Tamper-proof certification system based on secure non-volatile FPGAs

Diogo Parrinha

***Abstract*—Embedded systems supported on FPGAs are increasingly playing a bigger role on safety-critical areas. A particular example of a safety-critical system is a Hardware Security Module, providing private key management and usage in a secure and reliable way. However, commercially available systems are too expensive and limited in the provided functionality. On the other hand, existing volatile FPGA solutions do not adequately provide the needed security features. Herein, an open-source, low-cost and highly flexible re-configurable Hardware Security Module is proposed, supported by a System-on-Chip with a non-volatile FPGA. The presented solution operates as a versatile certification system that provides key management, digital signature services and is able to issue trustworthy certificates. The solution can be used, for example, in IT security applications through an integration with the included PKCS#11 interface. To further illustrate the flexibility of the proposed solution, a Log-Chain certification use-case is also presented. Experimental results suggest that the system is able to compute up to 2 sign/certification operations per second with a low cost, adaptable, and secure approach.**

***Index Terms*—HSM, Non-volatile FPGA/SoC, Certification**

I. Introduction

Modern Field-Programmable-Gate-Arrays (FPGA) provide an ever-growing programming possibilities, high flexibility and increasing hardware capabilities. For these reasons, there has been an expanding amount of applications for these devices, such as Data Centers, Medical, Aerospace, Defense and Security [1]. Along with this, the increasing need for data protection and system reliability, especially for safety-critical systems, has urged FPGA manufacturers to develop more secure and reliable devices, rather than solely focusing on power consumption and system performance. On the other hand, existing commercial security-oriented devices provide cryptographic operations and secure key management with an adequate performance but at a high cost and low flexibility (e.g. Hardware Security Module).

New FPGA technologies are starting to provide great flexibility and security at a low price [2], [3], allowing for the creation of cheaper systems with higher trust on their computation and ability to be easily re-configured. Over the last years and supported by these FPGA technologies, several authors have proposed secure computing platforms [4], [2], [5], using PUF-based key generation and re-keying mechanisms [6], [7]. FPGA based systems are also being proposed to perform security operations for safety-critical applications, such as a Secure Application Wrapper, which performs secure system authentication and data transfer with an external memory [8], or to allow users to offload sensitive computations to the cloud [9].

However, the existing state of the art solutions focus on particular problems and still present several vulnerabilities, partially due to the use of SRAM-based FPGAs [3], thus not allowing them to be used as Hardware Security Modules (HSM). An HSM system needs to meet certain mandatory requirements [10], such as anti-cloning mechanisms (e.g. PUF-based key generation), secure communication channels, internal non-volatile memories for master key storage, anti-tamper mechanisms, internal clock freshness (e.g. through a Timestamping Authority) and a common developer interface, such as PKCS#11.

Herein, a flexible HSM is proposed, supported by a non-volatile FPGA technology. Unlike existing commercial solutions, the considered system consists of an open-source design with the flexibility to add additional components both in software, as well as in the re-configurable fabric. The resulting system is capable of securely generating and managing keys, issuing digital certificates, generating digital signatures and generating and extracting key pairs through a PKCS#11 interface, as expected from a regular HSM, but on a cheaper and secure non volatile re-configurable device.

To illustrate the adaptability of the solution, a secure Log-Chain operation [11] has also been implemented. This complementary feature consists of creating a non-repudiable and certified chain-of-logs for Linux Syslog messages (that can also be used for transaction logs or medical receipts).

The obtained experimental results suggest that the resulting system is able to perform up to 2 sign operations per second on a Smartfusion2 SoC with a cost much lower than existing commercial HSMs, while providing the needed security and reliability features. This shows that with the existing non-volatile devices it is possible to create flexible and low-cost HSMs.

The paper is organized as follows: Section II presents the relevant state of the art, while the proposed solution is described in Section III. Section IV delineates the implementation of the proposed solution and the results are described and analysed in Section V. Section VI concludes the paper.

II. State of the Art

The following describes existing technologies allowing for the development of secure computing platforms that allow for the implementation of trusted computing systems. It starts by elaborating on possible implementation technologies, followed by an overview on SmartCards and HSMs. To conclude, existing FPGA-based secure computation solutions are analysed.

A. Implementation Technologies

There are three major implementation technologies when considering secure computation systems: CPUs with added security mechanisms, Application-Specific Integrated Circuits (ASIC), and FPGAs. In regard to CPUs, new architectures are being introduced that provide hardware security features to the software layer, such as ARM TrustZone or Intel SGX. While providing a very high flexibility at a low cost, these approaches are still subject to several attacks [12], [13].

