access patterns. As such, Ascend and Phantom do not suffer from attacks that probe the RAM address bus and other attacks that attempt to learn memory access patterns of the containers. However, they incur large slowdown.

Unlike SGX, which uses the Enhanced Privacy ID (EPID) [23] 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.

An FPGA-based security approach for cloud computing was proposed in [34]. Several security components are implemented as *Static Logic* (i.e., fixed, non-reconfigurable macros) on the FPGA. They use RSA public and private keys to form a Root of Trust (ROT) inside the FPGA. A Certificate Authority (CA) would certify the public and private keys for every FPGA. On top of the computational complexity due to the use of public cryptography, the private key was not protected using integrity checks; hence, a user could insert a sniffing circuitry to sniff the private key.

Some researchers proposed using TPMs [35] to provide trust for FPGA-based embedded systems [36], [37]. The use of TPMs assumes a secure channel with the user and requires users to take ownership of the TPM and set a chain-of-trust. This is not only impractical in a multi-tenant cloud environment where each user would need to take ownership of the TPM and set their own chain-of-trust, but does not guard

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Comparison Between Schemes for FPGA-Based Secure Computing

Feature	[34]	[36]	[37]	[39]*	[40]	This work
FPGA authentication	✓	×	×	✓	✓	✓
Integrity checking	×	×	×	×	×	✓
Suitability to the cloud+	✓	×	×	× / ✓	✓	✓

*Requires an ASIC to be augmented with an FPGA and may not be compatible with many current FPGAs.
+Schemes that require a secure channel between the user and the FPGA in the cloud are deemed unsuitable for the cloud.

against Man-in-the-Middle (MiM) and replay attacks by a malicious cloud administrator. Analysis of virtualized TPMs (vTPMs) showed that they are less secure than physical TPMs [38], even with a trusted host (i.e., the cloud). vTPMs have the same vulnerabilities to attacks by malicious cloud administrators as TPMs.

In [39] a model for securing users' data that is similar to TPMs was proposed. It was proposed to augment the cloud's servers with two types of chips that are paired cryptographically by the manufacturer; an FPGA as a processing chip and an ASIC as state chip that holds the state between power cycles (using non-volatile memory). The system guarantees integrity and freshness by maintaining a Merkle tree for user's data in the cloud storage. However, no implementation was provided. Furthermore, it is assumed that there is a secure channel between the state chip and the certificate authority. In addition, there were no key management policies for different users nor guarantees of integrity of the configured circuitry.

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 [40]. It uses public/private keys for encrypting/decrypting a symmetric key to be shared between the user and the FPGAs in the cloud. It also uses the FPGA's embedded symmetric keys for bitstream protection, which is not only not available for general users, but was already proven insecure against KPA as stated earlier. In addition, the proposed scheme requires a CA to certify FPGAs 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. This is another major drawback of this scheme. Further, it is not clear why a proxy re-encryption was needed since both the FPGA and the user have their own public/private keys.

Hence, existing protection schemes for users' data in the cloud are of two types:

- Secure Processor Platforms: These suffer from DRAM probing attacks if the DRAM contents are not encrypted, performance loss and cache attacks if the DRAM content is encrypted,
- FPGA-based: These reduce the attack surface since there are no operating systems or other system software involved, however, the proposed techniques either assume secure channels to the FPGA for key exchange which is not practical in cloud computing, or have no key management policy at all and use embedded keys which make them vulnerable to well-established attacks. Also, most of these techniques do not guarantee the integrity of the user's design.

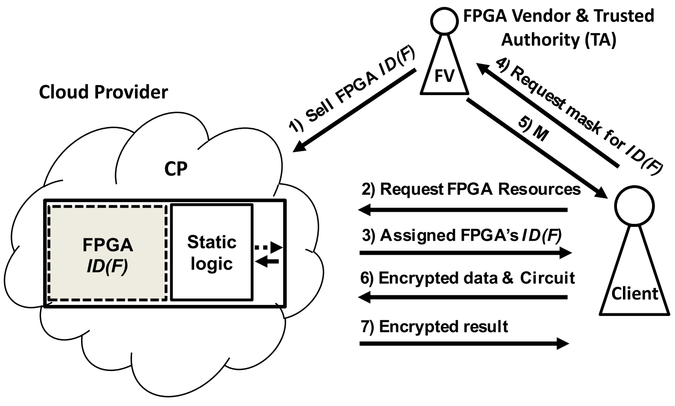

FPGA Vendor & Trusted Authority (TA)
FV
Cloud Provider
CP
FPGA ID(F)
Static logic
1) Sell FPGA ID(F)
4) Request mask for ID(F)
5) M
2) Request FPGA Resources
3) Assigned FPGA's ID(F)
6) Encrypted data & Circuit
7) Encrypted result
Client

Fig. 1. The general framework for the proposed scheme.

Table 1 below shows a comparison between our proposed scheme and previous schemes. As will be demonstrated later, our scheme is the only one that provides authentication for the FPGAs in the cloud, integrity checks for the user's circuitry (to detect tampering), does not require any special arrangement for the FPGA in the cloud nor a secure channel between the user and the FPGA, and is fully compatible with the cloud environment.

3 Overview of the Proposed Scheme

A special framework was developed to provide cloud users with secure FPGA-based computing enclaves without placing any trust on the cloud operators nor requiring secure channels between them and the on-cloud FPGAs. This framework is comprised of two major parts; special on-FPGA infrastructure, and protocols that allow users to securely place their circuits on the cloud-attached FPGAs and use them. The general framework is first introduced below and then the two major components are described afterwards in details.

3.1 The General Framework

Fig. 1 shows the general framework of the proposed scheme including the parties involved and the messages between them. In addition to the client and the Cloud Provider ($\mathbf{CP}$) who is providing network-attached FPGAs to be used for processing client's sensitive data, we consider an FPGA Vendor ($\mathbf{FV}$), and a Trusted Authority ($\mathbf{TA}$) that supplies the FPGA boards to the cloud provider (could be an FPGA or board manufacturer). The TA pre-installs the required logic on these FPGAs (called *Static Logic*) to facilitate the whole scheme. This includes a unique secret per FPGA, PUF_RN, that is only shared with the TA and is used as a root of trust for all subsequent operations on these FPGAs in the cloud. The scheme allows users to deploy their applications (i.e., circuits) and encrypted data to the FPGA, run these applications, and get the encrypted results. Users could develop their own application's circuitry or obtain it from a 3rd-party as a circuit Intellectual Property (IP). It is assumed that such designs have standard communication interfaces (i.e., 1 or 10 Gigabit Ethernet) that work with a standard communication controller (such as in our prototype).

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.

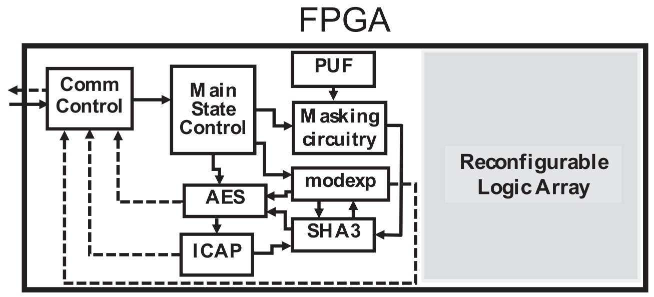


Fig. 2. The proposed structure for the FPGAs in the cloud. Non-shaded area represents the *Static Logic*. Arrows show data flow (control signals are not shown). Dashed arrows represent data going out of the FPGA.

Use Cases: In a typical *use case*, the FPGA is used to process sensitive/confidential data of an application, while SW is co-used for other tasks. For example, to securely access an encrypted database system in the cloud, a one-time trapdoor would be required for each access [46]. FPGAs can be used to generate such trapdoors, process the data and then write the encrypted results back to the database.

3.2 FPGA Infrastructure

Fig. 2 shows the proposed structure of the FPGAs to be used as secure computing enclaves in clouds or datacenters. The *static logic (*i.e., fixed, non-reconfigurable) blocks include a Key generation module, a modular exponentiation circuit, an AES block for Encryption/Decryption, SHA3-256 for Hashing, a controller to coordinate the different activities and load the user's application into the configurable region of the FPGA using partial configuration. Only the inputs/outputs of the Encryption/Decryption and SHA3 circuits are available to the users' circuitry.

