Secure Message Exchange System based on a SmartFusion2 SoC and its Evaluation as a HSM

Alexandre Rodrigues
INESC-ID, IST, Universidade de Lisboa
alexandre.v.rodrigues@tecnico.ulisboa.pt

***Abstract*—Hardware Security Modules (HSM) play a critical role in system security and cryptographic key management. However, they seldom have been used for securing communications between organizations and people. This paper presents a solution focused on providing authentication and confidentiality to communications, using a low-cost and portable HSM. This work studied and evaluated the services of the SmartFusion2 System-on-Chip (SoC), as the HSM of this system. The board provides a robust set of security services: AES encryption, a hashing function, HMAC, ECC primitives, a true random number generator, tamper detection and zeroization. The performance of these services was modelled and their limitations presented. The developed proof of concept focused on continuously encrypting and authenticating communications as an intermediary between the owner, and other communicating users. A key management solution is also provided, which takes advantage of the secure storage, given the device limitations, allowing for regular key updates. The developed system also provides shared key agreement with ECDH. The board is well-equipped to function as a HSM, but has some feature, memory and performance limitations herein analysed. This work provides the necessary groundwork and analysis for future work, using the SmartFusion2 as a HSM.**

***Index Terms*—Hardware Security Module; SmartFusion2 SoC; Confidentiality; Authentication.**

I. Introduction

Communications between individuals and organizations are a recurring target for attackers. This is of special concern to high profile individuals, who handle sensitive information, such as government officials and company executives. The security of communications depends on the cryptography keys and passwords used. These are usually stored, along with other sensitive information, in the user's computer, or not at all. A more hardened and secure solution is to separate the computer used for message exchange, and the device responsible for managing the sensitive data. A secure and independent solution is needed to establish secure channels of communication, and bear the responsibility of key storage and management.

There are dedicated devices currently on the market, designed to secure data and store keys. These types of devices have physical tamper-resistant measures against attackers, who wish to read the information on the device. They also provide fail-safe mechanisms in case of an attack. Smart Cards provide secure and portable tamper-resistant storage. They have lower processing power, and a smaller memory. They have a lower cost, so can be produced in bulk and easily replaced. Hardware Security Modules (HSM) are high grade devices, with more computational power and larger storage capacity for secrets. HSM have seldom been used for securing people's communications. They are frequently used to solve specific problems in the retail, server and healthcare industries [1], [2], [3], [4]. There is an available opportunity and need for a system, based on these devices, to enable and secure data exchange between entities. Such a system should store all security critical information and allow authorized individuals to securely communicate, while having a relatively low cost and be easy-to-use, so it can be distributed to multiple entities.

Herein, a system to secure channels of communication, built on a low cost SmartFusion2 System-on-Chip (SoC) is proposed. Contrary to existing solutions, this system focuses on providing confidentiality and authentication to data exchange between individuals, using internal keys, with the flexibility to model it to different use case scenarios. Additionally, the system can generate digital signatures and shared keys with asymmetric cryptography, allowing users to connect with new entities with similar devices, without exposing keys. The crafted system provides secure key storage. A key management solution was devised to improve the device's storage, by overcoming key storage limitations and increasing its lifespan, while retaining its functionalities. A common developer interface, with PKCS#11, is available to interface with all the developed services. This provides flexibility for developers to adopt this system for their own solutions. To demonstrate this, the system was used in a TCP library to secure the data channel between the client and server, using the device as an intermediary.

The SmartFusion2 cryptographic accelerators and developed services' performance was tested. Accurate performance models of these services were calculated and are presented.This provides an accurate prediction of the board's behaviour to future developers to base their work on. Compared to existing market solutions, this system provides a low cost HSM implementation and characterization. It offers comparable capabilities, engineered to their optimal level, applied to the presented scenario.

The paper is organized as follows: Section II covers the necessary technical background. Section III presents the relevant state of the art. Section IV presents the proposed solution, while Section V lays out the implementation details. The proposed solution evaluation and analysis is presented in Section VI. Section VII concludes the paper.

II. Background

This section details the technical concepts required to understand the proposed solution. It provides an overview of cryptography services and algorithms. Then it presents several general purpose computing devices.

A. Cryptography

There are four important cryptographic services relevant to this work. Confidentiality is used to scramble information, and hide the content from unauthorized entities. Integrity protects data from unauthorized modification. Authentication ascertains the origin of a message. Non-Repudiation prevents an entity from denying the authorship of a document or message. Symmetric ciphers are frequently used to provide authentication and confidentiality, using symmetric keys. Advanced Encryption Standard (AES) is one of the most popular symmetric-key algorithms. Asymmetric cryptography is often used to provide non-repudiation through digital signatures and agree on shared symmetric keys. There are two popular algorithms for public-key encryption, RSA and ECC [5]. ECC keys have the same level of security as RSA keys, with a smaller key size, e.g., a 160-bit ECC private key has similar security to a 1024-bit RSA key [6]. Therefore, ECC keys are more suitable for devices with storage limitations. ECDH is a popular key agreement algorithm and ECDSA for generating digital signatures, both using ECC keys.

B. Hardware Security Modules

A HSM is a high grade computational device, responsible for storage, management and generation of cryptographic keys, as well as cryptographic operations. Keys are never exposed outside and all operations are performed inside the HSM. These devices have physical security mechanisms to achieve tamper-resistance, random number generators, support several cryptographic algorithms and have fail-safe mechanisms in place, in case of an attack, e.g., overwriting all memories and configuration with zeros (zeroization) [7], [8]. These modules are usually costlier than other computational systems but are more advanced in processing power and available services.

