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

Alexandre Valente Rodrigues

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Ricardo Jorge Fernandes Chaves

Examination Committee

May 2021

Acknowledgments

I would like to thank my parents for always supporting me, encouraging me, and providing everything needed to succeed, throughout all my life. Without them this work and degree would not be possible. I also want to thank my grandma for always being a friend and supporter, even through hard times.

I would also like to extend my gratitude to my dissertation supervisor Prof. Ricardo Chaves for his support, knowledge and constant willingness to help, and push me to do better. His insights and our discussions have made me a better student. This experience is a valuable tool I will carry into my future as a professional.

Last but not least, to all my significant other, long time friends and colleagues that always helped me, and were there for me when I needed through tougher periods of my life. I will always be there for you. Thank you.

Abstract

Nowadays, computers are exposed to a wide range of attacks. Individuals with high responsibility jobs, such as government officials and top company executives are high profile targets. Successful attacks can compromise communications and leak sensitive information. Passwords and cryptographic keys are essential to the security of communications. The goal of this work was to isolate the user's computer from the system responsible for storage and management of keys. This was achieved by introducing a low cost, secure and portable Hardware Security Module (HSM). This work studied and evaluated the services of the SmartFusion2 System-on-Chip (SoC), as a HSM. The SmartFusion2 provides a robust set of security services: symmetric key encryption, a hashing function, authentication, public-key cryptography primitives, a true random number generator, tamper detection and zeroization. The performance and limitations of these services were evaluated and modelled. A proof of concept was also developed on the device, focused on providing authentication and confidentiality to communications. The device continuously encrypts and authenticates communications as an intermediary between the owner, and other communicating users. A key management service is also provided, which takes advantage of the secure storage, given the device limitations, and allowing for regular key updates. The developed system also provides shared key generation. 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.

Keywords

Hardware Security Module; Secure Communications; SmartFusion2 SoC; Confidentiality; Authentication.

Resumo

Hoje em dia, computadores estão expostos a uma variedade de ataques. Indivíduos com cargos de grande responsabilidade, como membros do governo ou executivos de empresas, são alvos valiosos. Ataques bem sucedidos podem comprometer a segurança das comunicações e levar a fugas de dados. Chaves criptográficas são essenciais para a segurança das comunicações, e são normalmente guardadas no computador do utilizador. O objetivo deste trabalho é separar o computador do sistema responsável pelo armazenamento de chaves. Este objetivo foi alcançado utilizando um módulo de hardware (HSM), seguro, portátil e de baixo custo. Este trabalho estudou e analisou os serviços da placa SmartFusion2 SoC, como o HSM deste sistema. A placa providencia uma gama robusta de serviços de segurança: cifra simétrica, uma função de hash, autenticação, chaves assimétricas, um gerador de números aleatórios, proteções físicas contra ataques e *zeroization*. A performance e limitações destes serviços foi avaliada e modelada. Além disso, um protótipo foi desenvolvido no dispositivo, que protege as comunicações. O dispositivo cifra e autentica as comunicações continuamente, e funciona como um intermediário entre o dono e outros utilizadores. Um serviço de gestão de chaves também é providenciado, que tira partido da memória protegida da placa, e permite a atualização de chaves regularmente. O sistema também permite a geração de chaves partilhadas. A placa está bem equipada para ser utilizada como HSM, no entanto foram identificadas algumas limitações nos seus serviços, memória e desempenho. Este trabalho fornece a análise essencial e um sistema base para a utilização da SmartFusion2 como HSM.

Palavras Chave

Módulo de Segurança; Comunicações Seguras; SmartFusion2 SoC; Confidencialidade; Autenticação.

Contents

List of Figures

List of Tables

Acronyms

AES	Advanced Encryption Standard
API	Application Programming Interface
AD	Associated Data
AEAD	Authenticated Encryption with Associated Data
CBC	Cipher Block Chaining
CBC-MAC	Cipher Block Chaining Message Authentication Code
CLI	Command line Interface
CTR	Counter
CCM	Counter with CBC-MAC Mode
CPU	Central Processing Unit
DH	Diffie-Hellman
DPA	Differential Power Analysis
ECB	Electronic Codebook
ECC	Elliptic-Curve Cryptography
ECDH	Elliptic-Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
FPGA	Field-Programmable Gate Array
GCM	Galois/Counter Mode
HMAC	Hash-based Message Authentication Code

HSM	Hardware Security Module
IDE	Integrated Development Environment
IPsec	Internet Protocol Security
IV	Initialization Vector
MAC	Message Authentication Code
NIST	National Institute of Standards and Technology
NRBG	Non-Deterministic Random Bit Generator
NVM	Non-Volatile Memory
OFB	Output Feedback
PGP	Pretty Good Privacy
PIN	Personal Identification Number
PKCS	Public-Key Cryptography Standards
PKI	Public-Key Infrastructure
RAM	Random-Access Memory
RSA	Rivest-Shamir-Adleman
SEU	Single-Event Upset
SHA	Secure Hash Algorithms
SoC	System-on-Chip
SRAM	Static Random-Access Memory
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TRNG	True Random Number Generator
TPM	Trusted Platform Module
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus

1 Introduction

Contents

In the modern world, most people access a computer for their everyday tasks such as, web browsing, chatting, entertainment, among many others. There is no limit to what you can achieve with the Internet, using just a computer. For this reason, computers have a wide range of attacks potentially exploitable by hackers, by taking advantage of software vulnerabilities or user mistakes. Suffering an attack on a personal computer can be highly damaging, as it can carry severe consequences for companies and countries. This is of great concern to people with high responsibilities, who may deal with sensitive information, such as government officials, military or top level company executives. These people have a higher profile, and are more likely to be targeted by hackers.

1.1 Problem

New attacks targeting computers are discovered daily. They can originate from zero-day vulnerabilities, phishing scams and many other sources. Attackers have endless opportunities, so it is extremely difficult to predict and protect against all threats. Communications security depends on the cryptographic keys and passwords used. These are usually stored, along with other sensitive information, in the user's computer. Instead of storing this information in the user's computer, a more hardened and secure solution is to separate the computer used by the user for communications, and the device responsible for managing keys and passwords. This device would bear the burden of securing communications and storing its sensitive data. The goal is to add another layer of security, to make it difficult to compromise communications, even if the user's computer is attacked. A secure and independent solution is needed to establish secure channels of communication, store keys and perform critical operations. A possible approach is the utilization of a personal and physical device, responsible for the storage of digital keys and all security critical operations. These devices need to be highly secure, independent and support a robust set of security services.

1.2 Requirements

In order to address the problem using the discussed approach, the implemented solution has several requirements to allow secure communications between multiple entities, with a security focused physical device. The solution developed on the device should perform all critical operations to the security of communications, as well as store all relevant passwords and keys which safeguard its security. Another requirement for this system is for it to be easy-to-use for the average user, with no technological expertise. The system must be relatively efficient to allow communications in an acceptable time frame and also be relatively low cost.

1.3 Document Structure

This document is composed of seven chapters, including the introduction. The second chapter covers the technical background needed to understand the solution and state of the art. Chapter 3 defines the existing problem, its devices, the necessary requirements for the solution, and possible use case scenarios. Chapter 4 covers the developed solution architecture, including its components, communication protocol and services. Chapter 5 presents the implementation details more specific to the hardware device. This includes the tools used, the implemented services, system details and potential feature trade offs. Chapter 6 evaluates the system's performance, capabilities and fulfilled requirements. The last chapter summarizes the developed prototype, its evaluation, and provides guidelines for future work.

2

Background and Related Work

Contents

This chapter details the technical concepts required to understand the problem, the proposed solution and the rationale behind it. It starts by providing an overview of cryptographic services, primitives and protocols. Then, it presents several general purpose computing systems which offer cryptographic services, and ends by presenting other relevant components.

2.1 Cryptography and Other Concepts

There are four important cryptographic services relevant for 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. To guarantee these services, two types of key infrastructure exist: symmetric and asymmetric. Symmetric keys are shared by two or more communicating parties. The same key is used to encrypt and decrypt data, or for other services. The keys are generally smaller and the operations faster than with asymmetric keys. Asymmetric keys constitute a pair for each side, a private and a public key. The private key is personal and should never be shared. The public key may be shared widely.

There are two types of ciphers: stream and block ciphers. Stream ciphers usually process 1 bit of data at a time. Stream ciphers are usually faster than block ciphers, have lower memory requirements and thus are more suitable to embedded devices with limited memory. Block ciphers encrypt fixed-length groups of bits, called blocks. They have a higher memory usage, in order to keep the blocks in memory. The processed data needs to be padded to a multiple of the block size. This can introduce problems if not done correctly. Another caveat of block ciphers is they are more susceptible to noise in transmissions. If a bit is flipped in a stream cipher, only the corresponding bit is affected, while with a block cipher, more than 1 bit can be affected.

2.1.1 Hash Functions

A cryptography hash function generates a fixed dimension value (digest) based on input texts, such as messages or files. Secure hashes provide message integrity by comparing digests, calculated before and after transmission, to determine if the message was modified. The same message always generates the same digest. To achieve this, hash functions must have several properties. The same input value always results in the same hash. Very different outputs must be generated from very similar inputs. It should be hard to find two messages which generate the same hash and it should be hard to find an input that produces a given hash. Popular and recommended hash functions include Secure Hash Algorithms (SHA)-2 and the newer SHA-3 [1]. Both versions have variants. For example, the SHA-256 function produces a 256 bit digest and SHA-512 a 512 bit digest.

2.1.2 Symmetric Encryption

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. It has multiple block and stream cipher modes, which offer confidentiality. It uses 128-bit blocks and keys can have 128, 192 or 256 bits. Some relevant AES encryption modes are presented next.

Electronic Codebook (ECB) is the simplest mode. It is a block cipher and works by encrypting each block with the symmetric key. If the same key is used for equal plaintext blocks, the result will always be the same. For this reason, patterns are easily seen and the mode is considered insecure.

Cipher Block Chaining (CBC) is another block cipher mode. It combines the first block of plaintext and an Initialization Vector (IV) with the XOR operator and encrypts the result. For the subsequent blocks, the previous ciphertext is used instead of the IV. The message needs to be padded to a multiple of the block size. If done incorrectly, the mode is vulnerable to padding oracle attacks [2]. Implementing ciphertext stealing, which avoids padding, is recommended for the security of CBC [3].

Output Feedback (OFB) mode repeatedly encrypts the IV for each block, xoring the result with the plaintext block. The encryption and decryption processes are exactly the same. The block cipher is only used in the encryption direction, which means the message does not need to be padded. Therefore, it is a stream cipher.

Counter (CTR) mode concatenates an IV with a counter beginning at 0. This sequence is encrypted and applied to the xor operation with the plaintext block. For each block the sequence is incremented by 1. This mode is comparable to OFB, as it is also a stream cipher and the encryption operation is exactly the same as the decryption.

The IV needs to be sent along with the ciphertext to the receiver, or the receiver will not be able to retrieve the entire message. There is no need for the IV to be kept secret. CBC, OFB and CTR modes are proved secure, assuming the IV is random and unique, meaning it is only used once for each key and message. Regarding performance, [4] states CBC is slower than CTR mode, and OFB is even slower. When efficiency characteristics matter, nothing comes close to CTR: it has better performance characteristics, compared to CBC and OFB.