ASICs allow to build integrated circuits for specific applications with fully dedicated security modules. ASIC based designs may also include microprocessors, memories, and dedicated components, potentially allowing for very secure systems (such as HSMs and Smart Cards). However, ASIC design approaches have very high fabrication cost when dealing with low market volume and have a low flexibility.

On the other hand, FPGAs are relatively low cost general-purpose devices that provide high flexibility and performance. They are composed of a Configurable Logic Block (CLB) array, which can be configured for the desired application after manufacturing. Recently, FPGAs are being integrated in System-on-Chip (SoC) designs, merging embedded CPUs, memories, and security modules with the FPGA fabric. The majority of FPGA technologies are supported by SRAM based storage, with only a few manufacturers providing non-volatile FPGAs.

Volatile FPGAs imply that their configuration data is stored in static memory cells and must be re-programmed on each start. The configuration bitstream must be read from an external source on boot or from an off-chip internal flash memory. Non-volatile FPGAs allow to store the configuration internally after the initial configuration.

B. SmartCards and HSMs

SmartCards and Hardware Security Modules provide a secure way to store and perform operations with private keys, while making sure these keys cannot be obtained from outside the device. While providing very high security assurance levels, Smart Cards provide a low throughput input/output communication and low computational processing capabilities. The main advantages of Smart Cards are their low price, their portability and flexibility (being used as credit cards, ID card, among others).

HSMs, on the other hand, are application-specific devices which provide secure cryptographic key generation, usage and management, for asymmetric and symmetric operations, such as data signing. Additionally, they provide anti-tampering features at the physical level. The main drawbacks are their high cost and low flexibility. There are several HSMs in the market for a variety of goals, prices, and with different characteristics. The criteria for choosing the right HSM for a given task includes performance, scalability, redundancy, API support, security, supported algorithms, authentication options and cost [10]. All support time synchronization and administrator authentication via PKCS#11 interface.

C. FPGAs as Secure Computing Platforms

In regard to secure computing platforms based on FPGAs, most of the relevant state of the art is supported by volatile (SRAM-Based) FPGAs. Most modern volatile FPGAs come with a secure boot process, in which the device loads an encrypted and authenticated bitstream from the configuration memory when powered on. While several ad-hoc solutions have been proposed [2] to strengthen the native technology, these devices are still vulnerable to several attacks, particularly when loading the initial configuration. The following describes the most related solutions in the state of the art.

Gaj et al. [4] consider the use of embedded microprocessor cores within the FPGA to achieve bitstream security, specifically for the reconfiguration of a Xilinx Virtex-II Pro on a Xilinx ML310 board. The authors consider the entire board as a secure device and not just the FPGA, meaning the path between the FPGA and the external memory is susceptible to tampering.

Arasu et al. [5] present the design of the Cipherbase secure hardware and its implementation using FPGAs. The Cipherbase system incorporates customized trusted hardware, extending Microsoft's SQL Server for efficient execution of queries using both secure hardware and commodity servers, allowing for the secure storage and exchange of data.

They implement their secure environment, the Trusted Machine (TM), using a volatile FPGA, which they assume to be secure and that an adversary does not have access to its internal state, since they believe that the TM is only vulnerable to side-channel analysis. The method they propose for generating encryption keys for database operations, consists of embedding a RSA key pair in the TM (with the public key published via a PKI), allowing clients to negotiate AES sessions for different fields and/or databases. The session establishment algorithm is not mentioned and the secure bootstrapping and operation of the FPGA is not described in their work. Moreover, they state that in case the keys are cached internally, they would have to be stored in a key vault, encrypted with a master key. However, they do not mention where this master key is stored and how it is generated, neither how the keys would be encrypted.

Nabeel et. al [6] propose strong hardware authentication for smart meters with efficient key management for Advanced Metering Infrastructures. Towards this, they use a Physically Unclonable Function (PUF) based on a feedback loop, in a Xilinx's Spartan-6 FPGA. While the usage of the PUF provides a strong protection against key leakage, the PUF error correction mechanism and cryptographic operations are done on the PC.

Wang et al. [7] propose a flexible and low-cost rekeying management scheme supported by a FPGA to improve the security and reduce the processing time of the rekeying processes in secure large-scale Storage Area Network applications.

To prevent the system from being attacked, the authors perform the key management at hardware level (on the FPGA), with the software simply sending commands to the key management module (hardware) using handlers to select the keys.

The internal memory stores the current active key pairs, while the outside flash memory backs up the remaining keys (after encrypting them).

The authors claim that their system prevents physical attacks because the keys are only used at the hardware level. At the software level, only key handlers are used. However, if an attacker has access to the devices, it can perform a multitude of attacks to the FPGA device and its configuration.