III. State of the Art

In general, HSM have been applied in several contexts, to exploit their cryptographic services, secure storage and physical tamper protections.

Lesjak et al. [1] developed a system to secure remote snapshot acquisition, between the vendor and customers, by attaching a HSM to the products distributed among customers. The messages are protected with the authenticated-encryption scheme AES-GCM and a TLS connection. The Infineon security controller stores the TLS keys, and protects the data with the authenticated-encryption scheme, using its True Random Number Generator (TRNG) and a DH based algorithm, for key establishment with ECC keys. The controller has protections against side-channel attacks such as Differential Power Analysis (DPA) and physical manipulation.

Seol et al. [2] proposed a system to isolate critical operations and sensitive data from cloud administrators, by implementing a HSM next to a virtual machine.

Wolf et al. [3] implemented a HSM on a 663 Xilinx Virtex-5 FPGA, to secure network communications in vehicles. The authors implemented several cryptographic algorithms on the FPGA, e.g., AES-128 bit and ECC point-multiplication with a 256 bit curve. The board was connected to a microcontroller running linux, with additional algorithms available from a cryptographic library.

An IBM 4764 PCI-X cryptographic coprocessor has been used to store and manage symmetric keys, which encrypt biomedical data [4]. The symmetric keys, used to encrypt the database, are transferred to the device using public-key cryptography. All database queries are performed by the coprocessor, since only it has the keys. The system uses AES with 128 bit encryption. Notably, the device has physical measures which ensure the keys are not leaked and the data is erased upon any attack.

Wherry [9] recognizes the need for a HSM in public key infrastructures (PKIs) to protect the cryptographic keys. Lorch et al. [10] uses an IBM 4758 cryptographic coprocessor to protect keys in a secure online repository for PKIs. The author's were able to store more than 800 2048-bit RSA key pairs on the device's secure storage. The PKI system interfaces with the HSM using a PKCS#11 interface. Keys are generated in the coprocessor and the private key is never extracted. The RSA implementation is used to sign certificates, while the public key can be extracted to the application. A PIN is necessary to access the coprocessor.

Beyond actual systems, several protocols and HSM applications have been proposed. Rssler et al. [11] applied a HSM to an e-voting electronic ballot box. The HSM is used for decrypting and verifying the signature of cast votes. The votes decryption is done solely inside the electronic ballot box during the counting process. Voters sign their ballot using a smart card, e.g., their citizen card. Only the HSM is capable of counting votes, using its private key. The author recommends 1024 bit RSA or 160 bit ECC keys for an actual implementation.

Additionally, authors have proposed using a HSM to secure web services by providing secure storage for keys and cryptography algorithms in the TLS protocol, but also for providing a complete security service, not just an algorithm implementation [12], [13].

Martina et al. [14] presents OpenHSM, an open cryptographic protocol to manage private keys in an application embedded in a HSM. The protocol was implemented with a customized FreeBSD system. The authors introduced administrator and operator groups to manage private keys inside the HSM. The hardware was projected to be tamper proof using a Security Unit (SU) to manage all sensors and protection mechanisms. The OpenSSL library and SQLite database were used to provide smart card support, data storage, secret sharing and X509v3 certificates.

Apart from developed systems, several market HSMs

have been studied. Kehret et al. [15] studied two devices. VaultIC460, a secure microcontroller manufactured by Inside Secure with a RISC CPU. It includes a varied offering of cryptographic algorithms, such as, AES encryption, public-key cryptography with RSA and ECC, Message Authentication Code (MAC), SHA, SSL support, as well as a random number generator. Additionally it includes several authentication mechanisms for users, to secure the connection between the application and device. It includes 112 KB of tamper resistant memory for key storage. ATECC508A from Microsemi is a small security controller with the asymmetric key algorithms: ECDSA and ECDH, along with SHA, a TRNG and storage of up to sixteen 256 bit keys. These types of controllers, in general, are not suitable for this work. They are very limited, with no symmetric key algorithms, preventing encryption of large amounts of data. They are designed to be added to Internet of Things devices.

The survey [16] studies the features of four HSM on the market, Keyper v2 by AEP, nShield Connect 6000 by Thales, Safenet Luna and Utimaco CryptoServer. All devices support authentication using smartcards, password or a PIN. The AEP and Utimaco also have additional smartcard integration, for backing up the device's internal keys. All devices have tamper resistant storage, a TRNG, as well as a varied range of supported cryptographic services. Several AES encryption modes for both 128 and 256 bit keys, SHA, HMAC and public-key cryptography with RSA. ECDSA and ECDH are supported by the devices from Safenet and Utimaco. All devices provide a PKCS#11 interface implementation for all cryptography services. The provided API can be used to build an application adapted to each user's requirements. The PKCS#11 implementation does not output any unencrypted sensitive information, such as keys.

One of the smallest and cheapest devices, the YubiHSM 2 by Yubico [17], is a USB sized device for €650. It supports several SHA algorithms, RSA, the asymmetric key ECC algorithms: ECDSA and ECDH, with multiple curves, a TRNG and the AES-CCM authenticated encryption algorithm. It provides a PKCS#11 implementation, 128KB of tamper resistant storage for keys, and an authenticated and encrypted connection, between the PKCS#11 API calls and the device. Compared to these HSMs which provide a PKCS#11 interface along with the device, the proposed solution offers a lower cost system with an API optimized for secure communications, using a SmartFusion2 SoC.