Key generation for the proposed scheme is based on an n-bit secret number, *PUF_RN*, that is pre-shared between the TA and the FPGA. It is either generated by an on-FPGA physically unclonable function (PUF) or is stored in a register in the *Static Logic* (as in this work) if the FPGA vendor does not provide PUFs on the FPGA. In case of using an on-FPGA PUF (e.g., such as Altera's SRAM-based PUFs [45]), the TA would read the PUF's output and store it before shipping the FPGA boards to the CP. Whether an actual PUF or a register is used to generate/store PUF_RN, the proposed scheme remains as is.

Much of the aforementioned infrastructure (PUFs, Hash, AES) is being pre-fabricated (i.e., as HW macros) on FPGA devices intended for use in clouds and data centers.

Assumptions. Since the static logic (including the PUF_RN register) has to be pre-configured by the TA before the FPGA boards are shipped to the CP, we assume that these boards have tamper-proof packaging and are shipped with batteries to be powered constantly to maintain the *Static Logic's* configuration. This is similar to Hardware Security Modules (HSMs) that are shipped from the key installation facility to the customers with batteries to maintain a constant power-on state to preserve the installed keys. It is also assumed that the FPGA devices have the following capabilities (exist in current FPGAs):

– A unique public identifier, such as a printed serial number or other means such as a unique device DNA, similar to that found in Xilinx FPGAs [42]. This non-volatile, unchangeable and permanently programmed value is just used to identify the FPGA assigned to a user. As was illustrated in [43], however, it is not suitable for device authentication. In this scheme, we provide another mechanism for FPGA authentication.
– External reconfiguration and read-back ports (such as JTAG) are disabled. This is to ensure that a cloud insider cannot readback or modify the *Static Logic* (including the PUF_RN) nor the user's application.
– Should support partial configuration through an Internal Configuration Access Port (ICAP). This is to enable the *Static Logic* to configure the user's application bitstream in other regions of the FPGA via the ICAP. It should be noted that since the ICAP is instantiated and controlled by the *Static Logic*, it can't be instantiated by any other circuit (only one ICAP instance can be instantiated in any FPGA).
– Support read-back (via ICAP) of static configuration such as Look-Up-Tables, interconnects, and I/Os only, but no dynamic data (e.g., RAM or Flip-Flop contents). This feature is used to check the integrity of the user's circuits and prevent HW tampering with these circuits.
– Have standard high-speed communication ports such as 1 or 10 Gigabit Ethernet (GE) for cloud integration,
– Have tamper proof packaging (i.e., if the IC package is removed, the device is permanently disabled).

Notation. The session mask *M,* is an n-bit random number generated by the TA. *M* contains exactly L number of 1s and is used by the masking circuitry (Fig. 3) to generate an *L*-bit random number *b* from the PUF_RN. *b* is used in the Ephemeral Diffie Hellman (EDH) key generation. *RN* is a secure random number generated by the client (i.e., the user). $Bit(client)$ is the partial bitstream representation of a client's design. *Config* is the actual FPGA configuration read-back through the ICAP. Encryption of a message *msg* using a key *k* is denoted as $MSG := Enc(msg, k)$ and the corresponding decryption as $msg := Dec(MSG, k)$. *E_BIT(client)* is used to denote an encrypted bitstream: $E\_BIT(client) := Enc(Bit(client), k).ID(Fi)$ is a unique identifier of a specific on-cloud FPGA *Fi*.

Fig. 3 shows a differential implementation of the masking circuitry that provides resistance against Differential Power Attacks (DPA) during *b* generation phase. It utilizes two *n*-bit shift (rotate) registers for the PUF_RN and its complement, and two *L*-bit registers for the produced *b* and its complement. The *L*-bits of the PUF_RN ($\overline{PUF\_RN}$) that correspond to the 1s in *M* are shifted into the *b* ($\bar{b}$) registers. Hence, when data is shifted into the *b* registers, both a 1 and a 0 are always shifted, thus masking the power footprint of this operation.

The modular exponentiation block (modexp in Fig. 2) is used for generating keys based on the Square-and-Multiply algorithm. The original algorithm calls for squaring the result at each step, and multiplication only when the corresponding bit of the exponent is 1. To protect against DPA, it was modified to also carry out the multiplication at each step but the result won't 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

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.

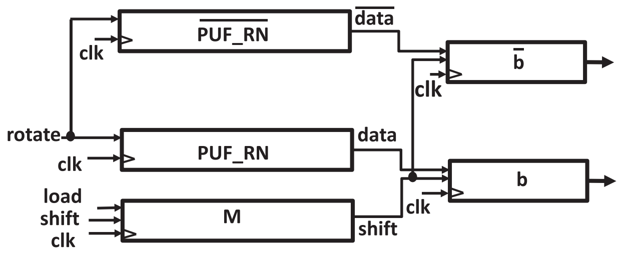


Fig. 3. The masking circuitry that generates *b* which is used for the session key. Differential circuitry is used to protect PUF_RN and *b* against DPAs. The *M* register's value is already known and needs no protection.

masks the power dependency on the exponent bits. Similarly, the AES-256 was implemented using a DPA-resistant design as in [48].

3.3 The Proposed Security Protocol

A special protocol was devised to facilitate the provision of FPGA-based secure enclaves for users' applications and data. This protocol provides the following:

- *FPGA Authentication* which assures users that they are using a specific physical FPGA and not an emulation SW or HW module in the cloud.
- *Confidentiality* of the user's application and data even against a malicious cloud insider.
- *Integrity Protection* of the user's application circuitry (i.e., detect any tampering with their circuitry).

Description of the different steps of the proposed 10-steps protocol, illustrated in Fig. 4, is as follows:

FPGA Authentication (steps 1-6). To ensure the confidentiality of their data and application, users need to authenticate the FPGA and establish a common shared session key with the FPGA so they can encrypt their application's circuitry and data before sending it to the FPGA over a non-secure channel. EDH key exchange is used to establish this shared key [47]. The FPGA authentication is established as follows:

- The client (i.e., the user) requests an FPGA-based enclave from the CP. The CP assigns an FPGA to the client and sends back its unique identifier $ID(Fi)$, steps 1-2.
- The client then forwards the $ID(Fi)$ to the TA and obtains the following FPGA authentication credentials; a session mask *M*, hash of the corresponding *b* (of that specific FPGA) concatenated with $ID(Fi)$, and the FPGA's session key portion $g^b \bmod p$, steps 3-4. 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 $g^b \geq p$. For a 128-bit symmetric security level, $length(p)$ must be 3072-bits for an acceptable level of security till 2030 as per NIST's recommendations [41].
- The client then forwards *M* and his/her own portion of the session key, $g^a \bmod p$, to the FPGA, step 5. Again, g^a must be $\geq p$ and a is a random number generated by the client. The FPGA uses *M* to generate *b*, then its portion of the session key, $g^b \bmod p$ and the $\mathrm{Hash}(b(M) + ID(Fi))$, and sends all these back to the client as authentication credentials, step 6. The user can now authenticate the FPGA by comparing the values of $\mathrm{Hash}(b(M) + ID(Fi))$ and $g^b \; mod \; p$ received from the TA and the FPGA. This prevents MiM and FPGA impersonation attacks.