All these cipher modes are malleable, meaning an attacker can modify a ciphertext C, to create ciphertext C' which will decrypt to plaintext P' that is similar to the original plaintext. Malleability is connected to message integrity. This is not considered a relevant weakness since these modes are only designed to offer confidentiality. If integrity is needed, one of these modes should be paired with a Message Authentication Code (MAC), or instead use a dedicated authenticated-encryption mode like Counter with CBC-MAC Mode (CCM) or Galois/Counter Mode (GCM) (discussed in Section 2.1.4), which guarantee both confidentiality and integrity.

2.1.3 Message Authentication Code

MAC is a value, also called tag, used for authenticating a message. A MAC algorithm generates a tag from a message and symmetric key. Unlike digital signatures, MAC does not offer non-repudiation since it uses a symmetric key, which is shared among all users. Anyone in possession of the key can generate a MAC for a message, as well as, verify a MAC generated with the same key. On the contrary, digital signatures utilize the private key of an asymmetric pair to generate, and public key to verify. Several techniques exist to construct a MAC. One is **Cipher Block Chaining Message Authentication Code (CBC-MAC)**, which utilizes the CBC block cipher to encrypt data. A chain of blocks is generated, and the last block is the tag. CBC-MAC also has similar caveats to CBC, it is only secure for fixed-length messages [4] and different keys have to be used for CBC encryption and tag generation.

Hash-based Message Authentication Code (HMAC) is different, it uses a cryptographic hash function, such as SHA-2, and a symmetric secret key to construct a tag. HMAC is secure, as long as the underlying hash function used is secure. Therefore SHA-2 is a good option. HMAC does not have the security problems of CBC-MAC. It is a popular and well-designed construction, but it is not the most efficient approach [4].

2.1.4 Authenticated Encryption

Authenticated Encryption with Associated Data (AEAD) schemes assure confidentiality and authenticity using symmetric keys. They may be more efficient than combining separate encryption and authentication techniques, such as the ones discussed in earlier sections, and are less likely to be used incorrectly. AEAD schemes also allow associated data to be included in the message, which is authenticated but not encrypted. This feature is useful for network packets. The header is visible but is authenticated, while the payload is encrypted and authenticated. AES has several of these schemes.

CCM is an AEAD mode that combines CBC-MAC for authentication with CTR for confidentiality. CCM uses a MAC-then-Encrypt approach. First, the MAC is computed from the message. Then the message and the tag are encrypted with CTR mode. Due to performing two encryption operations, CBC-MAC and then CTR, the mode is not as efficient compared to others such as GCM, which only performs one encryption operation. It is not an online mode, meaning it needs to know the message and Associated Data (AD) length beforehand. Therefore, AD cannot be preprocessed. Despite being a slower mode, it is secure and widely supported. It is included in Internet Protocol Security (IPsec), Transport Layer Security (TLS) and Bluetooth low energy.

GCM utilizes an encrypt-then-MAC approach. It first encrypts with CTR mode, then uses Galois mode of authentication to generate the tag. The Galois field multiplication supports parallel computation, making this mode faster than CCM. Beyond being parallelizable, it is online and the AD can be preprocessed. For security reasons, authentication tags should be at least 96 bits, even though the mode allows smaller tags. One limitation of GCM is, it can encrypt a maximum of 64 gigabytes of plaintext. Security analysis of several modes decisively states that GCM in hardware is unsurpassed by any authenticated-encryption scheme [4].

2.1.5 Asymmetric Encryption

Asymmetric cryptography uses public and private keys. It is commonly used to provide confidentiality, data integrity, authentication and non-repudiation. The private key must always remain secure with the owner. Public keys may be distributed. Encrypting a message with the public key provides confidentiality, since only the owner who possesses the private key can decipher the message. On the other hand, private key encryption provides authentication, since only the owner is in possession of the private key. These two different concepts can be combined to provide confidentiality, authentication and non-repudiation to a message, through digital signatures.

Compared to symmetric keys, asymmetric keys are less risky to distribute, as the public key can be viewed by anyone. However, there is the problem of validating public keys, which consists of guaranteeing a public key is owned by the correct identity. Once two parties have traded public keys, asymmetric and symmetric keys can be combined in a hybrid encryption scheme. It takes advantage of the faster symmetric encryption to cipher the data, and the asymmetric encryption to encrypt the symmetric key, and provide authentication. Alternatively, it can be used to share symmetric keys for usage with an authenticated-encryption scheme.

There are two popular algorithms for public-key encryption, Rivest-Shamir-Adleman (RSA) and Elliptic-Curve Cryptography (ECC) [5]. RSA has been used for decades, is well established and widely used. It is based on the difficulty of factoring the product of two large prime numbers. ECC is a more recent algorithm, based on the Elliptic Curve Discrete Logarithm Problem. The main advantage of ECC is it offers the same level of security, with a smaller key size. According to Gupta & Silakari, 2011 [6], a 160-bit ECC private key has similar security to a 1024-bit RSA key. ECC operations are potentially faster with smaller key sizes, so this makes elliptic curve based schemes more suited for less powerful and memory constrained devices [7]. With the threat of quantum computers, both ECC and RSA could become obsolete in the future, as they are vulnerable to brute force attacks from such devices.

2.1.6 Diffie-Hellman

The Diffie-Hellman (DH) key exchange algorithm allows two parties to agree on a shared secret, which can be used to derive a symmetric key. Both parties compute the secret from publicly exchanged integers, and private integers. Attackers listening on the exchanged public integers cannot compute the same secret, since both sides' private integers are never shared.

Elliptic-Curve Diffie-Hellman (ECDH) key exchange is similar, it computes the shared secret from ECC private and public keys, instead of integers. Incorporating ECC keys provides the same level of security compared with integers, but with a smaller bit size [8].

2.1.7 Digital Signatures

Signatures are a standard scheme for authenticating digital messages and ensuring the signer cannot repudiate the signature. A digital signature is generated by combining asymmetric cryptography and a hash function. A digital signature is generated by first computing a hash of the message, then signing the hash with the author's private key. The message is not directly signed, since public-key encryption is slow and messages are most likely bigger than the hash of a message, which has a fixed size. Third parties can validate the signature using the author's public key. Only the author, in possession of their private key, could have generated the signature. Digital signatures are a digital version of handwritten signatures [9], commonly used anywhere forgery detection is essential, for instance in financial transactions or software distribution. A popular digital signatures algorithm with ECC keys is Elliptic Curve Digital Signature Algorithm (ECDSA).

Qualified signatures are a special type of signatures where the private keys are generated and stored inside a device, such as a Smart Card, and are never exposed to the outside. For the owner to sign a document, the Smart Card is needed (something owned) and a Personal Identification Number (PIN) (something known). This strong signature legally represents a person or a group. This type of signatures are used in the Portuguese Citizen Card.

2.1.8 Transport Layer Security

TLS is a cryptographic protocol that aims to provide confidentiality and data integrity, during transmission, over the Transmission Control Protocol (TCP). It uses symmetric cryptography to encrypt data. A new set of symmetric keys is generated for each connection. TLS supports asymmetric cryptography which authenticates the identity of the communicating parties. TLS is widely used in web browsing, e-mail and instant messaging. The protocol can provide perfect forward secrecy, assuring any past connections are secure, if in the future encryption keys are compromised.

2.1.9 Public Key Infrastructure

Asymmetric cryptography needs a secure mechanism to validate public keys, achieved by guaranteeing a public key is owned by a certain identity. A Public-Key Infrastructure (PKI) is a central database of public-key certificates. It is responsible for managing, distributing, storing and revoking digital certificates. Digital certificates map public keys to identities, and are used to verify that a specific public key belongs to a given identity. A PKI has several components, e.g., a registration authority, a certification authority and a central database of stored keys. A user can submit other entities' public keys. Entities that trust the user responsible for the submission can use the public keys. There are alternative approaches to PKI, such as a web of trust. This mechanism uses self-signed certificates and third parties attest these certificates. This approach is implemented in Pretty Good Privacy (PGP) [10].

2.1.10 Random Number Generators

A True Random Number Generator (TRNG) can generate a sequence of numbers that cannot be predicted. Generating random numbers is a common and critical requirement for most cryptographic algorithms. Pseudo-random number generators are frequent in software approaches and are not truly random. They depend on an algorithm and initial conditions (seed) to generate random numbers. If the seed is known, the numbers are predictable. TRNG are hardware devices that generate numbers from unpredictable physical conditions. For this reason, TRNGs are perfect for use in cryptography and secure cryptoprocessors.

2.1.11 Public-Key Cryptography Standards #11

Public-Key Cryptography Standards (PKCS) #11 are a group of cryptographic standards, published by RSA Laboratories, which describe guidelines to manipulate common cryptographic objects. The standards define an Application Programming Interface (API) designed to interface between applications and cryptographic devices, such as smart cards or Hardware Security Module (HSM) [11]. Applications can use, create and modify objects, without exposing them to the application's memory. The standard has been widely used, promoting interoperability between devices. By using the same standard, devices can take advantage of another's API. Applications can access cryptographic devices through slots. The slots represent a socket or device reader. A session can be established through a slot, which represents a socket or device reader. The application can authenticate itself to a token with a default PIN. The token holds private and public objects, which can be keys or certificates, among other objects, and can be accessed by the application.

2.2 Secure Cryptoprocessors

In this section we discuss some computing systems that are relevant to this work. Secure cryptoprocessors are dedicated physical computational devices for performing operations, such as cryptography.

Among them, Smart Cards and HSM, which are frequently used to offer cryptography services, will be discussed next. Then, the SmartFusion2 System-on-Chip (SoC) board is presented.

2.2.1 Hardware Security Modules

A HSM is a high grade computational device, responsible for storage, management, generation of cryptographic keys, and cryptographic operations. Keys never leave the device 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., deletion of keys. Some devices have Central Processing Unit (CPU) optimizations to improve the performance of operations. These modules are much costlier than other computational systems but are more powerful in processing power and available services.

2.2.2 Smart Cards

Smart Cards are a type of HSM, credit card-sized with an embedded microchip and provide secure, tamper-resistant storage. These devices have a low price for manufacturing, which allows for bulk production and easy replacement if needed. They have a low computing power, and small memory which allows storage of a small amount of data. To be used, the cards need either contact or contactless readers. These characteristics make them extremely popular, used in many industries, such as retail, healthcare, communication and government.

2.2.3 Field-Programmable Gate Array System-on-Chip

A Field-Programmable Gate Array (FPGA) is an integrated circuit designed to be programmed and configured after manufacturing. FPGAs are often used to prototype and for highly specialized systems produced at low scale. One of the major advantages is their agility and flexibility to be customized for special use cases [12]. However, the reconfigurability may introduce certain weaknesses to the system. Its bitstream is vulnerable to cloning, if no additional protection is applied. The configuration data of these devices is stored in non-volatile memory and may be directly copied if no authentication mechanism is implemented [13].

FPGA is composed of an array of electrically programmable logic blocks. FPGAs can provide a cheaper and faster solution compared to other circuits [14]. These devices can be partially reconfigured while the rest is still running, which is great for production systems. FPGAs are a good option due to less time-to-market and lower cost. They have been used in systems targeting different areas: multimedia, networking, control and bioinformatics [15]. These types of devices have also been used for implementing cryptographic algorithms, e.g., AES and ECC [16].

Smartfusion2 SoC

The SmartFusion2 SoC, illustrated in Figure 2.1, integrates a non-volatile FPGA with a SoC and an internal Non-Volatile Memory (NVM) of 512 KB, for storing boot code. It has a Static Random-Access Memory (SRAM) with 64 KB protected against Single-Event Upset (SEU) or 80 KB unprotected. The board has an embedded ARM Cortex-M3 processor and a TRNG, which provides a quality source of entropy, a critical part of most cryptographic algorithms. It supports multiple cryptographic functions: AES, SHA-256, HMAC and ECC, among others.