IV. Proposed Solution

This paper proposes a low cost system based on a HSM responsible for securing communications between individuals and organizations. The HSM acts as an intermediary, where the data is forwarded and received back, before it is sent to its destination. The SmartFusion2 SoC was analysed to function as a HSM, and be used as the HSM of the system. The board integrates a non-volatile FPGA with a SoC, an internal Non-Volatile Memory (eNVM) of 512 KB and SRAM-PUF secure storage. Its RAM has 64 KB protected against SEU or 80 KB unprotected. The board has an embedded ARM Cortex-M3 processor with a TRNG and some cryptography algorithms: AES, SHA-256, HMAC and ECC. The device also has tamper detection mechanisms which can be used to trigger zeroization, which erases all its information.

The system proposed is composed of two main components: the physical device, responsible for all operations, and an application on the user's computer, which provides a straightforward interface to users. The HSM's services can be accessed through a PKCS#11 API, which provides a common developer interface.

Several services are proposed and detailed next. A secure data exchange service, a key management solution, shared key agreement, qualified digital signatures, key importation and several tamper protection mechanisms. To access the services, the user must authenticate with a PIN number. This number can be changed after login.

A. Secure Data Exchange

The main goal of this solution is to provide confidentiality, integrity and authentication, to entities with identical devices. To grant these services, communicating entities must have previously agreed on a symmetric key. This key will be stored in the devices of both entities, and is never exposed to the outside.

Symmetric encryption schemes provide confidentiality to data, while MAC algorithms provide authentication. AEAD schemes, which authenticate and encrypt messages, such as AES-GCM, may be more efficient and are less likely to be misused, compared to combining separate encryption and authentication schemes. Unfortunately, devices such as the SmartFusion2 SoC, do not provide AEAD schemes. However, it provides separate encryption and authentication algorithms. Thus, the proposed solution combines an encryption and authentication scheme, in order to provide the necessary cryptography services. Specifically, AES with CBC encryption and HMAC with the SHA-256 hash function. Studies recommend combining a secure encryption and secure MAC, with the encrypt-then-MAC method [18]. This method encrypts the plaintext first, then generates the MAC from the generated ciphertext.

The proposed encryption protocol is pictured in Equation 1. The plaintext data is encrypted with an internal symmetric key and a randomly generated IV, with the board's TRNG. Next, a MAC is generated from the concatenated IV and ciphertext. If the IV can be modified by an attacker, the original plaintext cannot be fully recoverable. Therefore, it is important that the MAC is generated from both the ciphertext and IV, this way the receiver can detect if either information was altered. The output data is the concatenation of the IV, ciphertext and MAC.

$$E_{key}\{Data, IV\}, MAC_{key}\{IV + E_{key}\{Data, IV\}\} \quad (1)$$

The decryption protocol is pictured in Equation 2. The protocol follows the same process as the previous protocol, but in reverse order. First, a new MAC is generated from the received IV and ciphertext. Then, the computed and received MACs

are compared. If identical, the ciphertext is decrypted with the same internal symmetric key used for encryption and the received IV, to obtain the plaintext.

$$(MAC_{key}\{IV + Ciphertext\} == MAC) => \\ => E_{key}\{Ciphertext, IV\} => Data \quad (2)$$

When choosing key sizes, and taking into account the limited storage capacity of the board, a smaller, but still secure, key size is preferred. The AES 128 bit and 256 bit services guarantee 128 and 256 bit security respectively. HMAC with SHA-256 provides 256 bit security, with 256 bit keys. According to the NIST recommendations [19], algorithms which guarantee both 128 and 256 bit security, are expected to be secure until 2031 and beyond. If storage is extremely limited, AES with 128 bit keys is a secure and adequate option. However, with 256 bit security, the system will have a longer life expectancy. The key used for HMAC should be different from the one used in encryption, to ensure the best security practices, by not reusing the same key in different algorithms. So in practice, a key used for securing communications is split in two keys, one for encryption and one for authentication.

Considering the RAM is limited to a maximum of 80 KB and the device does not provide either a continuous encryption or authentication implementation, there is a limit to the data size which can be secured by the service. To overcome this limitation, the characteristics of the AES CBC mode can be taken advantage of. CBC mode encrypts data one 16 byte block at a time, using an IV. The IV of the first block is the randomly generated value from the TRNG. The IV of the subsequent blocks, is the previously computed ciphertext block. If the data is received in chunks, the IV of the first chunk is the randomly generated value, while the IV of the next chunk is the last ciphertext block of the previous chunk. This allows continuous encryption of data divided in chunks. To avoid padding and guarantee CBC's security, ciphertext stealing was implemented.

The total data length is not sent initially, instead, the length of each chunk is sent before the chunk. This allows for a more flexible system. User applications calling the PKCS#11 API, can send data to the device as it is needed, even if it does not have the complete data initially. The downside of this approach is the device does not know the amount of data it will encrypt. Therefore, the internal buffer must be managed, so it adds complexity to the implementation.

As previously introduced, the board's AES implementation is not resistant to side-channel attacks, such as DPA. This means attackers with physical access to the device can potentially read and compromise the keys stored internally. In order to mitigate this and build a more robust system, a 128 bit AES core implemented in the board's FPGA, resistant to side-channel attacks, was also tested with this service.

B. Key Management

The system is responsible for the storage, security and management of its keys. Thus, it is essential for the system to have a secure and flexible key management solution, which takes advantage of the device's secure storage.

The SmartFusion2 provides a secure storage service, the SRAM-PUF. It has 56 available key slots, where one key can be saved in each slot with a maximum of 512 bytes for a single key. The service uses private eNVM pages to store part of the key data, which are limited to 1000 writes for each page, for a predicted lifespan of 20 years. Thus, there is a limit for the key storage frequency in the PUF service. It should be used carefully, restraining how often it is written to, in order to preserve its lifespan.