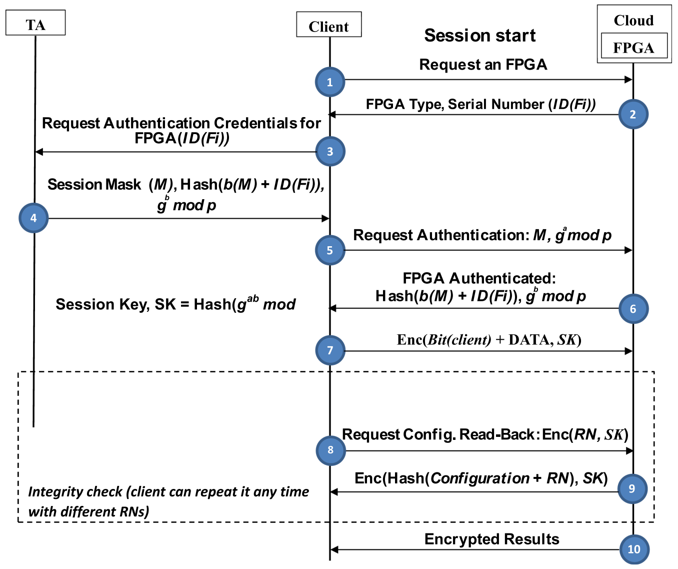

TA
Client
Cloud
FPGA
Session start
1 Request an FPGA
2 FPGA Type, Serial Number (ID(Fi))
3 Request Authentication Credentials for FPGA(ID(Fi))
4 Session Mask (M), Hash(b(M) + ID(Fi)), g^b mod p
5 Request Authentication: M, g^a mod p
6 FPGA Authenticated: Hash(b(M) + ID(Fi)), g^b mod p
Session Key, SK = Hash(g^ab mod
7 Enc(Bit(client) + DATA, SK)
8 Request Config. Read-Back: Enc(RN, SK)
9 Enc(Hash(Configuration + RN), SK)
Integrity check (client can repeat it any time with different RNs)
10 Encrypted Results

Fig. 4. Sequence diagram of the proposed protocol. Steps in the dotted box represent the integrity check of the configuration running in the FPGA.

Session Key (SK) Generation. Both the user and the FPGA then compute the $g^{ab} \; mod \; p$, completing the EDH key exchange. The actual session key is obtained by hashing the EDH key into 256-bit that provides a 128-bit level security. At this point a and b are destroyed by the user and the FPGA, respectively. The session key is destroyed at the end of the session to achieve Perfect Forward Secrecy (PFS). The FPGA secret PUF_RN is further protected by using a hashed version of b. This obscures PUF_RN even if a key is compromised.

FPGA Configuration (step 7). The client encrypts his/her circuit's configuration bitstream $Bit(client)$ using the established session key (*SK* in Fig. 4), and sends it to the FPGA, step 7. The fixed logic on the FPGA will then decrypt it and use it to configure the FPGA through the ICAP. Since bitstream readback is disabled and the ICAP is under the control of the *Static Logic* installed by the TA, the bitstream confidentiality is protected up to this point.

Circuitry Integrity Check (steps 8-9). To protect the integrity of the user's application circuitry and confidentiality against any form of *post-configuration* tampering (e.g., HW Trojans or sniffing circuitry inserted into the FPGA), the user chooses a random value *RN*, encrypts it with the session key and sends it to the FPGA requesting configuration readback, step 8. The *Static Logic* decrypts *RN*, reads back the FPGA configuration, hashes it with RN, encrypts the results using the shared session key, and sends it back to the user, step 9. The user uses this to validate the integrity of his/her circuitry on the FPGA. This check can be repeated at any time (with a new RN every time to prevent replay attacks) during the operation of the client's circuit on the FPGA.

Running The Application (step 10). The user can now send the data encrypted with the shared session key to the FPGA

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.

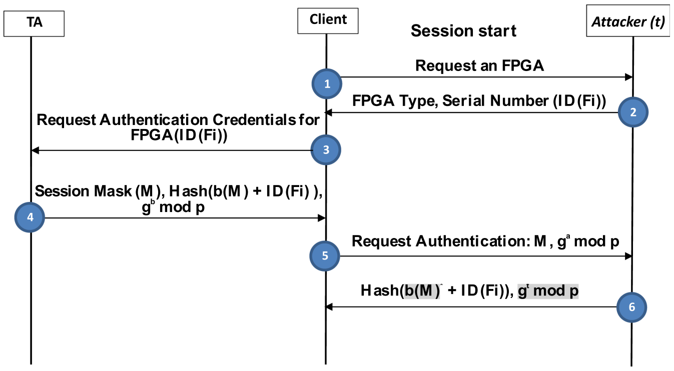

TA
Client
Session start
Attacker (t)
Request an FPGA
1
FPGA Type, Serial Number (ID(Fi))
2
Request Authentication Credentials for FPGA(ID(Fi))
3
Session Mask (M), Hash(b(M) + ID(Fi)), g^b mod p
4
Request Authentication: M, g^a mod p
5
Hash(b(M)* + ID(Fi)), g^t mod p
6

Fig. 5. Illustration of how FPGA impersonation is prevented by the proposed one-way authentication scheme.

where the *Static Logic* decrypts it, applies it to the user's circuit, encrypts the computation results with the shared session key and sends it back to the user, step 10. This ensures the confidentiality of the user's data.

The steps above are repeated for every session where M is never repeated. It should also be noted that this scheme also supports 3rd-party circuit IPs (i.e., the circuit is provided by an IP vendor). To protect the IP, the vendor can encrypt it using a different Mask and key (obtained through similar steps), and perform the integrity checks.

4 Security Analysis and Verification

4.1 Detailed Security Analysis

Relevant attacks on the proposed scheme are classified as *cryptographic attacks*, *network attacks* and *physical attacks*. Below is a detailed security analysis of the proposed scheme's resistance to these attacks.

4.1.1. Cryptographic Attacks

Attacker. assumed to be a malicious insider who tries to break the cryptographic oracle and obtain the session key established between the client and the FPGA.

Possible Attacks. Attacks such as *Known-Plaintext Attack* (KTA) are prevented since no plaintext version of any encrypted data is ever made available to any party other than the one who generated it. Attack models that are based on using related keys or known keys are prevented due to the use of PUFs or random uncorrelated numbers for each FPGA, from which keys are generated. Also, Ephemeral Diffie-Hellman is enabled by the fact that the steps of the protocol are repeated for every session, the session mask (M) is never repeated, and b is never disclosed as a plaintext to the client or to any other party. This yields Perfect Forward Secrecy (PFS); even if an FPGA's internal PUF_RN is leaked, all previously encrypted exchanges remain secure as the session keys cannot be re-created. Values of a and b are deleted by the Client and the FPGA, respectively, once each side establishes the session key. In addition, all session keys are deleted at the end of session.

4.1.2 Network Attacks

Attacker. assumed to be a malicious insider/outsider attempting to impersonate the FPGA and/or obtain sensitive data.

Possible Attacks. The considered attacks include impersonation, replay, and Man-in-the-Middle (MiM) attacks.

– Fig. 5 illustrates how FPGA *impersonation* attacks are prevented by the FPGA authentication steps 5-6 in the proposed protocol. Let an attacker (t) try to impersonate an FPGA with $ID(Fi)$. The TA sends the client the hash $Hash(b(M) + ID(Fi))$ which has to match the hash received from attacker t. In this case, the client receives $Hash(b(M)^* + ID(Fi))$ which does not equal $Hash(b(M) + ID(Fi))$. Hence, Fi impersonation by Attacker (t) is prevented, and *replaying* the hash to be sent by the Fi is also prevented because M is never repeated. Integrity checking is also secured through the use of the symmetric session key, and *replaying* it is prevented through the use of a newly client-generated RN.
– Fig. 6 illustrates how the proposed scheme prevents *MiM* attacks as a result of using the Ephemeral Diffie-Hellman. A MiM cannot re-compute the hash sent by the FPGA while providing the correct $g^b \, mod \, p$. Exchanges between the client and the TA (messages 3 and 4 in Fig. 6) can be further protected by the standard Secure Socket Layer (SSL) or Transport Layer Security (TLS) protocols. This prevents a MiM attacker from injecting the $g^t \, mod \, p$ value in message 4 of Fig. 6 (necessary for a MiM attack to succeed). It also allows the client to authenticate the TA preventing a MiM attacker from impersonating the TA.

4.1.3 Physical Attacks

Attacker. assumed to be a malicious insider that has physical access to the FPGA devices in the datacenter and is trying to obtain the device secrets and the user's data.

Possible Attacks. Physical attacks can be invasive, non-invasive, or semi-invasive.

– *Invasive* attacks damage the FPGA (and the b-generation circuitry) and hence secrets such as the PUF_RN can not be divulged and used to impersonate the FPGA.
– *Semi-invasive* attacks require depackaging the FPGA chip to divulge the PUF_RN without damaging the chip. The knowledge and equipment required to achieve this are only available to few states and major microelectronic manufacturing companies, rendering these attacks irrelevant to normal users.
– *Non-invasive* attacks such as side channel attacks require moderate level of equipment and knowledge to implement. An attacker attempts to use information leaked about the encryption or decryption process (differences observed when processing 1s and 0s of a key) to obtain the key. Information may leak out as timing, power, and/or electromagnetic radiation. Timing attacks against the static logic are not feasible since all its blocks have constant processing time (i.e., cycles) for any key value. Resistance to Power and Electromagnetic Radiation analysis attacks is achieved by using differential PUF_RN and DPA-resistant modexp and AES blocks as was established in [49].