Eguro et al. [9] propose a solution in which protected bitstreams are used to create a Root-of-Trust for the clients using cloud computing servers (which contain several FPGAs). The proposed system allows clients to upload their configuration data to the cloud, by performing partial re-configuration of the FPGAs. Since cloud administrators do not have low-level access to the computation within the FPGA, they consider potential vulnerabilities in the software stack to be avoided.

The attack model considers the volatile device to be totally secure and impossible to break, which is not the case as shown in [2]. Moreover, partial re-configuration security mechanisms are not described and they do not consider the freshness of the uploaded configuration data.

Graf [8] proposes a secure FPGA wrapper allowing the user to upload application to run on the FPGA, creating a secure user authentication interface and cryptographically securing the data transfers between the applications. The solution considers a secure token-based authentication scheme (using Java's iButton) and an FPGA-based encrypted memory controller. It is important to note that the user application which runs within the FPGA (protected by the wrapper) can only be re-programmed in the factory as there is no interface to reconfigure the users' applications.

The protocol used to negotiate the secure channel between the device and the iButton uses two RSA key pairs. However, the public keys are never made public, instead, they are embedded in the iButton and the created FPGA bitstream, with each side storing the public key of the other. Given this, every time a new iButton ID is added, the bitstream must be modified.

Overall, the state of the art solutions are rather specific to particular problems, lacking several of the features required to build a HSM. Moreover, these solutions make use of volatile FPGAs, which do not provide the needed protection level for safety-critical applications. Additionally, except for [6], none of the proposals consider the use of PUF-based mechanisms for secure key management. Finally, none of the devices contain hardware security cores that provide side-channel analysis protection and do not provide the ability to perform secure communication with external parties, while guaranteeing authentication and freshness of the exchanged data.

III. Proposed Solution

In this paper, a low-cost and highly flexible Hardware Security Module solution is proposed, capable of securely managing keys, computing digital signatures, issuing digital certificates and performing user key generation and usage. To support this, the use of the SmartFusion2 System-on-Chip from Microsemi is considered. Unlike the state of the art solutions, the Smartfusion2 SoC integrates a non-volatile FPGA, which uses flash memory for the configuration storage, rather than static memory cells. They consume less power and are more tolerant to radiation effects. Since they are non-volatile, the bitstream is at a lower risk of being probed, particularly during boot. According to the existing surveys [3], [14], the Smartfusion2 SoC offers the best variety of design and data security features when compared to other FPGA models from different vendors.

The main components of the proposed system, illustrated in Figure 1, include the main CPU, the embedded security cores, a real-time clock, internal memory, and IO interfaces. The integration of the provided features with the applications running on a PC is done through an extended PKCS#11 middleware (and resulting device driver), abstracting the users from the inner workings of the system. The created certificates are formatted according to the X.509v3 standard.

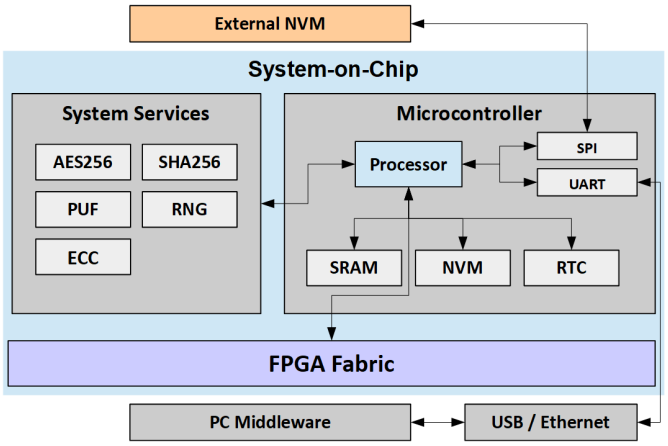

Fig. 1. The proposed overall system architecture. The light blue rectangle represents the secure System-on-Chip. The contents of the external Flash are encrypted.

To demonstrate the flexibility of the proposed architecture, the implementation of a novel log certification functionality, for the Linux Syslog, is also developed as a use case. This functionality consists in creating a signed Log-Chain (for transaction logs or for certifying medical receipts). Each message/command signature guarantees the authenticity of that message and all previously logged messages/commands, therefore creating a chain-of-logs that cannot be repudiated or modified.

The following describes how user and key management is performed, including key storage and validation, the used communication protocol, and details the proposed Log-Chain use case.

A. User and Key Management

In the developed system, two types of accounts are considered, namely a single administrative account and multiple (non-privileged) user accounts, each associated with an

asymmetric key pair. The administrator is able to generate certificates for external public keys and is able to create new users accounts, each with a new internal key pair. Users are allowed to request data signature operations using their (non-extractable) private key, update their PIN, request the generation of key pairs to be exported (not stored internally), and other available user operations, such as the addition and signing of Log-Chain entries, as provided in the use case.