So as to mitigate the limitation of the PUF storage, an alternate solution, in which the keys are stored in a non volatile memory was developed, as depicted in Figure 1. In

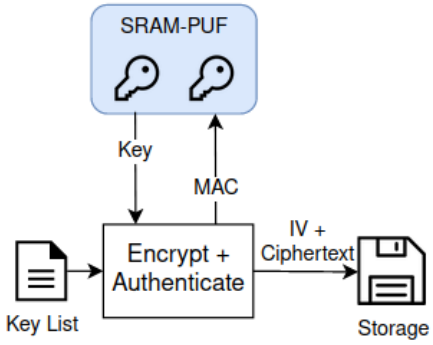

Fig. 1. Key management solution to store keys encrypted and authenticated in a non volatile memory using the SRAM-PUF secure storage

order to securely store the keys, they must be encrypted so as to hide their contents, and authenticated to detect any unauthorized modifications. To encrypt and authenticate keys, a symmetric key is necessary. This key is randomly generated in the device and stored in a dedicated PUF slot, only used for storing keys in memory. The ciphertext of the keys is stored in memory, along with the IV used for encryption. Both pieces of information are authenticated by generating a MAC, which is stored in another dedicated PUF slot. With this solution, a key can be accessed by generating the MAC of the stored data and comparing it to the one stored in the PUF slot. If they match, the keys are authenticated, and can be decrypted with the IV and dedicated PUF key.

While this solution still uses the PUF service, its usage is more measured. By using the PUF service, when one key is stored one slot is written to. In the case of the implemented solution, for each change of the key list stored in memory, only the MAC slot is written to. If keys are added one at a time, the amount of PUF slots written per key is identical for both options. However, multiple keys can be updated at once, if for example a list of keys is imported. In this case, multiple keys can be added or updated, with only one write to a PUF slot.

The amount of available key storage is not a problem, since the board allows for external storage devices to be connected, where keys can be stored, encrypted and authenticated. If

attackers get access to the key storage, the encrypted keys cannot be read, without the key protected in the SRAM-PUF.

C. Import Keys

To complement the key management solution, a key import operation was implemented, which imports a set of encrypted keys into the device. The service allows entities to receive a set of encrypted keys from another entity, forward the list to their device, which decrypts them with the secure data exchange service and stores them in the non volatile storage. The entity from which the keys are received is responsible for the distribution of keys among entities. So as to communicate with the trusted entity, all devices are delivered with a stored symmetric key for secure communications with the entity.

D. Key Generation

The goal of this service is to enable agreement of symmetric keys between entities with identical devices. This enables entities with no previously agreed keys, to securely communicate with each other, using internal keys stored in their devices. Two entities can agree on a symmetric key, using public key cryptography and the ECDH algorithm. Both entities must have a private and public key pair, and share both public keys with the other entity. Only the private key must remain a secret, the public key can be shared. Then, both entities can generate the same key, using their private key and the peer's public key. The SmartFusion2 SoC provides the necessary ECC scalar multiplication accelerator, to generate a shared key with the ECDH algorithm.

As mentioned, both entities must share their device's public key with the other entity. They must do it in a way, so that they can be sure the received public key is from the actual entity they wish to communicate with, and not an impersonator. This is usually achieved by a PKI, which is a trusted third party which stores, validates and distributes public keys. Instead of entities sharing public keys using an untrustworthy communication service, with no validation of the traded public keys, a PKI inspired system can be used. In this system, keys are exchanged using an intermediary, which is a special entity, trusted by both sides. Both entities should have previously agreed keys with the special entity, so they can securely send their public key. Thus, the special entity is responsible for distributing the public keys. Entities can trust the received keys belong to the correct entity.

E. Qualified Digital Signatures

Digital signatures provide non-repudiation to a piece of data. This prevents an entity from denying the authorship of a message. Qualified signatures are a special type of signatures where the private keys, which generated them, are stored inside a device, such as a HSM, and are never exposed to the outside. Therefore, the signature must be generated inside the HSM, and the device should support an algorithm for the generation of signatures, such as ECDSA. The private key, which will generate signatures, identifies the entity and is stored inside the device. In order for the device to be ready for the generation of signatures, the key must be randomly generated and subsequently stored in the device, before it is delivered to entities.

The device can generate signatures with the device's private ECC key, an implementation of the ECDSA algorithm and the SHA-256 hash function. The board is not equipped with a ECDSA implementation. The algorithm can be implemented, by combining the provided ECC security cores, TRNG, the SHA core, and a big numbers library, which must allow operations between numbers of at least 48 bytes.

F. Tamper Detection

An important part of a HSM based system is its physical protection measures against tampering and attacks. In order to defend the system against physical threats, the tamper detection, zeroization and secure boot functionalities of the SmartFusion2 SoC were taken advantage of.

Tamper attacks and possible attempts can be detected by the board. When these events occur, flags are asynchronously set, which warn the user from potential anomalies, errors and attacks. With this information, measures, such as zeroization, can be taken to protect the system. Zeroization is a process which erases all sensitive information from the device. This process can place the device in three possible states. It can be rendered permanently unusable, reset to its initial state or recoverable only with a key file supplied by Microsemi.

Another security measure of the SmartFusion2 board, is the computation of digests from the eNVM blocks and fabric configuration, which also holds the code. Every time the board is programmed or the configuration is changed, new digests are computed and stored in secure storage. On boot, the board computes the digests and compares them to the stored values. This allows the board to safeguard the integrity of its storage and configuration.