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.

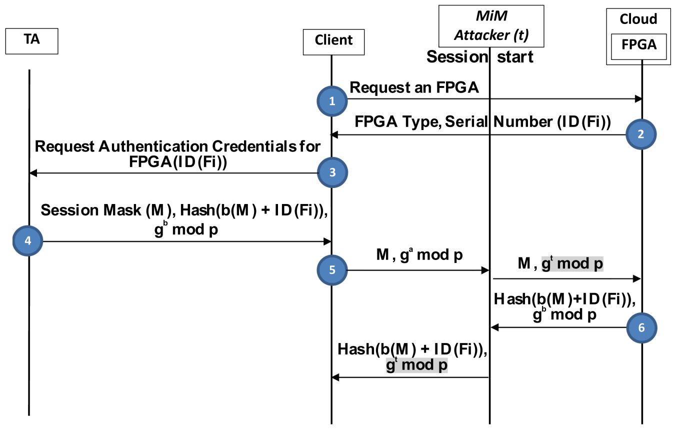

TA
Client
MiM Attacker (t)
Cloud
FPGA
Session start
1
Request an FPGA
2
FPGA Type, Serial Number (ID(Fi))
3
Request Authentication Credentials for FPGA(ID(Fi))
4
Session Mask (M), Hash(b(M) + ID(Fi)), g^b mod p
5
M , g^a mod p
M , g^t mod p
6
Hash(b(M)+ID(Fi)), g^b mod p
Hash(b(M) + ID(Fi)), g^t mod p

Fig. 6. Preventing man-in-the-middle attacks; Client receives $\mathrm{Hash}(\mathrm{b(M)} + \mathrm{ID(Fi)})$ and $\mathrm{g^b}$ mod p from the TA and expects the same values from the FPGA. An attacker forwards $\mathrm{Hash(b(M)} + \mathrm{ID(Fi))}$ and provides gt mod p which will make the attack unsuccessful.

Other relevant non-invasive attacks include Reverse Engineering and Tampering, Cloning, and Counterfeiting.

– In *Reverse Engineering* attacks, an adversary attempts to replace or tamper with some of the configuration blocks to inject HW Trojans in order to disclose FPGA secrets and sensitive data [49]. Such attacks are prevented by the configuration encryption using the established session key. Also, tampering with the user's circuit (after its configuration) by inserting HW Trojans is prevented since the *Static Logic*, installed by the TA, is the only entity that controls the ICAP. Hence, no one can configure the FPGA or readback the configuration except through the *Static Logic* and the implemented security protocol. Even if an attacker managed to install HW Trojans after a user had configured his/her application, the repeated integrity checks in steps 8-9 of the proposed protocol would expose that to the user.
– *Cloning* or *Counterfeiting* attacks involve an adversary creating an exact copy of an FPGA configuration and then use it with another FPGA of the same type and model. This can be easily done in the cloud environment where a malicious insider has access to these FPGAs. The attacker does not need to know the details of the design; the configuration is just regarded as a black box and the FPGA's secrets can be obtained by inserting a snooping circuitry. Our scheme prevents such an attack due to the use of unique PUF_RN for every FPGA. Hence, cloning a configuration to another FPGA will result in incorrect b and the FPGA authentication will fail, as illustrated in Fig. 6. In addition, the user can securely perform periodic integrity check to ensure that the FPGA is not modified while running the application.

4.2 Formal Verification of the Proposed Protocol

A formal verification tool, ProVerif [59], was used to verify the security of the proposed protocol and ensure that it is free from any vulnerabilities. ProVerif uses pi calculus to verify several properties of a protocol; secrecy (the attacker cannot obtain the secret), authentication and strong secrecy (the

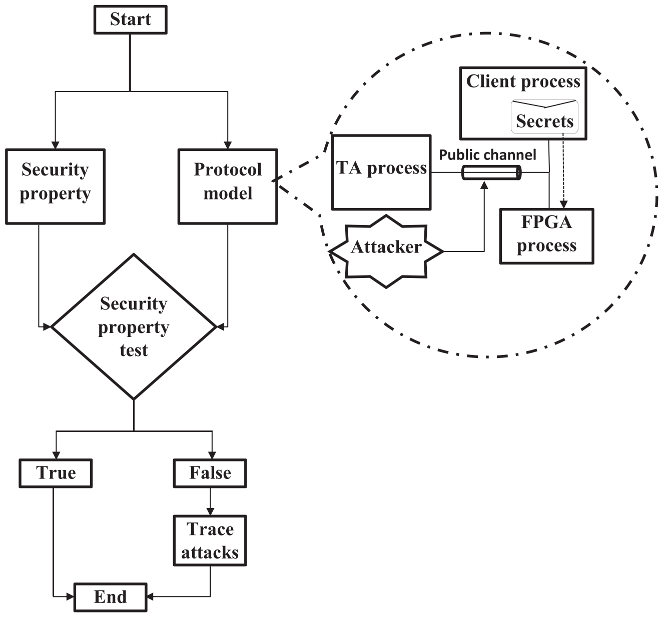


Fig. 7. The verification process of the security properties of the proposed protocol using ProVerif.

attacker cannot learn the changes made to the secret). Fig. 7 illustrates the formal verification process with ProVerif to test each security property and the different processes modeled. All security properties were verified to be true (i.e., the protocol met all the security properties that were queried).

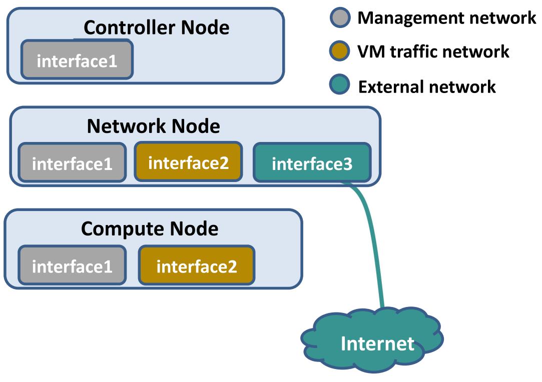
5 Experimental Results

5.1 Testbed Implementation

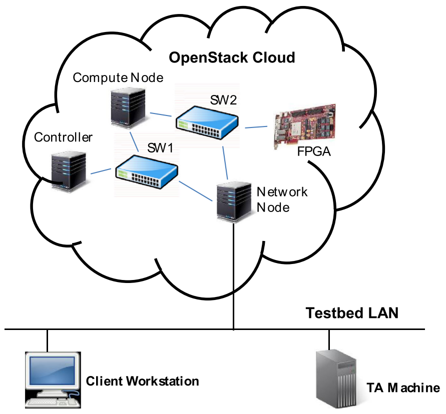
To evaluate the practicality and performance of the proposed scheme, a complete proof-of-concept prototype of a cloud-based FPGA system has been implemented using OpenStack's Juno release [50]. Fig. 8 shows both, the logical cloud infrastructure as per OpenStack guide, and the implemented physical testbed. It consists of three nodes; Compute, Network, and Controller nodes. Two Intel PowerEdge servers each with 16 cores, 32 Gb of RAM and 700 GB hard disk were used for the compute and the network nodes while an i5 PC with 4 Gb of RAM and 500 GB hard disk was used as a controller. Two Fixed Configuration Ethernet switches (SW1 and SW2) with 16 Gbps forwarding bandwidth, 32 Gbps switching bandwidth, and 64 MB/32 MB DRAM/Flash memories provided the interconnection fabric within the cloud. SW1 is used for cloud management and SW2 is used for the clients' communications with their VMs and FPGAs.

The cloud was attached to a LAN via the Network node (using a separate network interface) as shown in Fig. 8. Separate network interfaces were also used in the three nodes for the cloud management traffic and the traffic between virtual machines (VMs). The network and compute nodes contains $br-int$ and $br-ext$ (internal and external bridges, respectively) that are used to share the network interface to enable users to communicate with their virtual machines. No legacy networks were used in this work nor a storage network since storage is not implemented separately. This basic setup can be easily scaled up to thousands of compute nodes and additional networking nodes as needed.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.