Other than the user keys, the system uses two asymmetric key pairs (one for the root certification and one for device authentication and secure session establishment) and one symmetric key for external data wrapping. An additional asymmetric key pair is generated and stored for the Log-Chain signatures used in the presented use case. During device initialization, these keys are generated and stored internally using the FPGA built-in PUF mechanism. The use of PUFs provides additional protection against cloning and malicious private key retrieval.

Since most embedded systems only have small internal memories, the users' data (such as ID, PIN, keys and certificates) is stored on the external Flash memory, protected by the wrapping key. This data is encrypted using AES-256 in CBC mode and stored in the external memory. To assure data integrity and authenticity, the SHA-256 digest of the encrypted data, along with the IV used in the CBC (randomly generated for each block) are stored in the internal Flash memory. This way, the system can be sure that whenever the external data is accessed, it is valid and has not been tampered with. Rollback attacks cannot be performed on the external memory because if the information is different than the current data, the digest checks will reject the data and invalidate it.

The users' asymmetric key pair is generated using the existing True Random Number Generator. The resulting public key can be exported within an X509 certificate signed using the systems' root certification key. All asymmetric encryption is performed using Elliptic Curve Cryptography (ECC), given its better efficiency on embedded systems with low computational power and limited memory storage [15], [16].

For user login operations and authorizations, the respective PIN value must be inserted, according to the PKCS#11 standard.

In order for the system to properly manage the certificates and time dependant operations, a correct time/date keeping is required. However, as most embedded systems cannot provide trustworthy timestamps (as their Real-Time-Clock is reset on power-on), an external certified time/date value must be obtained.

Although the connecting PC could send its time to the device, it is not considered as a trustworthy element. Therefore, when the device is initialized and a first session is established, the device sends a challenge to the PC, which must be signed by a pre-defined Time Service (a service that provides a certified date and time) and sent back to the device within a limited time interval (e.g. 10s) with the challenge and time properly signed. In order to verify the Time Service signature, the device must be shipped/configured with the public key of the trusted Time Service. The device checks the 10 seconds interval using the internal Real-Time-Clock.

To assure an adequate root of trust, the public keys of the system must also be properly published and announced.

B. Communication and Session Establishment

For the connection between the device and the PC (e.g. via a USB or Ethernet link), two scenarios are considered, namely one where the connection is considered secure and the other where the connection cannot be considered secure. In the latter, the PKCS#11 messages, where for example the user authentication PIN is sent, must be protected by a cryptographic channel. This is achieved by establishing a secure channel using the Elliptic Curve Integrated Encryption Scheme(ECIES).

The communication can be configured by the administrator to be made through an open or a secure channel. When configured to communicate through a secure channel, session keys must be established before both parties (PC and device) can exchange data. Towards this, the middleware generates an ECC key pair (which is newly generated for each session) and using the known device Secure Session public key (P_{Device}) starts the ECDH protocol with the device, to define a new shared secret (S) using. From this shared secret and using the Key Derivation Function PBKDF2 (defined in PKCS#5) the encryption key (K_C) and the authentication key (K_H) are obtained by both parties, as illustrated in Table I.

PC/Middleware		**Device**
Obtain S with ECDH Derive K_C and K_H		
	$\{N_1\}_{K_C}, P_{PC}, HMAC \longrightarrow$	
		Obtain S with ECDH Derive K_C and K_H
	$\longleftarrow \{N_2, N_1'\}_{K_C}, HMAC$	
Verify N_1'	$\{N_2'\}_{K_C}, HMAC \longrightarrow$	
		Verify N_2'
	$\{Data, Ctr\}_{K_C}, HMAC \longrightarrow$	
	$\longleftarrow \{Data, Ctr+1\}_{K_C}, HMAC$	
	$\{Data, Ctr+2\}_{K_C}, HMAC \longrightarrow$	

TABLE I
SECURE SESSION ESTABLISHMENT.

After a successful key establishment, the data can be exchanged between the PC and the middleware securely. This is done by encrypting the data and a counter value (derived from the nonces exchange in the first phase) using AES-256 in CBC mode, and authenticating it with an HMAC-SHA256 using the derived authentication key (K_H).

Note that, while the PC is ensured to be communicating with the device, this protocol does not allow for the authentication of the PC or the user. This is achieved during the PKCS#11 login phase, using the corresponding PIN.