When the tamper attacks are flagged in the implemented system, the device does not accept any more PKCS#11 calls and zeroization is performed on the device, erasing all keys, configuration and data. Additionally, the secure boot checks are turned on. Zeroization was configured to reset the device to its initial state from fabric. The user also has the option to manually trigger zeroization, by pressing a button on the board. These measures can have a denial-of-service effect on the system, but are a trade-off deemed necessary, in order to avoid successful attacks and potential leaks of sensitive information.

V. Implementation

The system was implemented on a SmartFusion2 SoC Security Evaluation Kit, version M2S090TS from Microsemi. It combines an ARM-Cortex M3, a non-volatile memory (eNVM), FPGA and several cryptographic accelerators: AES-256, SHA-256, HMAC, SRAM-PUF secure storage and ECC multiplication and addition on the NIST P-384 curve. By default, the RAM is 64 KB. For each byte of RAM, there are 2 bits for error detection and correction, a total of 16 KB, which mitigates Single Event Upsets (SEU). It corrects

1 bit errors and detects up to 2 bit errors. The board was configured with this setting disabled in Libero software, to free the additional 16 KB of memory, for a total of 80 KB. The computer and device are connected through a serial connection using the available UART controller. All the services implemented on the Smartfusion2 are accessed through the developed PKCS#11 API. It was implemented on Windows 10 with C/C++ for the open source MinGW compiler.

The device was configured to detect tamper attempts. When the tamper attacks are flagged in the implemented system, the device does not accept any more PKCS#11 calls and zeroization is performed on the device, erasing all keys, configuration and data. Zeroization was configured to reset the device to its initial state from fabric. The user also has the option to manually trigger zeroization, by pressing a button on the board. Additionally, the secure boot checks are turned on.

Regarding the secure data service, with all the code and drivers needed for the implementation, only around 36 KB of space for data buffers is available. This means the device can secure up to 36 KB of data, using the same buffer for input and output. This was overcome by the proposed continuous secure data exchange service, which was implemented using the device's AES SoC accelerator in CBC mode and a HMAC software library [20].

To implement ECDSA, a big numbers library was included [21]. Qualified digital signatures generation was implemented, without verification. The library takes up around 54 KB of RAM space. Thus, this functionality only works with a limited buffer size of around 1.5 KB, if all other implemented features are disabled. Future work should revise this functionality, by evaluating existing lightweight libraries, which provide the necessary features, or even a custom implementation.

A. TCP Channel

With the secure data exchange service, entities can securely trade messages using an offline service such as e-mail or an online chat service. In order to truly establish a secure communication channel, and demonstrate the usage of the common developer interface, the secure data service was used to encrypt and authenticate a TCP connection. A TCP implementation using Windows sockets is used to exchange data through a channel, between a client and server, running on the same computer. The library was altered, to call the PKCS#11 API of the SmartFusion2 system, to encrypt the plaintext data in each TCP packet, before it is sent. Likewise, when a packet is received, the decryption API is called, in order to authenticate and retrieve the plaintext.

Before a secure TCP connection is established, both entities must agree on a symmetric session key, which will be used to encrypt and authenticate the connection. This is achieved by using the previously described key generation service with ECDH. After each side trades their public keys, they can compute the same symmetric key. In order to emulate two communicating entities using a similar system, one side is running a local implementation of the same cryptographic services of the proposed system, with the mbedTLS 2.26.0 library.

VI. Results and Analysis

This section presents the evaluation of the SmartFusion2 board and the developed prototype, its services, their performance, capabilities and limitations. The tests were all performed on a Windows 10 computer, running the user software which calls the implemented PKCS#11 API. Since the board does not provide a clock and API to measure elapsed time, the time is measured on the computer between PKCS#11 API calls. The elapsed time was measured with microseconds precision.

Each component was tested with at least 30 repetitions, in order to achieve a variance below 1%. Aside from the communications channel, the rest of the services were tested by minimizing the communications overhead. This way, the isolated service performance in the device can be more accurately assessed.

After measuring the results for multiple services, it was observed that most followed a close to perfect linear evolution, in function of the processed data size. Thus, their performance can be modeled with a formula composed of two different components $T_{Total} = T_{Constant} + T_{Data} * KB$, a constant value independent of processed data, and a factor dependent on the processed data size. These values were calculated and are presented next to the median average percentage error, so as to assess the accuracy of the models.

In order to assess the communication channel performance, and its impact on the system, the average time to transmit data was measured. The experimental throughput stabilized around 11 KB/s, as data size increased. The performance of the data channel, in milliseconds, can be modelled by a linear equation, with values $T_{total} = 7.281 + 88.638 * KB$, and a median average percentage error of 0.92 %.

A. SmartFusion2 Services

All the security accelerators of the SmartFusion2 SoC were tested. This includes the TRNG, AES SoC accelerator, SHA, HMAC, KeyTree and ECC scalar multiplication and point addition(Add.) services. Additionally, a side-channel resistant 128 bit AES core implemented in the FPGA was also tested. The isolated service throughput results are presented in Figure 2.

The AES SoC service was tested with all possible variations. Namely, with 128 bit and 256 bit keys, with all four available modes and with encryption and decryption. Only one result is shown, since there was no variation among them. The AES mode, key size or encryption/decryption operation does not impact the performance. Therefore, there is no performance advantage in choosing CTR mode over CBC mode, or any other mode.