(a) The Logical Architecture (nodes and network interfaces in each node) of the OpenStack Cloud implemented in the testbed.



(b) The physical implementation of the cloud testbed.

Fig. 8. The implemented cloud infrastructure.

A Xilinx Virtex-6 LX 550T FPGA prototyping board (with 1/10 Gbps Ethernet ports) was attached to the cloud as an autonomous HW resource using Python scripts that implement the driver-agent model supported by OpenStack as outlined in details in [51]. The FPGA is then scheduled through OpenStack and assigned to the user (similar to a VM) and all the user's traffic are forwarded to it.

A special user interface software that runs on the user's machine was developed. It performs the following; 1) Manages the setup and operation of an FPGA-based computing node on the cloud, 2) Establishes/manages a session with the cloud-based FPGA and handles all data transfers to/from the FPGA from/to the user's machine, 3) Handles all the messages with the TA (using a special port), 4) Generates all the encryption keys for the FPGA's configuration files, user's data and results, and 5) Computes the total time for establishing the secure FPGA-based computing node on the cloud.

The TA's server was emulated by a Python script running on the TA workstation on the LAN. It receives the client's request, performs a random shuffle to produce M, computes $g^b \, mod \, p$ and the hash $Hash(b(M) + ID(Fi))$, and sends them to the client. Since the TA is emulated within the same LAN as the client, Amazon cloud (in different sites) is 'pinged' to estimate the latency of communication with the TA. Using the CloudPing.info service [52], the ping command took an average of 290 ms from the testbed site. This represents the round-trip time taken to obtain the FPGA mask (M), $g^b mod \, p$ and the corresponding hash value from the TA.

TABLE 2
FPGA Resource Usage by the *Static Logic*

Static Logic	*LUTs*	*FFs*	*BRAMs*	*DSP*	F_{Max} *(MHz)*
Full System	78,244 (23.39%)	48,048 (7.02%)	14* (1.11%)	4 (0.46%)	171.5
SHA3-256	3,324	1,117	0	0	285.0
Ethernet	1,302	1,045	12	0	234.6
AES-256	4,068	1,215	2	0	264.0
modexp	63,422	43,038	0	4	117.9
b-generation	6,584	6,268	0	0	484.4
FSM	2,488	2,460	0	0	413.6

* *Out of 1,264 available BRAMs (*$\sim$ 264 Kb *out of 22,752 Kb total).*

The *Static Logic's* components were implemented using the FPGA's reconfigurable logic blocks. Two reconfigurable regions were made on the FPGA; one for the *Static Logic* and another for the users' applications. The *Static Logic* region was then configured (using partial configuration). Another partial configuration file is generated for the test application, which is sent to the FPGA through the cloud. The *Static Logic* would receive the application's partial configuration and configure the application's region using the internal configuration access port (ICAP). The *Static Logic* is made of the following components (their FPGA resource utilization is shown in Table 2):

1. A 256-bit SHA3 hashing block to hash the EDH shared key into 256-bit symmetric key (with 128-bit level security), and provide hashing for FPGA authentication and integrity checks. This circuit was designed and implemented based on the *Keccak sponge* function reported in [53]. The design required major changes to make it routable and to pipeline the rounds steps.
2. A 256-bit AES crypto-engine (still, the security level is 128-bit due to the use of SHA3-256 for symmetric key generation).
3. A 3,072-bit modular exponentiation block (modexp) based on the Square-and-Multiply algorithm.
4. The *b*-generation circuitry (of Fig. 2) containing 3 registers (total of 6,144-bits) for the PUF_RN, M, and b.
5. An Ethernet controller and a state machine to coordinate the operation of the *Static Logic* (including ICAP controller) according to the proposed protocol.

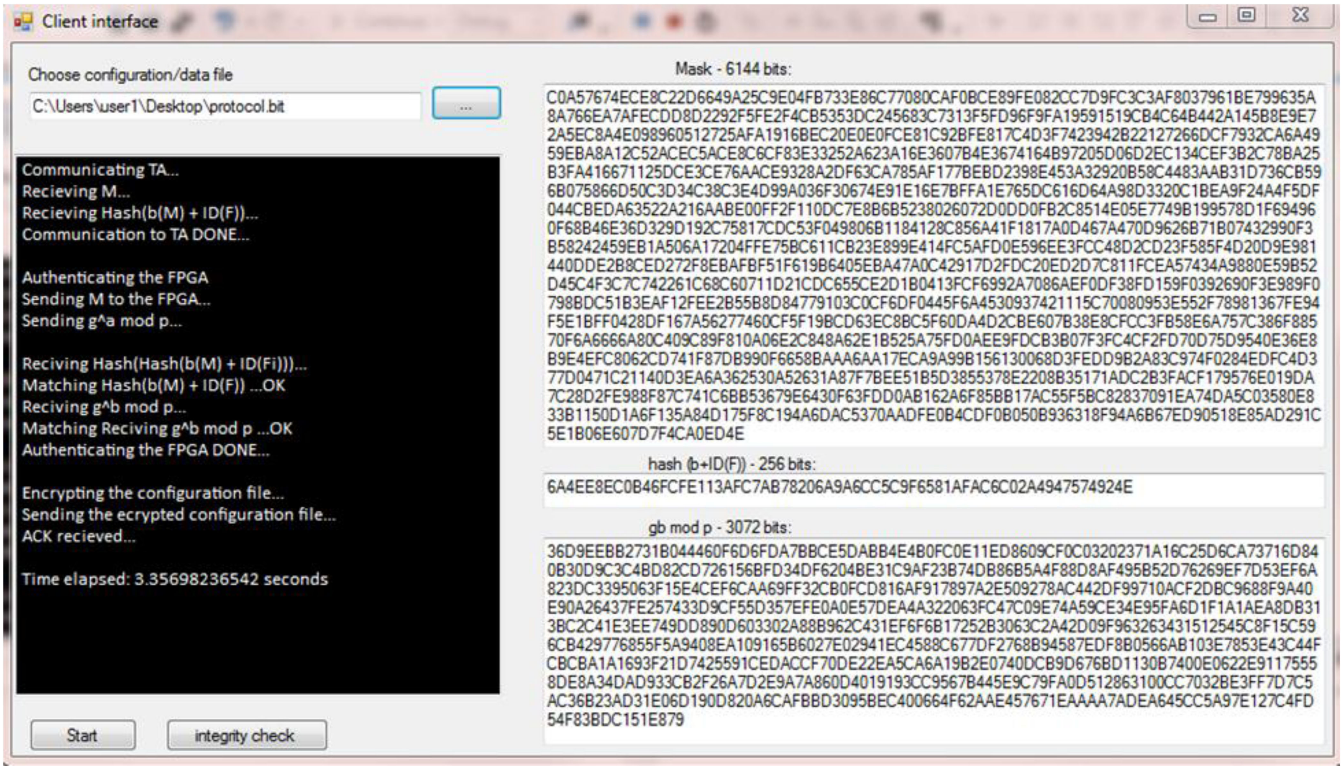
5.2 Custom Implementation of Static Logic

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 $702,058$ gates (total RAM and FFs count remain the same as the FPGA implementation). To put this into perspective, the total area of the *Static Logic* as custom HW macros would be $0.1034 \, \text{mm}^2$ in a state-of-the-art $16/14 \, \text{nm}$ fabrication technology; less than 0.1 percent of the area of a typical FPGA ($\sim 200 - 300 \, \text{mm}^2$).

5.3 Boot Time Measurements

To evaluate the practicality of the proposed scheme, the setup time of an FPGA-based computing node on the cloud was measured and compared to that of conventional SW-based virtual-machines. Boot times do not include the time

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.