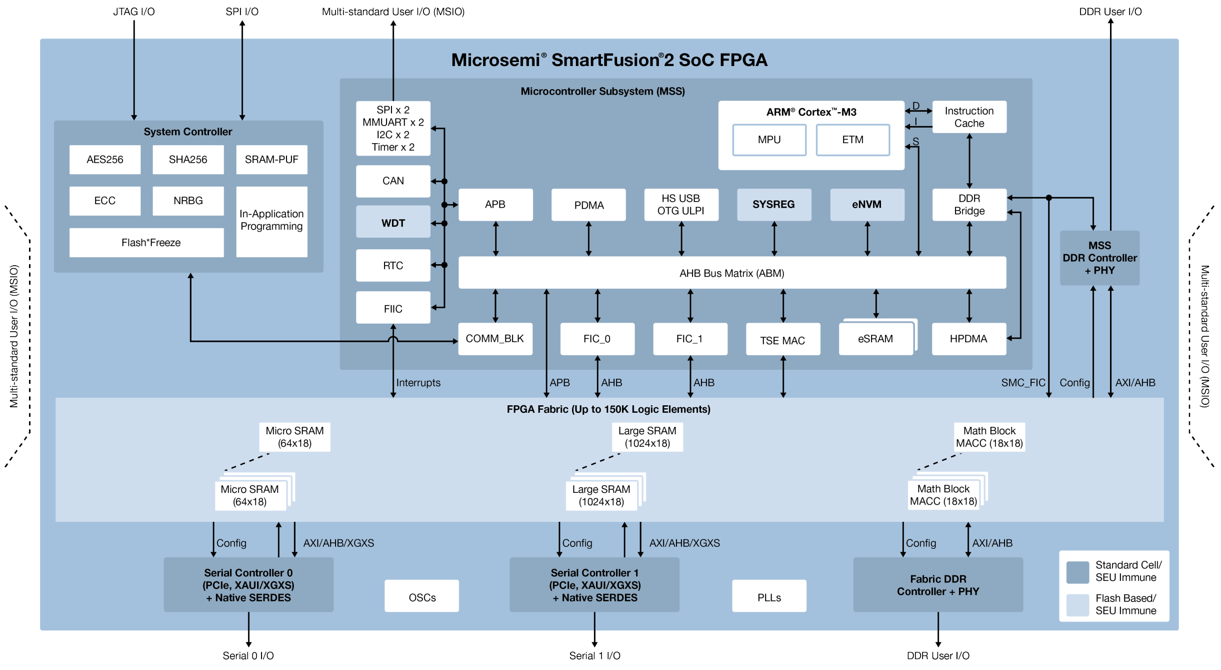


Figure 2.1: SmartFusion2 SoC FPGA Components

The AES system service supports encryption with 128 bit and 256 bit keys, with the following modes: ECB, CBC, OFB and CTR. It provides a SHA-256 hashing service, and a HMAC service using the same hashing function. The KeyTree service provides access to a SHA-256 based key-tree cryptography algorithm. Notably, it can be used to generate a message authentication code from a hash input and key, as a key derivation function with a key and salt as input or in challenge-response protocols. The TRNG service provides a raw entropy source based upon a noisy ring oscillator, measured against a system clock.

The device includes several relevant ECC accelerators, with the National Institute of Standards and Technology (NIST) defined P-384 elliptic curve. It can generate public keys from the corresponding private key, using its scalar point multiplication service, by multiplying the scalar (private key) with the curve's base point. This service also allows the generation of a shared secret with ECDH, by multiplying the device's private key with another user's public key. The device also includes a point addition service, which allows the implementation of the ECDSA algorithm for generating digital signatures.

The board provides a SRAM-PUF service to store user supplied or randomly generated keys. It holds up to 56 key slots with a maximum of 4096 bits each. It uses the random start-up behaviour of a 2KB SRAM block to determine a static secret, unique to each device. There is enough repeatability in the SRAM turn-on behaviour to reconstruct the same secret each time. This secret is used to derive cryptographic keys with 256 bit security strength. When power is removed, the secret disappears. At present, there is no technology able to detect the start-up behaviour of an SRAM [17]. It is determined by atomic-scale manufacturing differences in SRAM transistors. Each enrolled key generates a key code, required along with the secret to generate the specific key. The key codes are protected by AES encryption and stored in a private section of the eNVM.

The eNVM is a tamper-resistant nonvolatile memory. It has a size of 512KB, with the top 64 pages reserved for keys and passcodes, inaccessible by the user. It has a limited number of write cycles. For a predicted life span of 20 years, there is a limit of 1000 writes per page of 128 bytes [18]. For a smaller life span of 10 years, it has a limit of 10000 cycles per page.

The side-channel Differential Power Analysis (DPA) is an advanced power analysis technique, to compute values from statistical analysis of multiple cryptographic operations [19]. Not all services in the SmartFusion2 board are DPA-resistant. ECC point multiplication, Non-Deterministic Random Bit Generator (NRBG), SRAM-PUF and KeyTree have strong resistance measures. On the other hand, AES, SHA-256 and HMAC accelerators have very light countermeasures, and are not considered safe to use repeatedly with the same keys, or in situations where the adversary may be able to choose the ciphertext. For the non DPA-resistant services, there is a danger of key extraction if the same key is used repeatedly. To extract information with this type of attack, physical access to the device is needed. In this case, the device has tamper detection mechanisms, so its services can be blocked, or the device is completely reset and all sensitive information is deleted. This feature is called zeroization, and can be run when a tamper attempt is detected or manually by the user.

Secure boot

Boot code should be validated before its execution leads to potentially executing untrusted code. This can cause problems such as malware insertion, download of intellectual property or user spying. Most embedded processors do not validate code before it is executed. The SmartFusion2 SoC solves this problem by using a non-volatile memory (eNVM) to store boot code, which is write protected. It authenticates each stage of the boot process, to create a chain-of-trust. Other systems also implement solutions to secure boot code. Infineon secures the boot process for ARM platforms by incorporating a Trusted Platform Module (TPM) into the platform. The TPM operates as a root of trust, to certify the platform's integrity and correct system state. This prevents tampered kernels and fault attacks. Texas Instruments

Sitara processors allow the customer to specify a public key as a root of trust, into the device. This key is used to authenticate other keys, used to authenticate software components.

2.3 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. [20] 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 TRNG and a DH based algorithm for key establishment with ECC keys. The controller has protections against side-channel attacks such as DPA and physical manipulation.

Seol et al. [21] 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. [16] 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 [22]. The symmetric keys, which 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 [23] recognizes the need for a HSM PKIs to protect the cryptographic keys. Lorch et al. [24] uses an IBM 4758 cryptographic coprocessor to protect keys in a secure online repository for PKIs. The authors 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.

Several protocols and HSM applications have also been proposed. Rössler et al. [25] 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 [26], [27].

Martina et al. [28] 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 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.

Several HSMs on the market have been studied. Kehret et al. [29] 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, 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. This type 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 [30] 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.

HSMs on the market go from 650€ up to $39,000 [31], [32]. One of the smallest and cheapest devices, the YubiHSM 2 by Yubico [33], is a Universal Serial Bus (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. For comparison, a M2S090TS SmartFusion2 evaluation kit is priced at 384 € [34].

Overall, the studied systems and devices are not suited for this work for one or more reasons. All of the systems focus on a very specific context other than secure communications between entities. The HSMs used in these systems have a high cost and are not very flexible, which prevents its use for a low cost and portable secure communications device. Lastly, other systems use FPGAs which are not security focused and are vulnerable to tampering. Therefore, the proposed solution intends to use a lower cost and flexible system, such as the SmartFusion2 SoC with an integrated and secured FPGA. All the discussed systems could potentially be implemented in such a device. For this reason, it is important to assess the suitableness of this type of device as a HSM for this solution and potentially others. Thus, this work will also have the objective of analysing and evaluating the chosen device regarding its services, functionalities and performance as a suitable HSM alternative.

3 Problem Definition

Contents

This chapter defines the problem, the requirements and lists the necessary services of a potential solution, which successfully addresses the problem. First, some context surrounding the problem is given. Next, the profile of the target users is described and the requirements the solution must provide. Lastly, it details some possible use case scenarios.

3.1 Context

As discussed before, the same computers commonly used for communications and information storage are exploitable by attackers, and can cause minor inconveniences to severe repercussions, such as losing confidential data to malicious parties. This work focuses on securing communications between entities by using a physical device, independent of the user's computer, which is responsible for the security of communications.

3.1.1 Entities

These security focused physical devices are designed to be used by entities, such as, government officials, company executives and military officers. Not just individuals can have an interest in these systems. A device can be assigned to a group of people representing an entity. For example, in the military, a device can be assigned to the navy, one to the infantry and so on. Any ranked officer, or person with a certain level of authority, could use the navy's device, to communicate with other entities, on behalf of the navy.

3.2 Requirements

To effectively address the problem, there are several high-level requirements the solution must fulfill:

- Devices should be distributable to individuals or entities to be used by more than one person;
- The system must allow secure communications between individuals, representing themselves or an entity;
- The device should be independent from the user's personal computer;
- Users should be able to establish secure communications with new and existing entities;
- It should provide an easy-to-use interface for non-technical people;
- It should have a relatively low cost, to allow distribution of several devices among multiple people;
- Only authorized individuals should be able to use the device.

These requirements need to be translated into slightly more technical and tangible requirements. In order to secure communications, the solution should guarantee confidentiality, integrity and authentication. Additionally with asymmetric keys, the system can provide non-repudiation to documents or files, by means of qualified digital signatures. The device must securely store all keys, and perform all cryptographic operations pertaining to the security of communications. Keys must never be exposed unencrypted to the outside.

Additionally, the device should have physical tamper-resistant measures and mechanisms in place, in case of an intrusion, such as, permanent erasure of all sensitive data. This means that even if an attacker is in possession of the physical device, it should be impossible to extract any information from it. The solution should work with a plethora of devices, in order to increase the adoptability of the solution among clients, or to be easily adapted in other projects. Thus, the system should be accessible through a common interface, so it can be used by developers in their own system. The system should provide an application on the user's computer, which communicates with the physical device, and offers a simple interface to its services, for the average non-technical user. The system should use a common connection solution, e.g. USB cable, between the computer and device, to further increase adoptability. In addition, the system should perform the services in a reasonable time, so as to enable efficient text communications, e.g., a chat.

3.3 Use Case Scenarios

This section details use case examples the solution must cater to. Figure 3.1 will be used as an example and illustration for the scenarios.

3.3.1 User Authentication

Every user must authenticate himself before using the device. This can be done by providing a PIN, which the device will verify by comparing it with the one stored in its storage, before unlocking its services. The system will be configured with a default PIN, before it is delivered to entities. The users should be allowed to change this number.

For personal devices there is only one user, the owner, as illustrated by Bob in Figure 3.1. For groups and entities there can be multiple users, illustrated by Alice. In this case, there is only a single authentication PIN for the entity. All the user's with permission to communicate on behalf of the entity, must know the PIN. Only the entity is authenticated, the device does not authenticate itself to the user.

3.3.2 Secure Communications

The main goal of the system is to enable secure communications. Communications can be set up between two or more entities. For each configured communication, the same symmetric key is securely stored in each device. One key per communication. For a user to send secure data, first he authenticates himself to the device, then sends the data to it. The device will return it secured. The user can then send it through a convenient offline service such as email, or an online chat service. Only a recipient with a similar device and the same key can read the original contents.