C. Use Case: Log-Chain

To illustrate the flexibility and configurability of the proposed system, a Log-Chain feature was added, allowing for

a reliable way to log messages or commands into a signed chained log. This Log-Chain feature ensures that an entry cannot be removed from a log without breaking the cryptographic validity of the chain. This is done by having each log entry concatenated with the hash of the previous entry, the current counter value, and signing it with the Log-Chain private key. Note that, the counter is managed and kept inside the device, as well as the Log-Chain private key, which is only located and used in the device and cannot be extracted.

Each log entry is composed of the message (or command), the date and time, the identification of the requesting user (UID of the device), the internal counter, and the previous entry hash result, as:

Log Entry
Message_i | date_time | UID | counter | Hash_{i-1}

For example, if user 2 requests the addition of message "[670164.882] rm *.tex", the log entry will be:

[670164.882] rm *.tex | 10-04-2016,13:10:20 | 2 | 27 | FB$\cdots$71

where this would correspond to the 27th log entry and the hash of the previous entry request would be FB$\cdots$71.

Using this operation, it is possible to create a Log-Chain file. This is managed by the middleware at the PC side using dedicated calls. This middleware is responsible for the creation, management, and storage of the log files. To start a new Log-Chain, the administrator must define an initial hash value (as there is no $hash_{i-1}$) and a message. Following this, the file will contain each log entry with the respective system date, hash, and resulting signature, as depicted in Figure 2.

10-04-2016,10:54:10:{log_root} [*hash0*] [*signature0*]
10-04-2016,10:54:15:{log_entry_1} [*hash1*] [*signature1*]
10-04-2016,10:54:17:{log_entry_2} [*hash2*] [*signature2*]
10-04-2016,10:54:18:{log_entry_3} [*hash3*] [*signature3*]

Fig. 2. Log file example. Each horizontal line represents a line break. Hashes and signatures are Base 64 encoded.

To optimize the communication between the PC and the device, the hash values are computed in the middleware at the PC side and sent to the device. Herein, it is assumed that the PC is able to securely acquire each message/command (for example by having a daemon running in kernel mode) and is able to store the resulting signature and associated log entry into the file system. The device is only responsible for signing the log entries and ensure the signing order (by adding the device time and internal counter value). Note that, to accept the addition of a log entry (described above), and perform its signing, a valid user (UID) must establish a session with the device and authenticate himself using his PIN. However, thousands of logs entries may be requested per minute, making their signing computationally not feasible, especially on a low-cost embedded system. To cope with this, the proposed approach allows the application using the API, to decide when to request a signature. Instead of requesting the signing of every message/command, the application can request to sign every X messages and always requests the signature generation of the last one, as depicted in Figure 3.

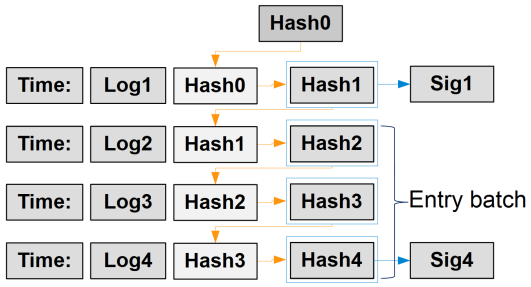

Fig. 3. The structure of a log chain with grouped log entries.

This allows to increase the system efficiency and adjust it to the performance of the device. Security-wise this is equivalent to signing each entry, but in order to validate a log entry belonging to the batch, the entire batch needs to be verified.

To validate any part of the chain, one only needs to use the starting hash value for that part of the chain, compute the intermediate hash values, until the chain point that needs to be validated, and validate the signature of that ending point. This is a relatively light process, since other than the hashing of each entry, only one signature needs to be verified. The developed middleware already extends the PKCS#11 interface to provide this functionality.

IV. Implementation

For the prototyping of the proposed certification system, the Smartfusion2 SoC is considered, in particular the SmartFusion2 Security Evaluation Kit with a Smartfusion2 90-TS. This SoC integrates a low power 32-bit ARM-Cortex M3, on-chip SRAM and non-volatile memory (eNVM), non-volatile FPGA fabric, and an embedded security core [17]. This security core provides AES-256, SHA-256, and ECC multiplication (NIST P-384 curve) primitives along with a SRAM-PUF.

While the device supports uClinux, the amount of memory required is too high for the available embedded memory. This would imply the use of the external memory, which would leave the system susceptible to several attacks. Given this, a BareMetal implementation was considered, providing a much more compact solution. Additionally, a BareMetal approach allows for a higher code verification and control without the typical unknown vulnerabilities of OS-based solutions. To assure better user isolation, the system was designed as single threaded, supporting one session at a time.