Client interface
Choose configuration/data file
C:\Users\user1\Desktop\protocol.bit
...
Start
integrity check

```
Communicating TA...
Recieving M...
Recieving Hash(b(M) + ID(F))...
Communication to TA DONE...

Authenticating the FPGA
Sending M to the FPGA...
Sending g^a mod p...

Reciving Hash(Hash(b(M) + ID(Fi)))...
Matching Hash(b(M) + ID(F)) ...OK
Reciving g^b mod p...
Matching Reciving g^b mod p ...OK
Authenticating the FPGA DONE...

Encrypting the configuration file...
Sending the ecrypted configuration file...
ACK recieved...

Time elapsed: 3.35698236542 seconds
```

Mask - 6144 bits:

```
C0A57674ECE8C22D6649A25C9E04FB733E86C77080CAF0BCE89FE082CC7D9FC3C3AF8037961BE799635A
8A766EA7AFECDD8D2292F5FE2F4CB5353DC245683C7313F5FD96F9FA19591519CB4C64B442A145B8E9E7
2A5EC8A4E098960512725AFA1916BEC20E0E0FCE81C92BFE817C4D3F7423942B22127266DCF7932CA6A49
59EBA8A12C52ACEC5ACE8C6CF83E33252A623A16E3607B4E3674164B97205D06D2EC134CEF3B2C78BA25
B3FA416671125DCE3CE76AACE9328A2DF63CA785AF177BEBD2398E453A32920B58C4483AAB31D736CB59
6B075866D50C3D34C38C3E4D99A036F30674E91E16E7BFFA1E765DC616D64A98D3320C1BEA9F24A4F5DF
044CBEDA63522A216AABE00FF2F110DC7E8B6B5238026072D0DD0FB2C8514E05E7749B199578D1F69496
0F68B46E36D329D192C75817CDC53F049806B1184128C856A41F1817A0D467A470D9626B71B07432990F3
B58242459EB1A506A17204FFE75BC611CB23E899E414FC5AFD0E596EE3FCC48D2CD23F585F4D20D9E981
440DDE2B8CED272F8EBAFBF51F619B6405EBA47A0C42917D2FDC20ED2D7C811FCEA57434A9880E59B52
D45C4F3C7C742261C68C60711D21CDC655CE2D1B0413FCF6992A7086AEF0DF38FD159F0392690F3E989F0
798BDC51B3EAF12FEE2B55B8D84779103C0CF6DF0445F6A4530937421115C70080953E552F78981367FE94
F5E1BFF0428DF167A56277460CF5F19BCD63EC8BC5F60DA4D2CBE607B38E8CFCC3FB58E6A757C386F885
70F6A6666A80C409C89F810A06E2C848A62E1B525A75FD0AEE9FDCB3B07F3FC4CF2FD70D75D9540E36E8
B9E4EFC8062CD741F87DB990F6658BAAA6AA17ECA9A99B156130068D3FEDD9B2A83C974F0284EDFC4D3
77D0471C21140D3EA6A362530A52631A87F7BEE51B5D3855378E2208B35171ADC2B3FACF179576E019DA
7C28D2FE988F87C741C6BB53679E6430F63FDD0AB162A6F85BB17AC55F5BC82837091EA74DA5C03580E8
33B1150D1A6F135A84D175F8C194A6DAC5370AADFE0B4CDF0B050B936318F94A6B67ED90518E85AD291C
5E1B06E607D7F4CA0ED4E
```

hash (b+ID(F)) - 256 bits:

```
6A4EE8EC0B46FCFE113AFC7AB78206A9A6CC5C9F6581AFAC6C02A4947574924E
```

gb mod p - 3072 bits:

```
36D9EEBB2731B044460F6D6FDA7BBCE5DABB4E4B0FC0E11ED8609CF0C03202371A16C25D6CA73716D84
0B30D9C3C4BD82CD726156BFD34DF6204BE31C9AF23B74DB86B5A4F88D8AF495B52D76269EF7D53EF6A
823DC3395063F15E4CEF6CAA69FF32CB0FCD816AF917897A2E509278AC442DF99710ACF2DBC9688F9A40
E90A26437FE257433D9CF55D357EFE0A0E57DEA4A322063FC47C09E74A59CE34E95FA6D1F1A1AEA8DB31
3BC2C41E3EE749DD890D603302A88B962C431EF6F6B17252B3063C2A42D09F963263431512545C8F15C59
6CB429776855F5A9408EA109165B6027E02941EC4588C677DF2768B94587EDF8B0566AB103E7853E43C44F
CBCBA1A1693F21D7425591CEDACCF70DE22EA5CA6A19B2E0740DCB9D676BD1130B7400E0622E9117555
8DE8A34DAD933CB2F26A7D2E9A7A860D4019193CC9567B445E9C79FA0D512863100CC7032BE3FF7D7C5
AC36B23AD31E06D190D820A6CAFBBD3095BEC400664F62AAE457671EAAAA7ADEA645CC5A97E127C4FD
54F83BDC151E879
```

Fig. 9. The user interface software showing the message exchanged and the time it takes to establish a secure session on the in-the-cloud FPGA.

for generating the FPGA's configuration file nor the time to generate the VM's image. Fig. 9 shows a snapshot of the user interface with the sequence of events to establish a secure session with the user's application on the FPGA which took ~ 3.36 seconds. To put this number in perspective, Table 3 shows the boot times for various virtual machine instances with CirrOS images on the OpenStack-based cloud implementation testbed. The boot requests were issued from a user machine on the same LAN as the cloud testbed. CirrOS is a 12-MB OpenStack small Linux based operating system image that is used for testing images in OpenStack clouds. As Table 3 shows, it took 41 seconds to boot a medium size VM. Using the same cloud configuration, the secure FPGA-based computing node (with 10 MB configuration file) was booted in about 3.36 seconds as shown in Fig. 9. This is about 12 times faster than the medium size VM. Moreover, considering a user with an internet connection of an average global speed of the internet (i.e., 6.3Mb/s [57]) he/she can establish a session from his/her location to the on-cloud FPGA in about 15 seconds.

TABLE 3
Boot Times (in Seconds) for Various Virtual Machine Configurations on the Implemented Cloud Testbed Compared to that of the Proposed Scheme on the Same Cloud

VM Size	Virtual cores	RAM	Disk	Ephemeral Storage	Boot time (sec)
xlarge	8	16 GB	10 GB	160 GB	52
large	4	8 GB	10 GB	80 GB	46
medium	2	4 GB	10 GB	40 GB	41
small	1	2 GB	10 GB	20 GB	34
tiny	1	512 MB	1 GB	0 GB	30
Time to boot-up a secure application on the FPGA using the proposed scheme in this work					3.36

5.4 Performance Evaluation

5.4.1 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 SHA3-256 achieved a throughput of 285 MB/s and a latency of 27 cycles to process 64B of data. Hence, the extra hashing step for the variant protocol will only add 27 cycles to the authentication time. Similarly, the AES-256 module had a throughput of 235 MB/s and 40 cycles latency for 16B of data. In fact, it only takes 17 ms to encrypt/decrypt a 4 MB file. Modexp component is rarely used and it is used only at the beginning of the session. It takes $\sim 6.3\,\mathrm{ms}$ to perform modular exponentiation for 3,072-bit base, exponent and modulus. The latency of the masking circuitry is 6,144 cycles for the 6,144-bit PUF_RN. These components can be operated at higher frequencies to handle a faster network connection. Table 4 shows that the speeds of the implemented *Static Logic* blocks are similar to the best FPGA implementations reported in previous works [54], [55], [56].

5.4.2 Performance Evaluation of a Secure Application

To evaluate the performance of the proposed scheme, a simple secure image processing application that performs edge-detection (bitmap format) was used. It reads encrypted images, decrypts them, performs edge detection, encrypts the result, and then writes it back. This represents an alternative scenario to the typical case of using trapdoors to process

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.