A possible usage example of secure data exchange between two users, Alice and Bob, is depicted in Figure 3.1. Firstly, Alice forwards the data, to be sent securely to Bob, to the device and the device returns the data secured with an internal key. Then, Alice can forward the result to any entity with the same internal key in their device, through a chat application, e-mail or other convenient service. Upon receiving the secure message from Alice, Bob sends it to the device, and receives the original message back. This service must prevent any third party from gaining access to the data, or altering the message. The receiver can be confident of the data's origin. It could not have been sent by any malicious entity.

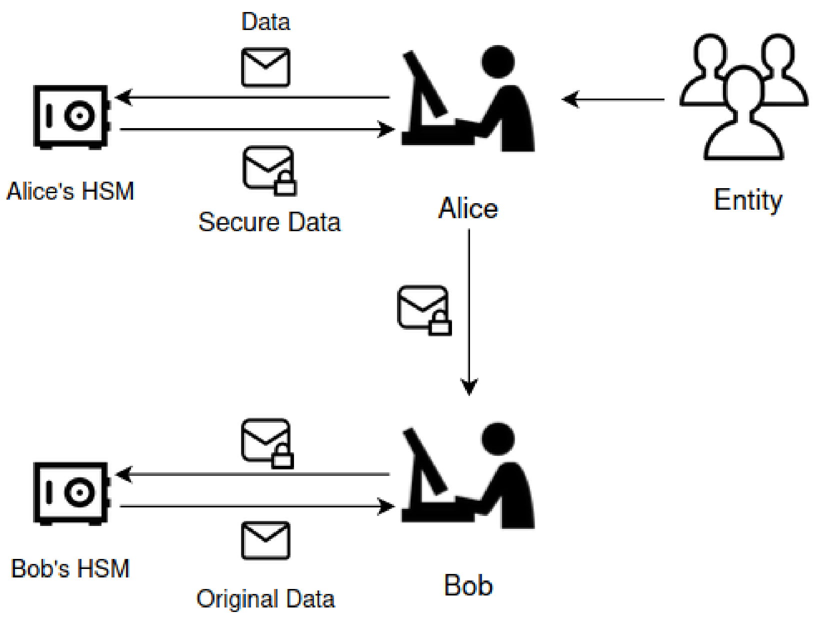


Figure 3.1: Secure data exchange service example using internal keys

Each entity, upon ordering the device, should be able to specify which entities it wants to communicate with. For example, Alice can request a device which enables two separate communication channels with Bob and Charlie, separately. Alice can also request a single channel with all three entities. In this case, only one shared key is required for all. Before devices are delivered, the keys will be generated and stored in the necessary device. When all involved entities receive their device, they can begin secure communications immediately.

The device should have a reasonable amount of key storage space, for communications with enough different entities. If the device's secure storage is limited, then a solution using a higher capacity non-volatile memory must be implemented. This gives entities the flexibility needed to communicate with the number of entities they choose.

Another possible use case scenario is the inclusion of a pair of asymmetric keys, generated inside each device, and never exposed to the outside. This allows the generation of qualified digital signatures, which can legally represent the entity. Then, when a user wishes to generate a signature for a piece of data, he must send the data to the device, which computes and returns the qualified signature.

3.3.3 New Communications

The operations described so far provide a usable and functional system to organizations and users, but it is not very flexible. It does not allow users to communicate with new entities, without returning the device to manufacturing, to be equipped with the necessary information. To avoid this, a service should be available to users with these systems in their possession, to securely trade data with another entity with an identical device, which the device previously could not. It should be easy to provide the functionality without burdening the user with additional responsibility.

A possible solution is the inclusion of a trusted entity, called a control station, where entities originally collect their devices. Each device is delivered with its own communication channel with the control station. This station is effectively treated as a special entity. Whenever users wish to communicate with another user, they can send a list of entities they intend to communicate with to the station, secured with the data exchange service. The control station subsequently supplies the user with the necessary information, to be imported in their device.

A use case scenario of this service is depicted in Figure 3.2. Alice forwards the list with the names of entities to the control station, which returns the key list to Alice. The list can be forwarded to the device, where the new keys are stored. Alice can then start communications with Bob and Charlie, using the secure data exchange service, as soon as everyone imports the list into their respective devices.

The service offers the possibility of setting up a regular communication update schedule. It grants the flexibility of key revocation when needed, data exchange with new entities, as well as updating communication keys, which have reached their expiration date, with no additional complexity to the user. A regular update schedule adds longevity to the system. It avoids overuse, and reduces the possibility of compromised connections. When keys reach the expiration date, new ones can be distributed. A less cumbersome possibility for the user is, whenever an entity is added to the system and receives their device, the station delivers a message to each entity, with the updated key list. The list can be imported by each user if needed.

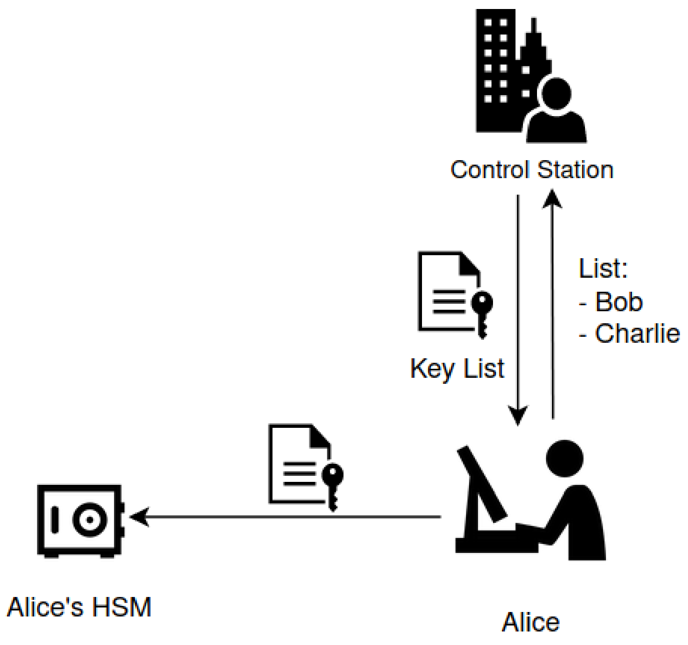


Figure 3.2: Import key list to device usage example

Summary

This chapter defines the context for the addition of a device, independent of a personal computer, which secures communications between entities, such as the military or government. The fundamental and non-technical requirements for an adequate solution were presented, such as, its characteristics, capabilities and usability. In short, the system should be composed of a user-friendly and low cost device, easily distributed among individuals or groups, and provide secure communications between them. Possible use case scenarios for the system were identified. The system could be used to authenticate a user to the device, to allow access to its secure communication capabilities. The system must allow communications between entities with identical devices, by the means of a computer connected to the device, which secures messages before transmission. The system could also allow the establishment of communications with new entities, by employing a control station which, when necessary, distributes the essential information among all entities, to allow new secure communication channels.

4 Architecture

Contents

The fundamental objective of this work is to develop a portable device, which enables entities to establish secure channels of communication. A solution was developed on a portable HSM, which secures communications between users, stores the owner's sensitive data, such as keys, and performs all services critical to its security. This is an easy process for the owner, who does not need to concern himself with any setup or management, the system is accessible and ready to use when received. This chapter presents an overview of the developed system, its services, the protocol used for communications and use cases, unconstrained by a specific physical device.

4.1 Overview

The system, pictured in Figure 4.1, is composed of two main components: the physical device, responsible for all operations, and the application on the user's computer, which provides a straightforward interface to users.

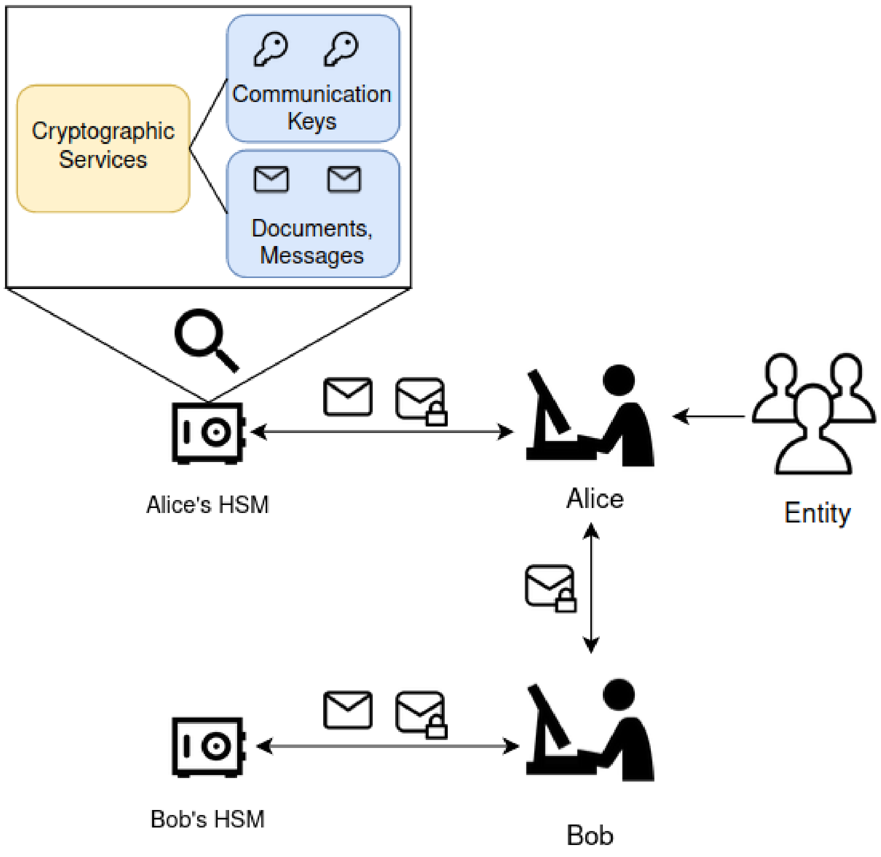


Figure 4.1: System Overview

The user's computer running the software is connected to the device, through a physical connection, used to send and receive data and commands. Using these tools, the user can request the device to perform the desired services. These services are implemented inside the device, which stores and manages keys, as well as other data. If the device is misplaced or stolen, the stored keys and documents are not at risk of being extracted. The developed software and physical tamper measures ensure it. Upon receiving their device, each user is only required to connect it to their computer using the appropriate cable. The system is ready to use, through the provided interface software, which should be used in the user's personal computer.

4.2 Services

This section describes each system service and all its relevant information. Firstly, it presents the authentication, then the secure data exchange service, qualified digital signatures and services to establish new communication endpoints.

4.2.1 Authentication

Each entity will have its own device, which can be connected to a computer. To access the device's services, all users need to be authenticated to the device, through a PIN, which is securely stored inside the device. There is only a single PIN, used to authenticate the entity to their device. Each entity can supply this PIN to users, which will use the device on behalf of the entity. After the user sends the correct PIN and is authenticated, the PIN can be changed.

4.2.2 Secure Data Exchange

As previously introduced, the main goal of this solution is to secure communications between multiple entities, with identical devices. Herein defined is the secure data exchange service, responsible for securing communications between entities. Communications are secured by providing three services: confidentiality, integrity and authentication. This allows communicating entities to ascertain the origin of messages and prevent unauthorized entities from reading and modifying them. To grant these services, communicating entities must agree on a symmetric key. This key will be stored in the devices of both entities, and is never exposed outside the device. In order for entities to agree on a key, the system provides services which will be discussed further ahead. In order to describe the secure data exchange service, we will assume a symmetric key has already been agreed between both entities.