For the cryptographic implementations, the mbedTLS 2.4.2 library (ARM®mbed) [18] was chosen, since it was developed with embedded systems in mind. This library provides software implementations of several encryption algorithms, for the generation of X.509v3 certificates, and other protocols, such as ECDH and ECDSA. Given its modular structure, it is possible to disable unused features, towards a lower code footprint, also allowing for the integration of alternative implementations to these algorithms.

To ensure that the system boots securely, power-on digest and clock frequency checks have been enabled. This

mechanism checks the digest of the internal memories and FPGA fabric configuration. To achieve its maximum protection against external attacks during operation, the system was configured to lock read/write/verify configuration and debug operations. The tamper detection mechanisms have also been enabled allowing for device Zeroization when an attack is detected. Meaning that, if an attack is detected, the device is completely reset losing its configuration. While this may cause a Denial of Service, it is the most reliable way to ensure the protection of the internal keys and data.

In regard to the available volatile memory, and given the limited available internal memory, both the eSRAM (64kB) and the FPGA fabric SRAMs (up to 64kB) were used as the system memory. The FPGA fabric can be configured in such a way that the FPGA's embedded memories can be combined and made available to the processor, while allowing the remaining logic of the FPGA to still be used.

While the SRAM-PUF service is used to store the administration keys, the remaining non-volatile data needs to be securely stored. However, the internal eNVM presents several challenging limitations. In particular, the memory endurance is limited to about 1000 writes cycles to each page of 128B [17]). It is also rather limited with a total of 512kB, partially used to store the developed ARM code.

The considered solution was to store the essential data in the internal eNVM, while the remaining data is stored on the reasonably larger external SPI Flash memory (with 8 MB), allowing about 100,000 write cycles per sector. This data is stored encrypted with the wrapping key, while the digest of the stored data is stored internally to assure the external data has not been tamper with.

The external data is organized and written in sectors of 4KB each. This is exploited by storing the data of each user (which includes the users' private key and public keys, identifier, and PIN) in isolated pages, making it easier to manage and re-encrypt when needed.

To take advantage of the available embedded security hardware and reconfigurable fabric, two additional versions of the system were also implemented. One using the AES-256, SHA-256 and ECC primitives, available in the SoC embedded security hardware core, and another one using a dedicated SHA-256 core [19] implemented on the FPGA fabric. These modifications were facilitated by the modularity of mbedTLS. However, since the SoC only provides the ECC scalar operations, and mbedTLS does not allow modularity in the ECC code, it was required to directly modify the point multiplication routines to use the embedded hardware security cores of the device.

V. Results and Analysis

In order to evaluate the performance of the developed certification system, the three created versions were implemented using the Libero SoC 11.7 and SoftConsole 3.4 for the architecture and code compilation, respectively. These three versions consist of the software-only version; the version that uses the embedded cores provided by the SmartFusion2 SoC (AES, SHA-256, and ECC scalar operations); and finally the FPGA accelerated version, which is identical to the embedded version but computes the SHA-256 primitive on the FPGA fabric.

Before evaluating the global performance of the system, the most data intensive cryptographic operations were evaluated, given the considered implementations. The obtained results were obtained on the device-side using the internal Real Time Clock.

The obtained results for the SHA-256 computation, depicted in Figure 4, suggest that when computing large amounts of data, the embedded core is able to achieve higher throughput results (10Mbps) than the software-only version (3.5Mbps), considering only the processing time for a 512-bit input block. The 512-bit input processing time, which includes data transfer, for the embedded SHA-256 core is about $53\mu s$, whereas the software-only version takes $148\mu s$.

However, for small amounts of data, the software based implementation provided by mbedTLS, can actually achieve better results than the computation using the embedded SHA-256 core. In fact, the embedded core version requires about $529\mu s$ to calculate the hash of 1kbit of data, which is almost the same as when computed in software ($564\mu s$). This is due to the fixed cost of $423\mu s$ to start-up the computation process, i.e. invoking and setting up the embedded core (e.g. field initialization and pre-computation). In the software call, invoking the algorithm takes a negligible amount of time, but there is still a fixed cost of $268\mu s$ for the algorithm start-up routines.

The FPGA based implementation has a starting overhead of just $5\mu s$, and a block processing time of $26\mu s$ (including data transfer). Therefore, the maximum throughput is about 20Mbps, which is approximately 2 and 6 times larger than the embedded core and software version throughputs respectively, at a cost of 5% FPGA usage.

For the AES, an identical issue occurs when using the embedded AES core, with a start up time of $505\mu s$. To encrypt each 128-bit data block, it requires $175\mu s$ towards a maximum throughput of about 730kbps. Given this start up time and encryption rate, it requires $1.9ms$ to encrypt 1kbit of data. The software implementation requires $0.55ms$ to encrypt 1kbit of data, reaching a maximum throughput of about 1.9Mbps.