TABLE 4
Performance Comparisons of the Static Logic Blocks With Other FPGA Implementations

Block	*This work*	*[54]*	*[55]*	*[56]*
SHA-256 Throughput	*285 MB/s*	–	–	*120 MB/s*
AES-256 Throughput	*1.85 Mbps/slice*	–	*1.44 Mbps/slice*	–
ModExp Latency	*6.3 ms (3,072 bit)*	*12.6 ms (2,048 bit)*	–	–

TABLE 5.
Performance Comparison Between a 'Bare Metal' SW Implementation and a Cloud-Attached FPGA Implementation of a Secure Image Processing Application

SW	*FPGA_1GE*	*FPGA_10GE*
Read, Decrypt, Compute, then Encrypt & Write: Total of 7.82 s	5.03 s (1.56X)	3.52 s (2.22X)

sensitive encrypted data in the cloud as was discussed before. It is also ideal for FPGA implementation. This experiment is meant to show that the FPGA implementation utilizing our infrastructure still maintains the performance superiority (as with standalone FPGA designs) over SW implementations. A standard C-code SW implementation of the same application was run directly (no containerization or virtualization) on a Xeon machine with 18 2.7 GHz Cores, 20 MB L3 Cache, and 32 GB of memory. This machine has native AES instructions, and Intel's AES SDK was used for the image encryption/decryption portions of the application. The experiments were carried over 1 GB of streamed images. Furthermore, for the SW version, the images (and encryption keys) were read/written back from/to the memory to achieve 'bare metal' performance. For the FPGA implementation, images were streamed to the FPGA from a workstation via a LAN switch. Table 5 reports the total processing time of 1 GB of images. For the SW implementation, it shows a breakdown of the different steps of the application (read & decrypt), compute the edges, and encrypt & write Back. The results for the FPGA are reported for two Ethernet speeds; 1 Gps (FPGA_1GE) and 10 Gps (FPGA_10GE). For both cases, the FPGA circuitry operated at 100 MHz as the speed is communication-bounded. These results show that for such an application, secure processing on the FPGA achieved a speed up of 1.56X (for 1 Gps connection) over a SW implementation that utilized Intel AES instructions. The speed up would be more significant if the SW implementation was run in a container or a VM. Furthermore, using a secure processor (e.g., with Intel SGX technology), would increase the SW processing time even further. Unfortunately, we do not have access to such a processor, but many researchers reported a slowdown of up to 5.5X with SGX, especially if the enclave's data/code size exceeds that of L3 cache (e.g., [58]). If the enclave's data/code size exceeds the 128 MB maximum memory allocated for the processor (as private memory), SGX would actually be 200X slower than a conventional SW implementation.

6 Conclusions and Discussion

A new scheme for providing safe enclaves on FPGAs for users' data and applications in public datacenters and clouds has been developed. The proposed protocol provides FPGA authentication, confidentiality and integrity protection of the users' applications, and confidentiality of their data. Security analysis showed that it provides strong protection against relevant attacks and achieves perfect forward secrecy. A complete proof-of-concept prototype implementation showed that the proposed scheme is feasible even with existing FPGAs, simple to implement, efficient in terms of resource utilization and takes less time to boot as compared to conventional software-based virtual machines. Even at a moderate FPGA frequency of 100 MHz, a secure application runs faster on the FPGA than a SW version running on 'bare metal' (i.e., with no containerization, virtualization, or a secure enclave such as Intel's SGX). However, unlike SW-based enclaves, the new scheme requires re-designing an application to make it run on an FPGA as a HW circuit. Advancements in high-level synthesis and the abundance of circuit IPs from various vendors, can provide significant help in this regard.

Acknowledgments

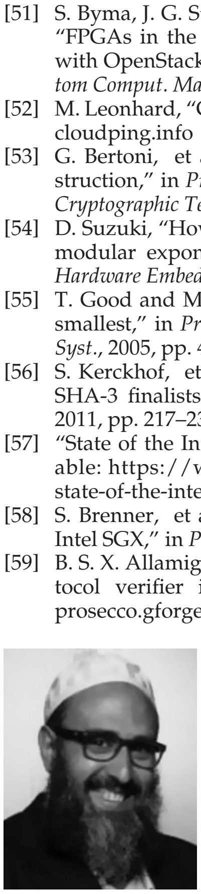
The authors wish to acknowledge facility support provided by King Fahd University of Petroleum and Minerals.

References

[1] J. Casey, "Top 5 cloud computing challenges," 2016. [Online]. Available. http://trilogytechnologies.com/top-five-challenges-of-cloud-computing
[2] "Impact of cloud computing on healthcare," 2012, Cloud standards customer council [Online]. Available. http://www.cloud-council.org/deliverables/CSCC-Impact-of-Cloud-Computing-on-Healthcare.pdf
[3] J. OltsikI, "The 2013 vormetric/ESG insider threat report," 2013, [Online]. Available. https://www.thalesesecurity.com/resources/research-reports-and-white-papers/2013-vormetric-insider-threat-report
[4] "How bad is the insider threat?" Carnegie Mellon University US State of Cybercrime Survey, 2013. [Online]. Available. https://resources.sei.cmu.edu/asset_files/Presentation/2013_017_101_58739.pdf
[5] T. Ristenpart, et al., "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proc. 16th ACM Conf. Comput. Commun. Security*, 2009, pp. 199–212.
[6] M. Arrington, "In our box: Hundreds of confidential twitter documents," Techcrunch.com, 2009, [Online]. Available. http://techcrunch.com/2009/07/14/in-our-inbox-hundreds-of-confidential-twitter-documents/
[7] I. Anati, et al., "Innovative technology for CPU based attestation and sealing," in *Proc. 2nd Int. Workshop Hardware Architectural Support Security Privacy*, vol. 13, 2013. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.405.8266&rep=rep1&type=pdf
[8] D. Grawrock, "Dynamics of a trusted platform: A building block approach," 1st ed., Intel Press, 2009.
[9] J. Götzfried, et al., "Cache attacks on Intel SGX," in *Proc. 10th Eur. Workshop Syst. Security*, 2017, Art. no. 2.
[10] M. Schwarz, et al., "Malware guard extension: Using SGX to conceal cache attacks," in *Proc. 14th Conf. Detection Intrusions Malware Vulnerability Assessment*, 2017, pp 3–24.
[11] F. Brasser, et al., "Software grand exposure: SGX cache attacks are practical," 2017. [Online]. Available: https://arxiv.org/abs/1702.07521
[12] S. Lee, et al., "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," 2016. [Online]. Available: https://arxiv.org/abs/1611.06952

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.