As discussed in the previous chapter, and depicted in Figure 4.1, Alice's computer will communicate with her HSM, in order to secure data, to be sent to Bob. For this, there needs to be a communications protocol between the computer and HSM, to define what data is sent, in what order and by who. To secure a piece of data sent by Alice, the device needs to receive the data, and return the data encrypted and authenticated with an internal key, previously agreed by Bob and Alice. Thus, the communication protocol, between the device and computer application, is illustrated in Figure 4.2.

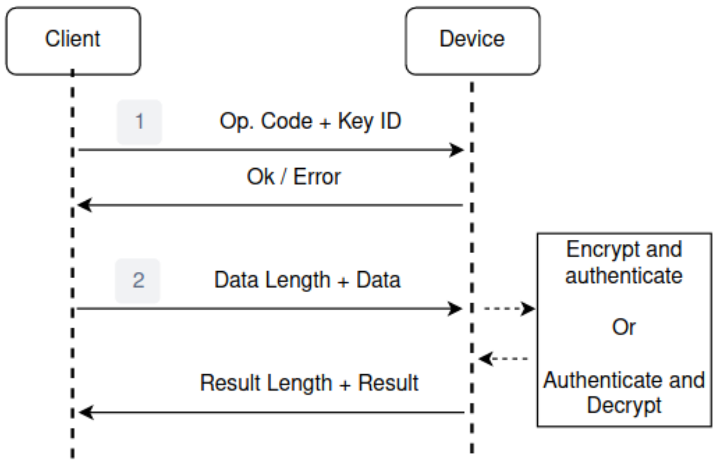


Figure 4.2: Communication protocol for data ciphering and authentication in the HSM with internal keys

Since the system is designed to have multiple services, and be able to store multiple keys, there needs to be a mechanism to identify the different services and keys. Services will be identified through an operation code, with an unique value for each service. Similarly, keys will be identified through an ID with an unique value for each key. Alice must know the ID for every key and to what entity it was agreed with. The ID is stored in the device along with the corresponding key.

Thus, if Alice wants to secure data, the computer must first send a message composed of a code, identifying the secure data exchange service, and the key ID, to identify the symmetric key which will be used to secure data. Upon receiving the message, the device will check internally if the user is authenticated and if both the operation code corresponds with an existing service and if the key with the corresponding ID exists. Subsequently, the corresponding success or failure status is returned by the device.

After a successful exchange, the data can be sent to the device. Before the data is received, the device must know the size of the data which it will receive from the computer. After the data is processed in the device, its output is sent back to Alice, which can be securely sent to another entity, with the same symmetric key in their device. If Bob wants to obtain the plaintext of the data sent by Alice, the same communication process is used. Bob sends the correct operation code and key ID, receives a success status response, then sends the secured data from Alice, and waits for the plaintext response from the device.

To produce the secure data, the plaintext must be encrypted and authenticated to provide the three necessary cryptography services: confidentiality, authentication and integrity. On the other hand, there must also be a process to authenticate and decrypt the data, in order to retrieve the plaintext. Therefore, a protocol for encryption and encryption with authentication is needed.

As presented in Section 2.1, 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. These boards usually provide separate encryption and authentication algorithms. Thus, to mitigate this problem, the proposed solution combines an encryption and authentication scheme, in order to provide the necessary cryptography services. Both algorithms can be combined in different orders. Studies recommend combining a secure encryption and secure MAC, with the encrypt-then-MAC method [35]. This method encrypts the plaintext first, then generates the MAC from the generated ciphertext.

From this information, the proposed encryption protocol is pictured in Equation 4.1. First, the plaintext data is encrypted with an internal symmetric key and a randomly generated IV. 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.

$$E_{key}\{Data, IV\}, MAC_{key}\{IV + E_{key}\{Data, IV\}\} \tag{4.1}$$

The decryption protocol is pictured in Equation 4.2. 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 receiver can be sure of the data's origin and integrity. The ciphertext is decrypted with the same internal symmetric key used for encryption and the received IV, to obtain the plaintext.

$$(MAC_{key}\{IV + Ciphertext\} == MAC) => E_{key}\{Ciphertext, IV\} => Data \tag{4.2}$$

4.2.3 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. 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, a private and public key pair must be randomly generated in the device, or imported from the outside if the device does not support this. This must be done before the device is delivered to entities.

As with the previous service, a communications protocol between the user and device must be defined. The device must receive the data, from which it will generate the signature, while the user will receive the generated signature. The devised communication protocol is presented in Figure 4.3.

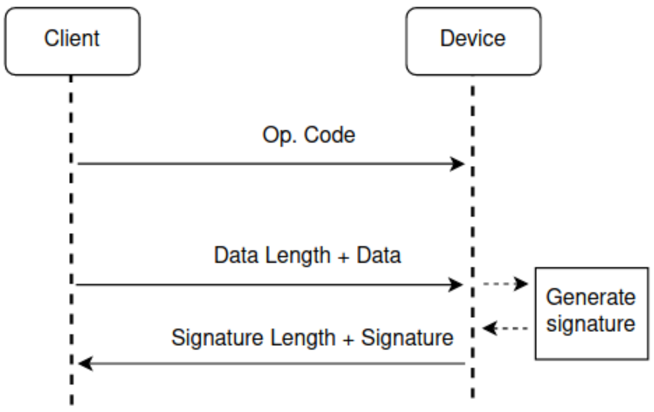


Figure 4.3: Communication protocol for generating digital signatures using internal HSM asymmetric key pair

In order to identify the service, the user will send the corresponding operation code, along with the data to be signed. Upon reception of the data, the device checks if the user is authenticated and subsequently generates the qualified signature, using its internal private key. Afterwards, the device responds with the generated signature. Other entities can validate the signature using the author's public key. Only the entity, in possession of their device with its private key, could have generated the signature.

As discussed previously, public key cryptography is slow. Therefore, a digital signature is generated by first computing a hash of the message, then signing the hash with the private key, as shown in Equation 4.3. In general, generating a signature from a hash is faster, since it has a small fixed size, while the signed data can be of any size.

$$Sign_K\{Hash\{Data\}\} \tag{4.3}$$

4.2.4 Establishing New Communication Endpoints

The system provides the flexibility to establish secure communications with new entities, with identical devices. To achieve this, entities must agree on a new symmetric key, which can be subsequently used with the secure data exchange service, presented in Section 4.2.2, to secure communications between them. All new keys are stored in the device's secure storage, along with its unique ID. Two different mechanisms to generate and agree on new symmetric keys are presented next. The first uses public key cryptography and the second uses symmetric key cryptography.

4.2.4.A Key Generation

As introduced in Section 2.1.6, two entities can agree on a symmetric key, using public key cryptography. 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.

Just like previous services, there needs to be a communication protocol to trade the necessary information, and generate the key. The device must receive the peer's public key and return its own public key. After this trade, the key is generated and stored in the device. Following these guidelines, the protocol to generate a shared symmetric key with another entity, using asymmetric cryptography, is detailed in Figure 4.4.

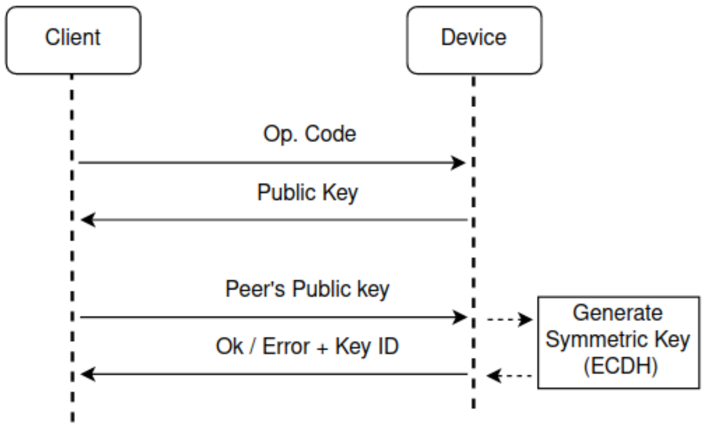


Figure 4.4: Communication protocol to generate a symmetric key with another entity, using an internal private key

The protocol starts with the user sending the operation code, signaling the device it wants to generate a symmetric key. The device responds with its public key, which needs to be shared with the other entity. After the user receives the peer's public key, it forwards this key to the device, and waits for a response. If successful, the device internally generated the symmetric key, which is now stored in its secure storage. The device also returns the new key ID to the user, so the entity can keep track of the ID which it needs to use to securely communicate with the other entity. The internal key generation process only needs the internal private key and the received public key.

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.

4.2.4.B Import Keys

The key importation service provides a mechanism to share and import keys to the entities' devices using symmetric key cryptography. This alternative provides flexibility to the system. Users are not constrained to key sharing using public key cryptography. This service also provides a mechanism to solve another problem. If symmetric keys are used incorrectly or for too long, attackers can collect enough data to potentially break the security of keys. This can be mitigated by this service, since it allows updating the existing keys, to avoid its overuse.

In order to import the keys, the device receives an encrypted and authenticated list of keys, decrypts them with the secure data exchange service used for secure communications and overwrites the existing internal keys in the device. The communication protocol for this service is depicted in Figure 4.5.

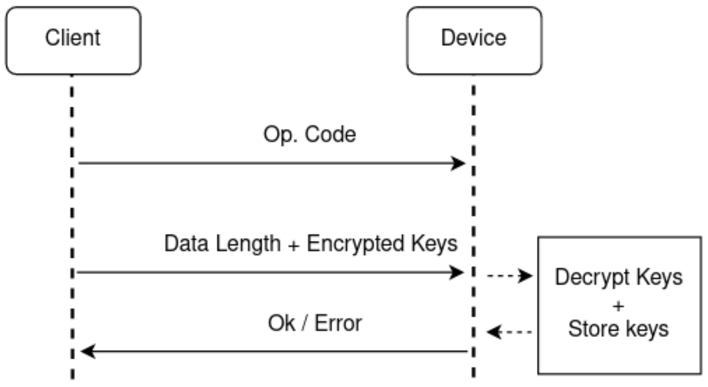


Figure 4.5: Communication protocol to import encrypted and authenticated symmetric keys, and store them in the HSM

The operation code, sent by the user, signals the device it will subsequently receive a list of keys.

Upon receiving the keys, the device authenticates and decrypts the keys using an internal key. Each key on the list is accompanied by 1 byte for its size and 1 byte for its ID.

As mentioned, the keys are received encrypted and authenticated. This must have been previously processed by another special entity, using the secure data exchange service. The entity importing the keys into their device, must have a symmetric key in their device, to communicate with the special entity. This entity called ”control station”, was introduced in the previous chapter. The sole purpose of the entity is to act as a trusted third party, which distributes keys to the entities, through the secure data exchange service. Each system must be delivered to entities with an internal symmetric key for communications with the control station. Thus, entities can communicate with the control station to request keys with no setup or key agreement necessary.

Whenever an entity wishes to communicate with others, they can send a list of entities they intend to communicate with to the station, secured with the data exchange service. The control station subsequently supplies the user with the list of keys, to be imported in their device. For example, if Alice wants to communicate with Bob, Alice can request the key from the control station. Then, the control station generates the key and sends it to both Bob and Alice, encrypted and authenticated with different predefined keys for both. Alice and Bob use this service to import the key into their devices, and can subsequently begin secure communications between each other.

4.3 Summary

This chapter presented the system’s functionalities, algorithms and communications protocols, between a secure and portable device, and the user’s computer. The system grants confidentiality and authentication to communications between any number of users, as soon as they receive the device, without overloading the users with convoluted tasks or responsibilities. It provides entities with the flexibility to manage which entities they wish to communicate with by themselves, or by offloading the management responsibility to a trusted control station.