Most services throughput, shown in KB/ms, increases and eventually stabilizes at a specific value. SHA stabilizes at around 1.2 KB/ms, and both SoC AES and HMAC at around 0.1 KB/ms. The AES core implemented on the FPGA is

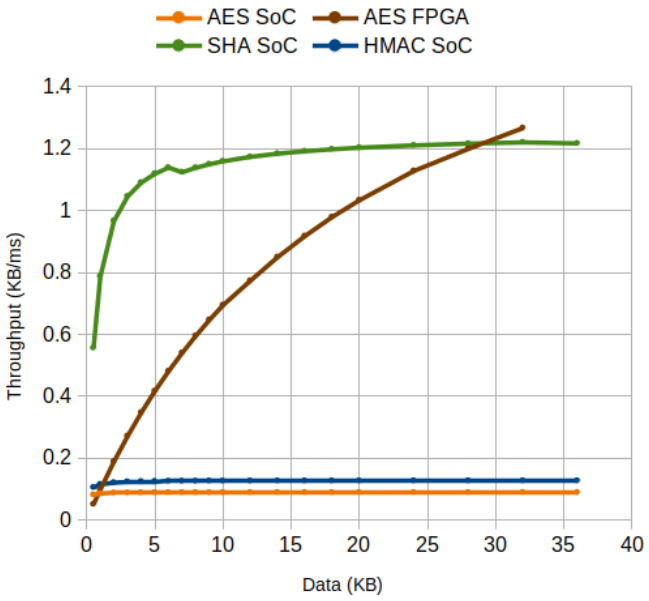

Fig. 2. Security services throughput evolution

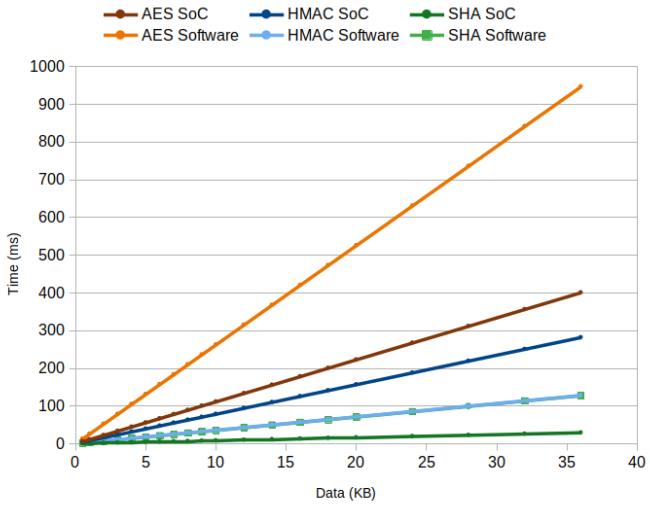

Fig. 3. Performance comparison of the board's cores and a software implementation

significantly faster than the SoC AES core, with performance comparable to the SHA accelerator. Due to its side channel mitigations and performance advantage, it is a significantly better choice than the AES SoC core.

All data dependent services followed a near perfect linear evolution, in function of the processed data size, as presented in Table I. The TRNG service was tested by generating 16 random bytes, up to the maximum allowed of 128 bytes. As expected, the performance barely increases with the data size. The error percentages are all below 1%, except for the TRNG service, proving these models accurately predict the performance of the services.

Time (ms)	AES SoC	SHA	HMAC	TRNG	Add.	ECDH
Constant	0.489	0.498	0.783	0.368	7.204	545.381
Data (KB)	11.124	0.807	7.815	0.007	-	-
MAE	0.12%	0.84%	0.13%	2.29%	-	-

TABLE I
SMARTFUSION2 ACCELERATORS PROCESSING TIME PERFORMANCE VALUES ACCORDING TO A LINEAR MODEL

Core/Software Comparison: The SHA and HMAC performance difference results were enigmatic. The HMAC data dependent portion, 7.815 ms, is nearly 10 times higher than the SHA value, 0.807 ms. This means HMAC's time performance degrades nearly 10 times faster than the SHA performance. This is a surprising result, since the HMAC algorithm is composed of two hash computations and uses SHA-256, so the results are expected to be closer. The software implementation, included in Section V, of HMAC, SHA and AES were tested for comparison with the SmartFusion2 SoC services. The library used for HMAC and SHA was [20], and for AES [22].

Analyzing the time performance results in Figure 3, the SHA and HMAC software results are almost identical, the HMAC is a slightly worse performer. Compared to the software results, the SHA core is significantly faster, and deteriorates very slowly as data size increases. The opposite happens for the HMAC core. It is convincingly a worst performer, compared to both HMAC and SHA software implementations. This is an ambiguous result, as there is no clear reason for the HMAC core performance degradation, compared to the SHA core and the software implementations. One could assume it is caused by possible DPA mitigations, but it would still not explain the meaningful discrepancy compared to the SHA core, which also includes these mitigations. The AES software implementation was tested with encryption in CBC mode and a 256 bit key. As expected, it performs worse than the core service. CTR mode with the same configuration was also tested, and has very similar performance to CBC.

Memory Performance: The read and write performance of the different device's memories, along with the PUF service were tested. Both eSRAM and eNVM memories were tested from 0.5 KB to 36 KB. The PUF performance was tested from 16 bytes, up to its limit of 512 bytes.

The linear model values were calculated from the results, and are presented in Table II.

Time (ms)	RAM(W)	NVM(R)	NVM(W)	PUF(R)	PUF(W)
Constant	0.012	0.01	8.03	128.49	747.84
Data (KB)	0.014	0.02	298.10	0.006	0.008
MAE	2.76%	3.76%	0.31%	0.17%	0.06%

TABLE II
SMARTFUSION2 READ AND WRITE PERFORMANCE VALUES ACCORDING TO A LINEAR MODEL

The results show that the PUF service barely fluctuates with the data size, since a slot only goes up to 512 bytes. It has an almost constant read and write performance. Regarding the eNVM, its write performance deteriorates significantly

with increasing data sizes. The RAM write performance is comparable to the eNVM read performance.