[13] L. Duflot, et al., "Using cpu system management mode to circumvent operating system security functions," in *Proc. 7th CanSecWest Conf.*, 2006. [Online]. Available: https://pdfs.semanticscholar.org/62be/ba49b7a9eb50c0a860547cceb2863e994aa2.pdf
[14] S. Embleton, et al., "Smm rootkit: A new breed of os independent malware," in *Proc. 4th Int. Conf. Security Privacy Commun. Netw.*, 2008, pp. 1–12.
[15] J. Rutkowska and R. Wojtczuk, "Preventing and detecting xen hypervisor subversions," Blackhat Briefings USA, 2008. [Online]. Available: http://www.invisiblethingslab.com/resources/bh08/part2-full.pdf
[16] F. Wecherowski, "A real smm rootkit: Reversing and hooking bios smi handlers," *Phrack Mag.*, vol. 13, no. 66, 2009. [Online]. Available: http://phrack.org/issues/66/11.html
[17] R. Wojtczuk and J. Rutkowska, "Attacking smm memory via Intel cpu cache poisoning," *Invisible Things Lab*, 2009. [Online]. Available: https://blog.invisiblethings.org/2009/03/19/attacking-smm-memory-via-intel-cpu.html
[18] G. E. Suh, et al., "Aegis: Architecture for tamper-evident and tamper-resistant processing," in *Proc. 17th Int. Conf. Supercomputing*, 2003, pp. 160–171.
[19] D. Champagne and R. B. Lee, "Scalable architectural support for trusted software," in *Proc. 16th IEEE Int. Symp. High Perform. Comput. Archit.*, 2010, pp. 1–12.
[20] V. Costan, et al., "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 857–874.
[21] C. W. Fletcher, et al., "A secure processor architecture for encrypted computation on untrusted programs," in *Proc. 7th ACM Workshop Scalable Trusted Comput.*, 2012, pp. 3–8.
[22] M. Maas, et al., "Phantom: Practical oblivious computation in a secure processor," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2013, pp. 311–324.
[23] E. Brickell and J. Li, "Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities," *IEEE Trans. Depend. Secure Comput.*, vol. 9, no. 3, pp. 345–360, May/Jun. 2012.
[24] P. Swierczynski, et al., "Protecting against cryptographic trojans in FPGAs," in *Proc. 23rd IEEE Int. Symp. Field-Programmable Custom Comput. Mach.*, 2015, pp. 151–154.
[25] A. Moradi, et al., "Black-box side-channel attacks highlight the importance of countermeasures an analysis of the Xilinx Virtex-4 and Virtex-5 bitstream encryption mechanism," in *Proc. Cryptographers' Track RSA Conf.*, 2012, vol. 7178, pp. 1–18.
[26] M. Masoomi, et al., "A practical differential power analysis attack against an FPGA implementation of AES cryptosystem," in *Proc. Int. Conf. Inf. Soc.*, 2010, pp. 308–312.
[27] F. Khelil, et al., "Fault analysis attack on an FPGA AES implementation," in *Proc. New Technol. Mobility Security*, 2008, pp. 1–5.
[28] V. Carlier, et al., "Generalizing square attack using side-channels of an AES implementation on an FPGA," in *Proc. Int. Conf. Field. Programable Logic Appl.*, 2005, vol. 2005, pp. 433–437.
[29] E. De Mulder, et al., "Electromagnetic analysis attack on an FPGA Implementation of an elliptic curve cryptosystem," in *Proc. Int. Conf. Comput. Tool*, pp. 1879–1882, vol. 2, 2005.
[30] J. Zhao, et al., "Differential power analysis and differential fault attack resistant AES algorithm and its VLSI implementation," in *Proc. 9th Int. Conf. Solid-State Integr.-Circuit Technol.*, 2008, pp. 2220–2223.
[31] J. Wu, et al., "FPGA-based measurement and evaluation of power analysis attack resistant asynchronous S-Box," in *Proc. IEEE Int. Instrum. Meas. Technol. Conf.*, 2011, pp. 1–6.
[32] S. A. Kadir, A. Sasongko, and M. Zulkifli, "Simple power analysis attack against elliptic curve cryptography processor on FPGA implementation," in *Proc. Int. Conf. Electr. Eng. Inf.*, 2011, pp. 2–5.
[33] S. Sun, et al., "Experiments in attacking FPGA-based embedded systems using differential power analysis," in *Proc. IEEE Int. Conf. Electro/Inf. Technol.*, 2008, pp. 7–12.
[34] K. Eguro and R. Venkatesan, "FPGAs for trusted cloud computing," in *Proc. 22nd Int. Conf. Field Programmable Logic Appl.*, 2012, pp. 63–70.
[35] Trusted Computing Group, "TPM main specification Level 2, revision 1.2, version 116," 2011. [Online]. Available: http://www.trustedcomputinggroup.org/tpm-main-specification.
[36] T. Eisenbarth, et al., "Reconfigurable trusted computing in hardware," in *Proc. ACM Workshop Scalable Trusted Comput.*, 2007, Art. no. 15.
[37] B. Glas, et al., "A prototype of trusted platform functionality on reconfigurable hardware for bit-stream updates," in *Proc. 19th IEEE/IFIP Int. Symp. Rapid Syst. Prototyping*, 2008, pp. 135–141.
[38] J. Cucurull and S. Guasch, "Virtual TPM for a secure cloud: Fallacy or reality?" in *Proc. RECSI*, 2014, pp. 197–202.
[39] V. Costan and S. Devadas, "Security challenges and opportunities in adaptive and reconfigurable hardware," in *Proc. IEEE Int. Symp. Hardware-Oriented Security Trust*, 2011, pp. 1–5.
[40] L. Xu, et al., "PFC: Privacy preserving FPGA cloud - A case study of map reduce," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2014, pp. 280–287.
[41] E. B. Barker, Recommendation for Key Management, Special Publication 800-57, Part 1 Rev. 4, NIST, 01/2016. [Online]. Available: https://www.nist.gov/publications/recommendation-key-management-part-1-general-0
[42] E. Peterson, "XAPP1084(v1.3): Developing tamper resistant designs with Xilinx Virtex-6 and 7 Series FPGAs," Xilinx, 2013. [Online]. Available. https://www.xilinx.com/support/documentation/application_notes/xapp1084_tamp_resist_dsgns.pdf
[43] S. Goren, O. Ozkurt, A. Yildiz, and H. F. Ugurdag, "FPGA bitstream protection with PUFs, obfuscation, and multi-boot," in *Proc. 6th Int. Workshop Reconfigurable Commun.-Centric Syst.-Chip*, 2011, pp. 1–2.
[44] C. Böhm and M. Hofer, *Physical Unclonable Functions in Theory and Practice*, 1st ed., New York, NY, USA: Springer, 2012.
[45] "Stratix 10 – Overview, " Altera, 2017. [Online]. Available: https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html
[46] K. Kaushik and V. Varadharajan, "One time trapdoor based searchable encryption," in *Proc. IEEE Int. Conf. Cloud Eng.*, 2017, pp. 167–174.
[47] R. C. Merkle, "Secure communications over insecure channels," *Commun. ACM*, vol. 21, no. 4. pp. 294–299, 1978.
[48] P. Maistri, et al., "Countermeasures against em analysis for a secured FPGA-based AES implementation," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, 2013, pp. 1–6.
[49] R. S. Chakraborty, et al., "Hardware Trojan insertion by direct modification of FPGA configuration bitstream," *IEEE Des. Test*, vol. 30, no. 2, pp. 45–54, Apr. 2013.
[50] T. Fitfield, "Introduction to OpenStack," *Linux J.*, vol. 2013, 2013, Art. no. 4.
[51] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "FPGAs in the cloud: Booting virtualized hardware accelerators with OpenStack," in *Proc. IEEE Int. Symp. Field-Programmable Custom Comput. Mach.*, 2014, pp. 109–116.
[52] M. Leonhard, "Cloudping info," [Online]. Available. http://www.cloudping.info
[53] G. Bertoni, et al., "On the indifferentiability of the sponge construction," in *Proc. 27th Annu. Int. Conf. Adv. Cryptol. Theory Appl. Cryptographic Techn.*, 2008, pp. 181–197.
[54] D. Suzuki, "How to maximize the potential of FPGA resources for modular exponentiation," in *Proc. Adv. Cryptol. - Cryptographic Hardware Embedded Syst.*, 2007, pp. 272–288.
[55] T. Good and M. Benaissa, "AES on FPGA from the fastest to the smallest," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2005, pp. 427–440.
[56] S. Kerckhof, et al., "Compact FPGA implementations of the five SHA-3 finalists," in *Proc. Int. Conf. Smart Card Res. Adv. Appl.*, 2011, pp. 217–233.
[57] "State of the Internet Quarterly Report," Akami. [Online]. Available: https://www.akamai.com/us/en/about/our-thinking/state-of-the-internet-report/.
[58] S. Brenner, et al., "SecureKeeper: Confidential ZooKeeper using Intel SGX," in *Proc. 17th Int. Middleware Conf.*, 2016, Art. no. 14.
[59] B. S. X. Allamigeon, and V. Cheval, "ProVerif: Cryptographic protocol verifier in the formal model," 2014. [Online]. http://prosecco.gforge.inria.fr/personal/bblanche/proverif



M. E. S. Elrabaa received MASc and PhD degrees in electrical & computer engineering from the University of Waterloo, Waterloo, Canada, in 1991 and 1995, respectively. From 1995 until 1998, he worked as a senior component designer with Intel Corp., Portland, Oregon, USA. He is currently a professor with the Computer Engineering department, King Fahd University of Petroleum & Minerals (KFUPM). His current research interests include reconfigurable computing, cloud-based custom computing machines and systems-on-chip. He authored and co-authored numerous papers, a book and holds seven US patents.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.

M. Al-Asli received the BSc and MSc degrees in computer engineering from KFUPM, in 2009 and 2012, respectively. He received the PhD degree in computer science and engineering from KFUPM, in 2017. His research interests include cloud computing, reconfigurable computing and VLSI.

Marwan Abu-Amara received the BS degree in computer engineering from Kuwait University, Kuwait, in 1989, the MS and PhD degrees in electrical and computer engineering from Texas A&M University, USA, in 1991 and 1995, respectively. From 1995 to 2003, he worked for Nortel Networks, Richardson, Texas, USA, as a senior technical advisor for the Wireless Network engineering group. Since 2003, he has been with the Computer Engineering Department at KFUPM. His research interests include computer and network security and resiliency, and wireless communications and networking.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on April 01,2024 at 12:14:25 UTC from IEEE Xplore. Restrictions apply.