5 Implementation

Contents

This chapter presents the details of the developed solution on the HSM, the user software and the developed API, which accesses the HSM services. It covers the implementation details, namely the standards and libraries, the developed services, their logic and how the device's accelerators are leveraged. The main components are the communications, secure data exchange service, digital signatures, key management solution and tamper detection features.

5.1 Libraries and Tools

The device's implementation is running on the SoC of a SmartFusion2 board, version M2S090TS from Microsemi, presented in Section 2.2.3. The application was implemented using the C programming language, with the SoftConsole version 3.4 Integrated Development Environment (IDE) provided by Microsemi. The board was configured using Libero version 12.2. The device functions as a HSM, connected through a serial connection to a computer. This connection is only for prototyping, the board also has a much faster USB connection. It was programmed using the external FlashPro4 programmer, required to develop and debug embedded applications with the SoftConsole IDE [17].

A common developer interface, to access the device's services, was implemented using the PKCS#11 standard, in order to standardize the accessibility of its services to other systems. The developed user application has a simple Command line Interface (CLI) interface, which invokes the PKCS#11 functions. Both the software and the API were implemented in C/C++, for the open source MinGW compiler for Windows 10. The device implementation uses the available Universal Asynchronous Receiver-Transmitter (UART) communication controller and corresponding drivers for communications, while the user application uses the windows drivers to communicate with serial ports.

5.2 HSM Services

The services were implemented using the SmartFusion2 security accelerators: SRAM-PUF, AES, HMAC, SHA, ECC, TRNG and tamper detection. All the implemented services are accessed through the developed PKCS#11 API, with the communication protocols detailed in Section 4.2.

By default, the device's Random-Access Memory (RAM) has 64 KB for running code. For each byte of the RAM, there are 2 bits for error detection and correction, a total of 16 KB. This feature mitigates SEU, which can flip memory bits. It corrects 1 bit errors and detects up to 2 bit errors. If more memory is needed for more application code, the board can be configured with this setting disabled in Libero. This frees the additional 16 KB of memory for RAM, for a total of 80 KB.

5.2.1 Login

Entities must be authenticated before they can access the device's services. The login operation authenticates the user, using an eight digit PIN. There is only a single PIN to authenticate the entity. The individual users are not authenticated. The PIN is stored in the secure storage of the internal device. The available secure storage and management solutions which can be used to store the number are discussed ahead.

5.2.2 Secure Communications

The goal of this service is to provide authentication and confidentiality to the data being exchanged, between entities with these devices, using internal keys. In order to provide these services, a protocol was previously presented in Section 4.2.2, which combines a symmetric encryption and authentication scheme. The algorithms are combined with an encrypt-then-MAC approach, meaning the plaintext is encrypted first, then authenticated.

The SmartFusion2 SoC supports the type of algorithms needed for this protocol. For symmetric key authentication, the board supports HMAC with the SHA-256 hash function, which uses a 256 bit symmetric key, to generate a 256 bit code. As discussed in Section 2.1.3, it is a well-designed construction, even though it is not the most efficient. For confidentiality, the board provides several AES symmetric encryption modes, with 128 bit and 256 bit keys. Among them, CBC and CTR are the most favourable options because of their security. For every encryption operation, a random 16 byte IV is generated, using the TRNG service.

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 [36], 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.

For the implementation, AES 256 bit was used and the board was configured with its maximum of RAM, 80 KB. With all the code needed for this implementation, a maximum of 36 KB is available for the plaintext and ciphertext buffers. Initially, two buffers of 18 KB each were used, one for the input data and one for the output result. This was improved to a single 36 KB buffer, where the initial input is stored, as well as the result while it is computed. Therefore, with this protocol implementation, the system is limited to securing 36 KB of plaintext. In order to improve this, a continuous cipher and authentication implementation is needed. The data would not be received in full, but instead, the data would be divided in chunks, which are received by the device, computed and returned to the user, one after the other. However, the board is not capable of continuous operations using the provided AES and HMAC implementations.

To overcome this limitation, the characteristics of the AES CBC mode can be used, as illustrated in Figure 5.1. CBC mode encrypts data one 16 byte block at a time, using an IV. The IV of the first block

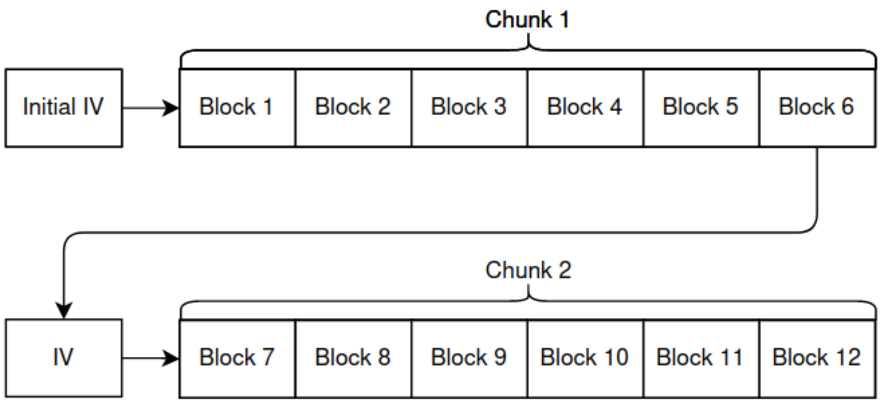


Figure 5.1: Continuous CBC block encryption of data divided in chunks

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. In order to allow continuous authentication, a lightweight software implementation of HMAC was included [37].

Since with this approach, the full data is divided in chunks, a modified communications protocol is necessary, so the device can receive and send the data, one chunk at a time. The developed continuous communications protocol is depicted in Figure 5.2. For the initial API call, the operation code identifying the service is sent, along with the key ID which identifies the internal key stored in the device. Then, each subsequent update call, sends a chunk of data along with its length. For each chunk received, the device adds the chunk to its buffer, encrypts it, updates the HMAC calculation, and returns the ciphertext chunk. When there are no more chunks to receive, the board returns the remaining ciphertext and final MAC value. The total output is composed of the initial IV, ciphertext and MAC. In order 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

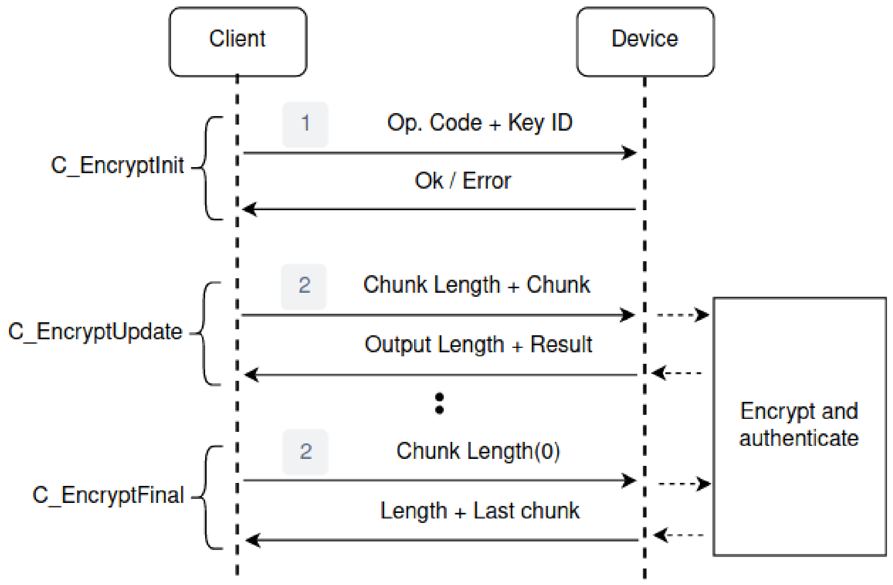


Figure 5.2: Communication protocol for continuous data ciphering and authentication in the HSM with internal keys

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.

The reverse operation for authentication and decryption of data is exactly the same. The only difference is in the operation code value, so the device knows what operation to perform.

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. The performance of the FPGA and SoC AES cores are analysed in the next chapter.

5.2.3 Key Management

Symmetric and asymmetric keys are essential for the functionality and security of the system and its services. Secure symmetric keys are necessary for the security of communications, while asymmetric keys are needed for key agreement and digital signatures. All these keys are stored inside the device, and the system is responsible for their security and management. 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.

As described in Section 2.2.3, 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 5.3. In order to securely store the

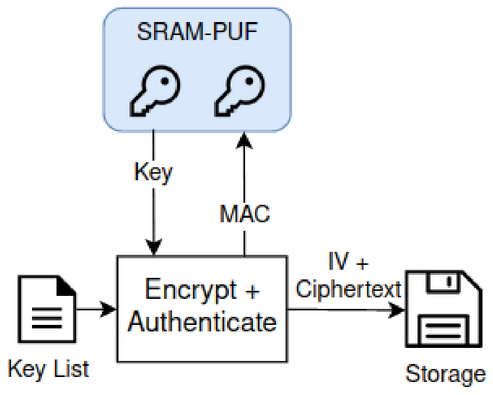


Figure 5.3: Key management solution to store keys encrypted and authenticated in a non volatile memory using the SRAM-PUF secure storage

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 less frequent. 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.

The service, introduced in Section 4.2.4.B, for importation of keys was designed, in order to provide a list of keys for the key management solution. 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.

5.2.4 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. This service takes advantage of public-key cryptography in order to agree on a key. The key generation service follows the protocol and guidelines defined in the previous Section 4.2.4.A. Both entities have a private and public key pair. After sharing public keys with each other, they can generate the same secret using their private key and the public key of the other entity. The SmartFusion2 SoC provides the necessary ECC scalar multiplication accelerator, to generate a shared key with the ECDH algorithm. The accelerator multiplies the 48 byte private key with a 96 byte public key to generate a 96 byte shared value, which can be stored using the system's key management solution.

5.2.5 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. As pictured in Figure 5.4, 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 on the P-384 curve, 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.

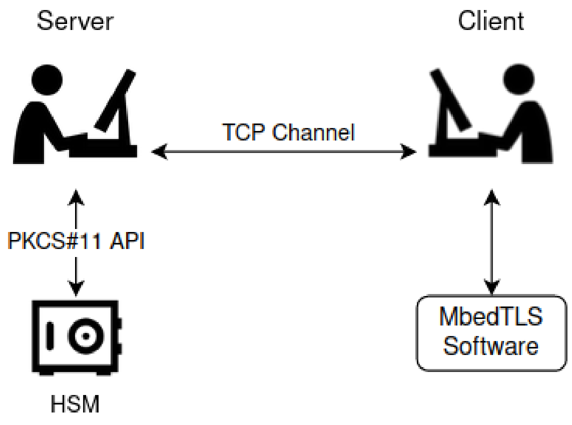


Figure 5.4: TCP connection between a server using the HSM system through PKCS#11 API calls and a client running a MbedTLS implementation

5.2.6 Qualified Digital Signatures

The goal of digital signatures is to provide non-repudiation to messages. This prevents an entity from denying the authorship of a message. The service was designed to provide an improved version of signatures, called qualified digital signatures, where the private keys, which generate signatures, are stored inside the HSM, and are never exposed to the outside. This guarantees the signature generator is someone with access to the device with the private key.

The SmartFusion2 SoC provides ECC key support which can be used to generate signatures with the ECDSA algorithm. However, the board does not provide an ECDSA implementation. Either the board uses an external source for digital signature generation, which would cease to be qualified, or the algorithm must be implemented in the board.