B. Implemented Services

This section presents the performance results of the implemented services from Section V. Each service's performance depends on the used accelerator's, memory access and implemented logic.

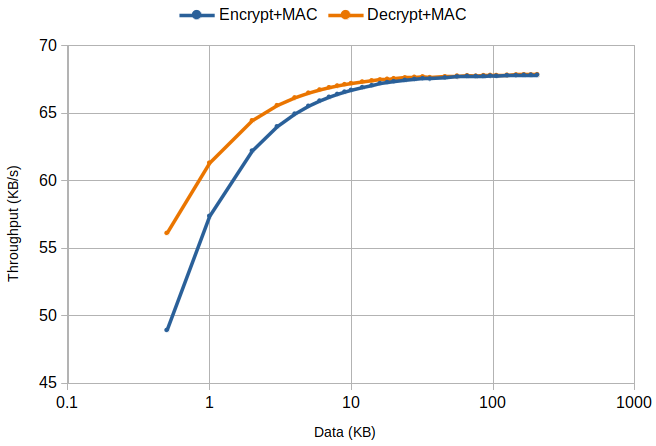

Fig. 4. Continuous encryption and decryption throughput evolution for increasing input data sizes

Figure 4 plots the throughput (KB/s) values for the continuous encryption and decryption services. Throughput steadily increases and stabilizes after 10 KB, at around 68 KB/s, for both services.

Similarly to the previous tests, the time performance results evolve linearly. The obtained values are presented in Table III. By analysing the calculated models we can detect a difference

Time (ms)	Encryption	Decryption	ECDSA	Key Gen.
Constant	2.556	1.557	629.434	578.092
Data (KB)	14.730	14.729	-	-
MAE	0.18%	0.04%	-	-

TABLE III
IMPLEMENTED SERVICES PROCESSING TIME PERFORMANCE VALUES ACCORDING TO A LINEAR MODEL

in the constant component of the encryption service compared to decryption. This is due to the random IV generation with the board's TRNG on the encryption service. Key generation with ECDH performs at a constant time of 578.092 ms, and ECDSA at 629.434 ms. Scalar multiplication has a big impact on the performance for both ECDH and ECDSA. The median average percentage error for the continuous encryption/decryption models is below 0.19%, so the models almost perfectly represent its performance.

As mentioned previously, the continuous encryption/decryption with MAC implementation uses a 36 KB buffer. Thus, the service can receive chunks of up to 36 KB at a time. Since the service is continuous and can encrypt and authenticate a limitless amount of data, the buffer does not necessarily need to be the maximum possible value. The service might have comparable performance with a smaller buffer, which frees up memory space for further implementation code.

In order to understand the impact of the buffer size on performance, the previous test on the encryption and MAC service was repeated, with varying buffer sizes, from 0.1 KB up to 35 KB. All tests revealed the same linear performance behaviour as the previous test with a 36 KB buffer. In order to compare and understand the service performance for each buffer size, the maximum, minimum and average throughput are pictured in Figure 5. The worst performance, and therefore

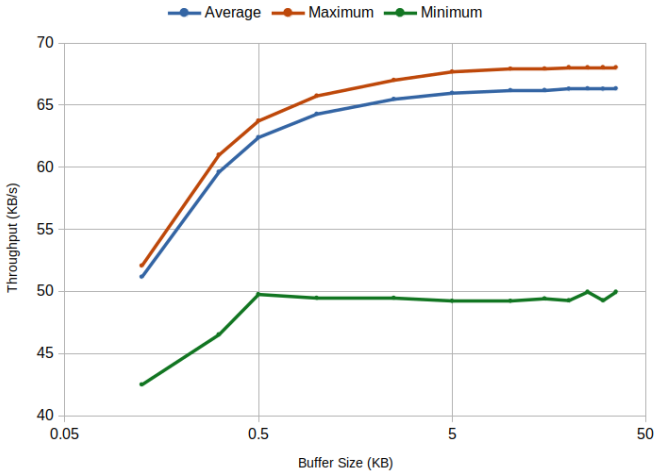

Fig. 5. Continuous encryption and MAC throughput evolution with increasing buffer sizes

lower throughput, is achieved with the lowest data size of 0.5 KB. The throughput eventually stabilizes around its maximum value, with data sizes higher than 20 KB. A smaller range from the minimum to maximum throughput, for a particular buffer size, indicates the throughput increases slower, as data size increases.

Analysing the plot, the average throughput significantly increases from 0,1 KB up until 5 KB, after which, the average throughput stabilizes around 66 KB/s and the maximum at 68 KB/s. If the goal is to maximize throughput, there is no advantage in using a buffer bigger than 10 KB. Even a 5 KB buffer has almost identical performance.

The minimum throughput spikes at 49.5 KB/s with a 0.5 KB buffer, then is relatively constant. As discussed before, the minimum throughput is achieved at the lowest processed data size of 0.5 KB. Thus, by analysing the graphic, the minimum throughput increases until the buffer size (0.5 KB) is equal to the amount of data being processed (0.5 KB). When this happens, the complete data can be sent in only 1 chunk. After that, there is no benefit in having a bigger buffer size, since the lowest amount of processed data is still 0.5 KB. Therefore, if the amount of data which will be processed is known beforehand, the system can be configured with a buffer

of equal size, so as to maximize performance and available RAM space.

The SmartFusion2 SoC is a portable board, with a robust set of security services. Compared to the existing HSMs on the market, this device is one of the cheapest, so it is adequate for distribution among several users. Market HSMs go from 650 up to $39,000 [23], [24]. A M2S090TS SmartFusion2 evaluation kit is priced at 384 [25].