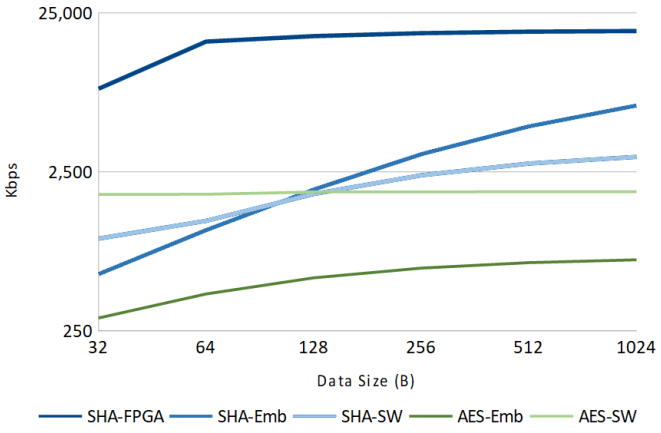

Fig. 4. SHA-256 and AES-256 throughputs.

The actual performance gain provided by the embedded security core comes with the Elliptic Curve operations. The scalar multiplication is much slower when performed in software, requiring 28.4s per operation, than when performed in the embedded security core, requiring 0.57s per operation.

This impact is noticeable in the overall operations time, depicted in Table II for the three implemented versions. The use of the embedded security cores or FPGA improved

TABLE II
OPERATION TIMES FOR THE THREE VERSIONS CONCEIVED.

Operation	FPGA-accelerated	Embedded	Software
Gen. Key Pair	580 ms	590 ms	28.4 s
Sign	784 ms	788 ms	28.5 s
Gen. Certificate	809 ms	816 ms	28.6 s
Add log entry	791 ms	794 ms	28.5 s

implementations allows for speedups between 35 to 48 times when compared to the software-only implementation, impacted mostly by the used ECC operations. The performance difference between the FPGA and embedded implementations is rather reduced, resulting from the use of the FPGA-based SHA-256 computation.

Given that the embedded security core provides some side-channel protection mechanisms for the ECC operations and that when using this implementation, the operations time are below 1 second, no FPGA-based ECC operations are considered. Nevertheless, the reconfiguration capabilities and modular structure of the proposed design allows for additional FPGA accelerators to be added.

Regarding possible attacks, the used secure communication channel prevents not only eavesdropping but also Man-in-the-Middle attacks, since the resulting session key depends on the private keys of both entities and the PC knows the public of the device. The device does not care with whom it is communicating as long as it receives the correct user ID and PIN values during the PKCS#11 session. Additionally, perfect forward secrecy is also ensured, since the session key depends on both private keys (PC and device) and the key pair from the PC is randomly generated for each connection. Once the session is terminated and the private key from the PC destroyed, the session key can no longer be recovered. Given that each message is accompanied by a counter, verified by both parties, the system is also protected against replay attacks. Spoofing attacks are also not possible since the user must authenticate itself before using the certification system (using his UID and PIN) and the certification device authenticates itself by proving the procession of the respective private key.

In regard to the state of the art, the existing academic proposals lack mandatory requirements to be used as HSMs, such as a secure communication channel, anti-cloning mechanisms (e.g. PUF-based key generation), internal non-volatile memories for master key storage, high entropy random number generators, anti-tamper mechanisms, internal clock freshness (e.g. through a Timestamping Authority) and a common developer interface, such as PKCS#11.

In regard to HSMs that exist in the market [10] (such as AEP Keyper v2, SafeNet Luna SA 4.4, Thales nShield Connect 6000 and Ultimaco CryptoServer Se1000) support the typical signing and key management operations, for several algorithms, such as AES, RSA, ECDSA, ECDH and SHA-2, which are available through a PKCS#11 interface. The HSM with the lowest data signing performance, the AEP Keyper v2, is able to perform 310 signatures per second for a key size of 1kbit, and as low as 13 signatures per second for a key size of 4kbits. On the other hand, the CryptoServer Se1000 is the fastest HSM, performing up to 1,160 signatures per second for keys of 1kbits. Concerning the internal key storage capacity, the AEP Keyper v2 is the one supporting the highest amount of keys, up to 8000 1k bit keys, while the one that supports the least amount is the Thales nShield, which has limited onboard storage.

The proposed system is significantly slower in its current version (with the ECC operations performed in the embedded core and not in the FPGA), being able to perform about 2 signatures per second. The key storage capabilities are also limited to the available device memory, storing up to 4,096 wrapped keys (and respective user data) in the external ROM.