In general, the ECDSA algorithm is composed of two types of operations. ECC operations, e.g., scalar multiplication and point addition, which are supported by the board. The other is large numbers operations, such as modular multiplications and divisions. These operations are not supported by the board, a big numbers library is necessary. In case of the supported P-384 curve, the library should support operations between 48 byte numbers.

Thus, a big numbers library was included in the system [38]. The library was stripped of all unnecessary code and headers, keeping the necessary big integer operations. The ECC primitives and true random number generator were included in the library code. Digital signature generation was implemented, without verification. The library takes up around 54 KB of RAM space, with all other services disabled. This is an extremely limited use case, and less than ideal. Future work should revise this functionality, by evaluating existing lightweight libraries, which provide the necessary features, or even a custom implementation.

5.2.7 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 utilized.

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. 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. Depending on the context in which the system is used, it can be a trade-off deemed necessary. However, the system can have mechanisms to enable users to continue using the device. For example, a user can insert a special code provided by an administrator, in order to unlock the device after it has detected a potential tamper attack.

5.3 PKCS#11 API

In order to demonstrate the flexibility of the system, a PKCS#11 API was developed to access the HSM services. Developers who wish to use this system for future work on a SmartFusion2 SoC can call this API from the developed software. PKCS#11 defines two types of users, the regular user and the security officer. The implementation does not make that distinction. Both types of user can login, and gain access to all other services. The full list of the implemented PKCS#11 functions, their arguments and description are listed in Table A.1.

Summary

This chapter presented a proof of concept HSM application supported on a SmartFusion2 device, using its security services and accelerators. It specified the used libraries and tools, all implemented services motivation, details, advantages and drawbacks, as well as the developed PKCS#11 API to access these services. The prototype focused on secure communications with authentication and confidentiality. It provided improvements on the architecture baseline set in Chapter 4, using the device's capabilities and benefiting from software implementations. It also provided flexibility with a key management solution, using the board's secure storage efficiently, new key generation and qualified digital signatures using public-key cryptography features. The work also explored the device's drawbacks and provided possible ways to mitigate them. Lastly, it presented the available tamper detection options and zeroization features, with implementation examples.

6 System Evaluation

Contents

This chapter presents the evaluation of the SmartFusion2 board and the developed prototype, its services, their performance, capabilities and limitations. The system and board were also evaluated regarding the fulfilled requirements, outlined in Chapter 3.

6.1 Performance Tests

The test objectives, configuration, results and conclusions are presented for every component tested. The communication channel, device's security services and implemented services were tested. Two performance metrics are used, the average test processing time, and the tested component's throughput.

6.1.1 Testing configuration

The tests were all performed on a Windows 10 computer, running the user software which calls the implemented PKCS#11 API. The software communicates with the SmartFusion2 device through a serial port. For all tests, the connection was configured with a 115200 bit/s baud rate, with 8 data bits, no parity bits and one stop bit. 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 using the function `gettimeofday()` with microseconds precision, available in the C library `sys/time.h`.

In order to thoroughly study the performance and scalability of each component, the transmitted data size was varied, for components where it can potentially have a performance impact. The data size was tested, when possible, up to 36 KB, which is the maximum buffer size, limited by the 80 KB of RAM.

The communication channel performance was measured by transmitting a certain amount of data up to 36 KB. For each data size, the test was repeated 30 times, in order to achieve a variance below 1%. For the remaining tested components, this test method produced unstable results due to the communications overhead on every test run. Thus, the remaining services were tested by running the service 100 times inside the device, for each API call. The measured processing time is divided by 100 to obtain the average time. This method minimizes the communications overhead, since it is only done once, so the isolated service can be more accurately assessed. The services were run 100 times inside the device, since it produced below 1% variance for every test run of every service. Due to the write limitations of the PUF and eNVM, their performance tests were performed with only 10 repetition to preserve its write cycles.

The SmartFusion2 and implemented services were tested taking into account three different time components $T_{Total} = T_{Call} + T_{Transmittion} + T_{Service}$. The T_{Call} component is the elapsed time for invoking the PKCS#11 API, before any data is transmitted. From the results, we can conclude that the impact of this component on the performance is negligible, since it is practically always 0. The data transmission time follows the performance behaviour of the communications channel. Thus, all service tests focused solely on measuring the performance of the isolated service on the device, with no data transmission. To get a complete model of the overall performance of the system from the user's perspective, the isolated service processing time can be added to the communications time.

6.1.2 Performance models

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. In those cases, performance can be modeled with a formula composed of two different components 6.1, a constant value independent of processed data, and a factor dependent on the processed data size.

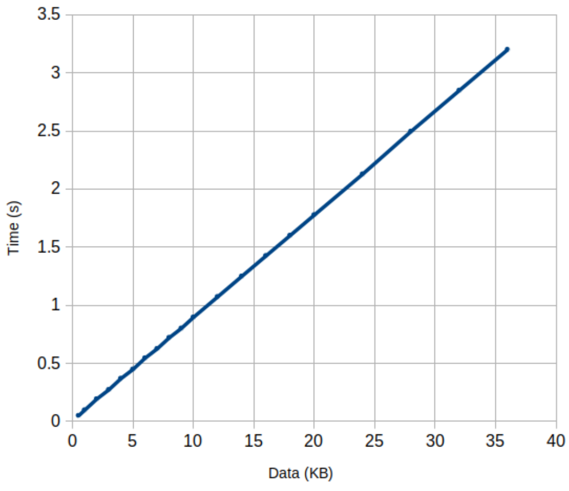
$$T_{Total} = T_{Constant} + T_{Data} * KB \tag{6.1}$$

In those cases, in order to provide a characterization of the service's performance, these values were calculated. So as to assess the accuracy of the models, the median average percentage error was calculated and is presented next to its values. Tests which always process a fixed data size, such as the ECC and KeyTree core services, only have a constant time component.

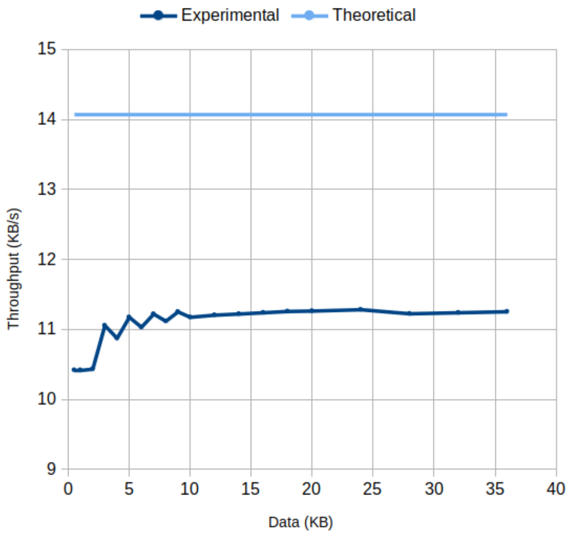
6.1.3 Communications

In order to assess the communication channel performance, and its impact on the system, the average time to transmit data was measured. For each test, the computer application sends data of a specific size to the device, which returns an acknowledge message on reception. The average transmission times for each value are displayed in Figure 6.1(a). The values range from 0.048 seconds for 0.5 KB, to 3.2 s for 36 KB. Comparing the communication channel performance to the isolated services performance, we observe the communications channel has time values from 0 to 3.2 seconds, while with the same data sizes, the performance of the services does not go above 1 second. Therefore, we conclude the bottleneck of this system is located in the serial port connection. This is expected since this channel is intended for prototyping and the board provides a much faster USB connection for production systems.

In the subsequent graphic, the throughput was calculated from the transmission results. Figure 6.1(b) depicts the experimental throughput and theoretical throughput. The theoretical throughput was calculated from the baud rate $115200/8 = 14.06KB/s$. We can observe that the experimental throughput starts at around 10 KB/s for smaller data sizes, and stabilizes around 11 KB/s, as size increases. We can conclude that the practical throughput is relatively close to the theoretical, and as expected, stabilizes as the data size increases. The performance of the data channel, in milliseconds, can be modelled



(a) Data Transmission Time



(b) Channel Throughput

Figure 6.1: Serial port data transmission results

by a linear equation 6.1, with values $T_{total} = 7.281 + 88.638 * KB$, and a median average percentage error of 0.92 %.

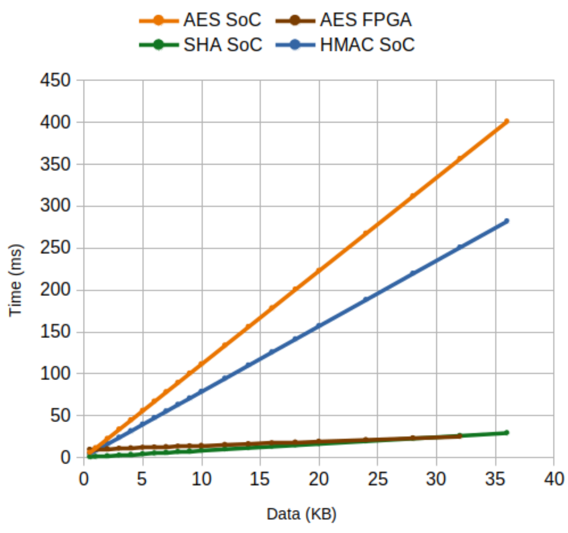
6.1.4 SmartFusion2 Services

All the security accelerators of the SmartFusion2 SoC were tested. This includes the TRNG, the AES SoC accelerator, SHA, HMAC, KeyTree and ECC scalar multiplication and point addition services. Additionally, a side-channel resistant 128 bit AES core implemented in the FPGA was also tested. The isolated service processing time and throughput results are presented in Figures 6.2(a) and 6.2(b).

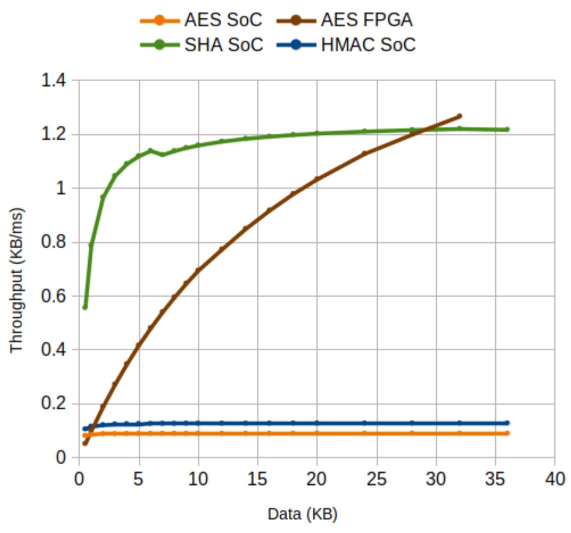
The AES SoC service was tested with all possible variations. Namely, with 128 bit and 256 bit keys, with all four available modes and with encryption and decryption. Only one result is shown, since there was no variation among them. The AES mode, key size or encryption/decryption operation does not impact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

Overall, the SoC AES and HMAC total service time increases faster, compared to SHA, which increases significantly slower. The AES core implemented on the FPGA is significantly faster than the AES SoC core, with performance comparable to the SHA accelerator. Due to its side channel mitigations and performance advantage, the AES FPGA core is a significantly better choice than the AES SoC core.

Figure 6.2(b) represents the throughput calculated from the processing time results previously gath-



(a) Processing Time



(b) Throughput

Figure 6.2: Security services performance and throughput evolution

ered. It shows 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 AES and HMAC at around 0.1 KB/ms.