The proposed system provides a fully functional HSM, comparable to the state of the art offerings. It provides confidentiality, authentication and non-repudiation, with tamper detection and protection against attacks, as well as a PUF based secure storage solution with improved lifespan and storage capacity. Like the state of the art HSMs, it provides a common developer interface with PKCS#11.

VII. Conclusion

In this paper a low cost and versatile secure message exchange system is proposed and implemented on a SmartFusion2 SoC. The solution takes advantage of the device's security features, cryptography algorithms and PUF based secure storage. The developed system provides continuous encryption and authentication to communications, shared key agreement, qualified digital signatures and a key import service. It also provides a PUF based key storage solution, which improves its lifespan, storage capacity and flexibility. The board is protected with tamper detection mechanisms, startup digest checks and zeroization.

This work also contributes with an extensive characterization of the SmartFusion2 device. It studied each security service advantage and possible trade offs. Furthermore, it models the performance of every service, providing a useful prediction of the system's behaviour.

References

[1] C. Lesjak, H. Bock, D. Hein, and M. Maritsch, "Hardware-secured and transparent multi-stakeholder data exchange for industrial iot," in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. IEEE, 2016, pp. 706–713.

[2] J. Seol, S. Jin, D. Lee, J. Huh, and S. Maeng, "A trusted iaas environment with hardware security module," *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 343–356, 2015.

[3] M. Wolf and T. Gendrullis, "Design, implementation, and evaluation of a vehicular hardware security module," in *International Conference on Information Security and Cryptology*. Springer, 2011, pp. 302–318.

[4] M. Canim, M. Kantarcioglu, and B. Malin, "Secure management of biomedical data with cryptographic hardware," *IEEE Transactions on Information Technology in Biomedicine*, vol. 16, no. 1, pp. 166–175, 2011.

[5] D. Mahto, D. A. Khan, and D. K. Yadav, "Security analysis of elliptic curve cryptography and rsa," in *Proceedings of the world congress on engineering*, vol. 1, 2016, pp. 419–422.

[6] K. Gupta and S. Silakari, "Ecc over rsa for asymmetric encryption: A review," *International Journal of Computer Science Issues (IJCSI)*, vol. 8, no. 3, p. 370, 2011.

[7] P. FIPS, "140-2 federal information processing standards publication–security requirements for cryptographic modules," *Issued May*, vol. 25, 2001.

[8] M. K. Bond, "Understanding security apis," Ph.D. dissertation, University of Cambridge, 2004.

[9] D. C. Wherry, "Secure your public key infrastructure with hardware security modules," SANS Institute, Tech. Rep, Tech. Rep., 2003.

[10] M. Lorch, J. Basney, and D. Kafura, "A hardware-secured credential repository for grid pkis," in *IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004.* IEEE, 2004, pp. 640–647.

[11] T. Rossler, H. Leitold, and R. Posch, "E-voting: A scalable approach using xml and hardware security modules," in *2005 IEEE International Conference on e-Technology, e-Commerce and e-Service*. IEEE, 2005, pp. 480–485.

[12] A. Baldwin and S. Shiu, "Hardware encapsulation of security services," in *European Symposium on Research in Computer Security*. Springer, 2003, pp. 201–216.

[13] M. C. Mont, A. Baldwin, and J. Pato, "Secure hardware-based distributed authorisation underpinning a web service framework," *HP Laboratories Bristol*, 2003.

[14] J. E. Martina, T. C. S. de Souza, and R. F. Custodio, "Openhsm: An open key life cycle protocol for public key infrastructures hardware security modules," in *European Public Key Infrastructure Workshop*. Springer, 2007, pp. 220–235.

[15] O. Kehret, A. Walz, and A. Sikora, "Integration of hardware security modules into a deeply embedded tls stack," *International Journal of Computing*, vol. 15, no. 1, pp. 22–30, 2016.

[16] J. Ivarsson, A. Nilsson, and A. Certezza, "A review of hardware security modules fall 2010," Technical report, Certezza, 2010, Tech. Rep., 2010.

[17] Yubico, "Yubihsm 2," last visited 2021-05-27. [Online]. Available: https://www.yubico.com/pt/product/yubihsm-2-hardware-security-module/

[18] H. Krawczyk, "The order of encryption and authentication for protecting communications (or: How secure is ssl?)," in *Annual International Cryptology Conference*. Springer, 2001, pp. 310–331.

[19] E. Barker, "Nist special publication 800-57 part 1, revision 5," *NIST, Tech. Rep*, 2020.

[20] O. Gay, "Software implementation in c of the fips 198 keyed-hash message authentication code hmac for sha2," http://ouah.org/ogay/hmac, 2013, last visited 2021-05-27.

[21] R. Benadjila, A. Ebalard, and J.-P. Flori, "libecc project," https://github.com/anssi-fr/libecc, 2017, last visited 2021-05-27.

[22] C. Heath and R. Misoczki, "Tinycrypt cryptographic library," https://github.com/intel/tinycrypt/, 2017, last visited 2021-05-27.

[23] L. Harbaugh, "Thales nshield connect offers enterprise-class key management," *Network World*, 2009, last visited 2021-02-23. [Online]. Available: http://www.networkworld.com/article/2246758/security/thales-nshield-connect-offers-enterprise-class-key-management.html

[24] J. Schlyter, "Hardware security modules," last visited 2021-02-23. [Online]. Available: https://internetstiftelsen.se/docs/hsm-20090529.pdf

[25] M2s090ts-eval-kit pricing. Last visited 2021-02-02. [Online]. Available: https://eu.mouser.com/ProductDetail/Microchip-Microsemi/M2S090TS-EVAL-KIT/?qs=HNBw3F7vE2zzRkt03XBdWg==