However, the proposed work presents a fully functional, open source, and adaptable HSM system with customization capabilities, as demonstrated by added the Log-Chain functionality. Moreover, in terms of pricing, a regular HSM can cost up to 35,000€ [20], [21] while the presented implementation, on a Smartfusion2 90-TS SoC Security Evaluation kit, as an approximate cost of €400.

VI. CONCLUSION

In this paper, a low-cost and highly flexible open-source HSM is proposed and implemented on the Smartfusion2 SoC device. Unlike the state of the art, the developed system takes advantage of non-volatile technologies, with built-in security features, such as side-channel analysis protected embedded cores, PUF-based key management, and secure session establishment with external parties, while guaranteeing the confidentiality, authentication, and freshness of the exchanged messages.

The developed system is able to generate digital signatures, issue certificates for public keys and generate and extract asymmetric key pairs for users. To demonstrate the flexibility of the proposed solution, the use-case Log-Chain is also presented. Performance-wise, a thorough analysis of the cryptographic operations shows that the system is able to perform up to 2 sign operations per second on a Smartfusion2 SoC with a cost much lower than existing commercial HSMs, while providing the needed security and reliability features.

REFERENCES

[1] Global Market Insights, Inc. (2016) FPGA Market size worth $9.98bn by 2022. [Online]. Available: https://www.gminsights.com/pressrelease/field-programmable-gate-array-fpga-market

[2] H. Kashyap and R. Chaves, "Compact and on-the-fly secure dynamic reconfiguration for volatile FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 9, no. 2, p. 11, 2016.

[3] R. Druyer, L. Torres, P. Benoit, P.-V. Bonzom, and P. Le-Quere, "A survey on security features in modern FPGAs," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2015 10th International Symposium on*. IEEE, 2015, pp. 1–8.

[4] A. S. Zeineddini and K. Gaj, "Secure Partial Reconfiguration of FPGAs," *Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology*, pp. 155–162, 2005.

[5] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan, "Orthogonal security with cipherbase." in *CIDR*, 2013.

[6] M. Nabeel, S. Kerr, X. Ding, and E. Bertino, "Authentication and key management for advanced metering infrastructures utilizing physically unclonable functions," in *Smart Grid Communications (SmartGridComm), 2012 IEEE Third International Conference on*. IEEE, 2012, pp. 324–329.

[7] Y. Wang and Y. Ha, "FPGA based Rekeying for cryptographic key management in Storage Area Network," in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*. IEEE, 2013, pp. 1–6.

[8] J. Graf and P. Athanas, "A key management architecture for securing off-chip data transfers," in *Field Programmable Logic and Applications (FPL)*, 2004, pp. 33–42.

[9] K. Eguro and R. Venkatesan, "FPGAs for trusted cloud computing," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 2012, pp. 63–70.

[10] J. Ivarsson, A. Nilsson, and A. Certezza, "A Review of Hardware Security Modules," opendnssec.org, Tech. Rep., 2010.

[11] B. Tulu, H. Li, S. Chatterjee, B. Hilton, D. Beranek-Lafky, and T. Horan, "Design and Implementation of a Digital Signature Solution for a Healthcare Enterprise," *AMCIS 2004 Proceedings*, p. 43, 2004.

[12] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza, "AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 440–457.

[13] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "ARMageddon: Cache attacks on mobile devices," in *Proceedings of the 25th USENIX Security Symposium*, 2016, pp. 549–564.

[14] T. Feller, "Towards Trustworthy Cyber-Physical Systems," in *Trustworthy Reconfigurable Systems*. Springer, 2014, pp. 85–136.

[15] S. Selvakumaraswamy and U. Govindaswamy, "Efficient transmission of pki certificates using elliptic curve cryptography and its variants." *International Arab Journal of Information Technology (IAJIT)*, vol. 13, no. 1, 2016.

[16] M.-D. Cano, R. Toledo-Valera, and F. Cerdan, "A certification authority for elliptic curve X. 509v3 certificates," in *Networking and Services, 2007. ICNS. Third International Conference on*. IEEE, 2007, pp. 49–49.

[17] Microsemi, "IGLOO2 FPGA and SmartFusion2 SoC FPGA," Microsemi, Tech. Rep., 2016.

[18] (2016) mbedtls. [Online]. Available: https://tls.mbed.org/

[19] (2016) SHA-256 HASH CORE. [Online]. Available: https://opencores.org/project,sha256_hash_core

[20] (2009) Thales nshield connect offers enterprise-class key management. [Online]. Available: http://www.networkworld.com/article/2246758/security/thales-nshield-connect-offers-enterprise-class-key-management.html

[21] "Safenet network hsm," 2017, https://www.infosecservice.com/product/safenet-network-hsm/.