All data dependent services followed a near perfect linear evolution, in function of the processed data size, as presented in Table 6.1. Analysing the calculated values can lead to further conclusions regarding

Time (ms)	AES SoC	AES FPGA	SHA	HMAC	TRNG	KeyTree	ECC Add.	ECDH
Constant	0.489	9.541	0.498	0.783	0.368	1.655	7.204	545.381
Data (KB)	11.124	0.492	0.807	7.815	0.007	-	-	-
MAE	0.12%	0.25%	0.84%	0.13%	2.29%	-	-	-

Table 6.1: SmartFusion2 services time performance according to a linear model

the AES FPGA and SoC core. The FPGA core has a much higher constant performance compared to the SoC core. This is relevant for smaller data sizes, where the constant value is the most relevant. Thus, for smaller sizes, the FPGA core might not be the better choice in regards to performance. Furthermore, we can conclude that the scalar multiplication (ECDH) is the slowest, with each run taking more than half a second. 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.

Core/Software Comparison

The SHA and HMAC SoC cores 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 the previous chapter, of HMAC, SHA and AES were tested for comparison with the SmartFusion2 SoC services. The library used for HMAC and SHA was [37], and for AES [39].

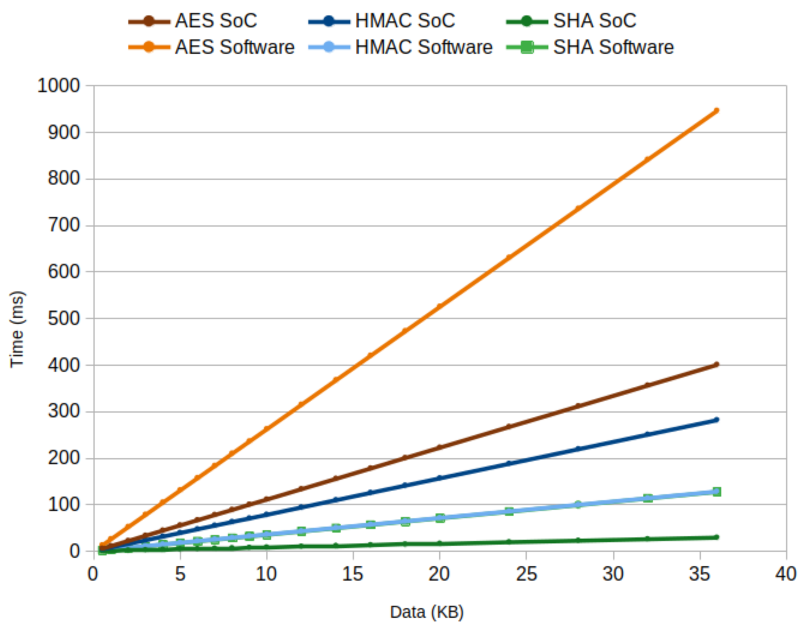


Figure 6.3: Performance comparison of the board's cores and a software implementation

Analyzing the time performance results in Figure 6.3, 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. This leads to the conclusion that the HMAC on the SoC is a worst choice compared to the software implementation, for both performance and features. The software HMAC provides continuous authentication while the SoC does not.

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 performance results for the RAM, NVM and PUF are pictured in Figures B.1 and B.2 of Appendix B.

The results show 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. The RAM read performance was also tested, however, the results are not shown due to its extremely negligible and erratic performance. Instead, a simple increment operation test is presented, in order to show a benchmark for the RAM's performance.

The linear model values were calculated and are presented in Table 6.2. From the results we can

Time (ms)	RAM Op.	Write RAM	Read NVM	Write NVM	Read PUF	Write PUF
Constant	0.085	0.012	0.01	8.03	128.49	747.84
Data (KB)	0.34	0.014	0.02	298.10	0.006	0.008
MAE	1.90%	2.76%	3.76%	0.31%	0.17%	0.06%

Table 6.2: SmartFusion2 read and write performance time values according to a linear model

conclude the volatile RAM is the faster memory, while the eNVM is slower. The SRAM-PUF has the worst performance with smaller data sizes, since besides using the eNVM for storing part of the data, it performs calculations for both read and write operations in order to securely generate or store the keys.

6.1.5 Implemented Services

This section presents the performance results of the implemented services from Chapter 5. Each service's performance depends on the used accelerator's, memory access and implemented logic. The tested services were, encryption and authentication, decryption and authentication, both using the AES SoC accelerator and the HMAC software implementation; the key import service, using the SoC AES, HMAC and PUF cores, ECDSA and ECDH, both using the ECC scalar multiplication accelerator. ECDSA also uses the ECC point addition core. It is worth noting that the encryption service also uses the TRNG service, to generate a random IV. The continuous encryption and decryption services used a 36 KB buffer for data.

The performance of the key import service is depicted in Figure B.2 in Appendix B. The figure compares this service to the PUF read and write performance, in order to compare both key storage options. Just as the PUF service, the key import service barely varied, due to the small data interval. Analyzing the results in Figure B.3 of Appendix B, both encryption/decryption and authentication services have very similar values, due to being based on the same service's and AES encryption and decryption having no discernible performance difference.

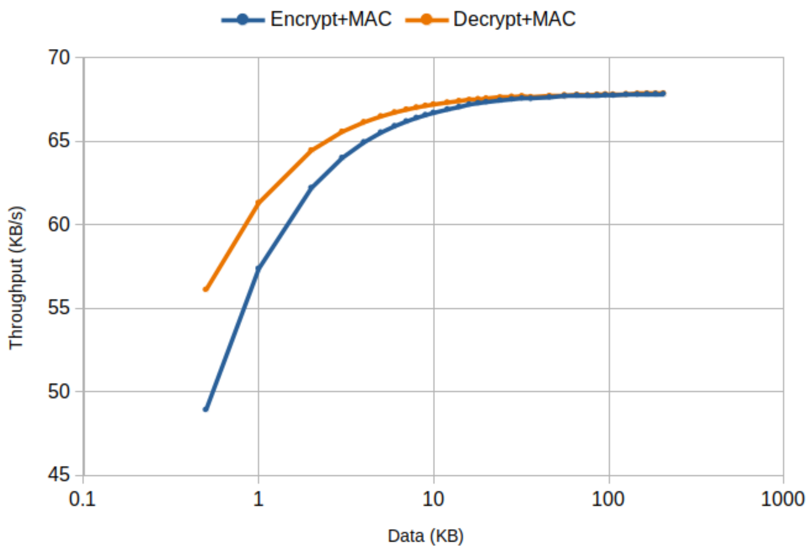


Figure 6.4: Continuous encryption and decryption throughput evolution for increasing input data sizes

Figure 6.4 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 6.3.

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

Table 6.3: Implemented services time performance values according to a linear model

By analysing the calculated models we can detect a difference 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 6.5. The worst performance, and therefore lower throughput, is

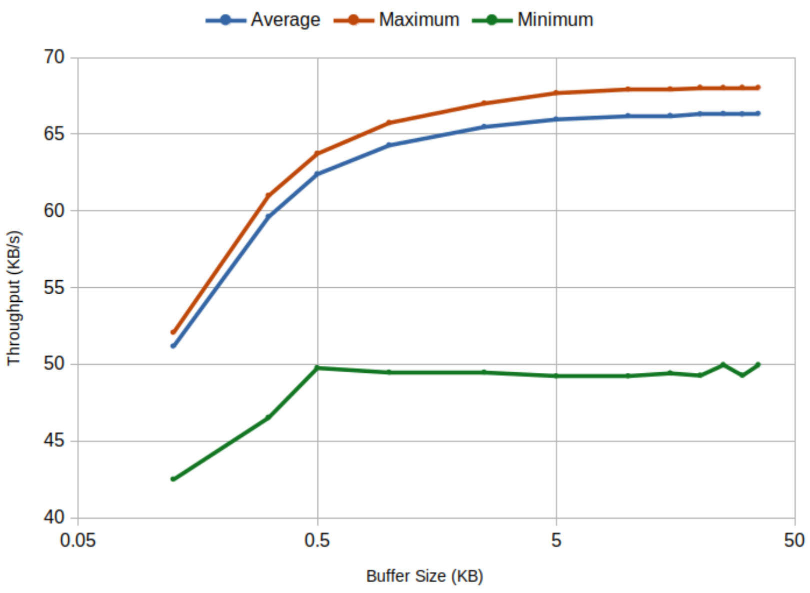


Figure 6.5: Continuous encryption and MAC throughput evolution with increasing buffer sizes

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.

6.2 Requirements

This section lists all the fulfilled requirements of the developed system, defined in Chapter 3.2. The SmartFusion2 SoC is a portable board, with a robust set of security services. It provides a set of security services on the same level of higher grade HSMs on the market, presented in Chapter 2. 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 [31], [32]. A M2S090TS SmartFusion2 evaluation kit is priced at 384 € [34]. Therefore, this device provides the necessary services to function as a HSM replacement, with a lower cost and the flexibility to include FPGA and software implementations to overcome its limitations. It does not offer the same performance and specialized services as other HSMs, such as smart card support, but that is to be expected for a lower cost device.

The board provides encryption and authentication with AES and HMAC, a secure storage service for keys, and shared key generation with ECDH. It provides tamper detection capabilities, secure boot and zeroization. It has the necessary functionalities to provide authentication and confidentiality to communications, which is the main requirement of the system. The developed prototype uses the board's services, except for the HMAC software implementation.

The developed key management solution allows the periodic replacement of keys. Therefore, communications with new entities, with similar devices, can be established. Users must authenticate themselves to the device, using its authentication PIN. This way entities can manage which users have access to the device, on behalf of the entity. Whenever a tamper attempt or error is detected, the device does not accept any more API calls, and can be completely erased. This mitigates potential physical attacks and data leaks. A PKCS#11 API was developed to expose the device's functionalities and increase device interoperability.

Summary

In this chapter, the implemented system performance was tested. Starting with the communication channel performance, then the SmartFusion2 security accelerators and implemented services. In order to accurately predict the performance of the studied services, performance models were calculated from the performance results. These models allow anyone using this device to accurately study and predict the performance of services, implemented on the board. Thus, this chapter provides a robust study of the SmartFusion2 board, its services characteristics, performance, as well as possible services to implement, feature trade offs and efficiency concerns.

7 Conclusion

Contents

This work focused on evaluating the security features of the Microsemi SmartFusion2 board, modelling its performance, and developing a proof of concept system to secure communication channels between devices.

7.1 Overview

The SmartFusion2 SoC provides a varied array of security services using symmetric keys: AES with 128 and 256 bit key encryption, SHA-256, HMAC authentication with 256 bit SHA and a SHA based key derivation function. For asymmetric cryptography it offers ECDH for key generation, and additional ECC primitives, with the P-384 NIST defined curve, which can be used to implement digital signatures. Lastly, a true random number generator, a PUF based secure storage solution, tamper detection capabilities with several detection flags, and a zeroization feature with multiple recoverability options.

The prototype implemented on the device, focused on the implementation of a service which provides authentication and encryption to a TCP channel, using symmetric keys. The system is able to encrypt up to 36 KB of data using the AES and HMAC accelerators, with adequate 256 bit security. This limitation is imposed, due to the limited 80 KB of RAM. This hurdle was overcome by implementing a continuous authenticated encryption service, using an HMAC software implementation, and taking advantage of the characteristic of the AES CBC mode.

A key generation service using asymmetric key pairs was also implemented. It generates a shared secret, using an internal private key and a public key. The system also has a limitation in its ECC primitives. It does not provide an ECDSA implementation. In order to implement signatures, a big numbers library must be included in the device. The inclusion of such a library is complex due to its tendency to be heavy in code space and the device's limited RAM memory, even when disabling error correction and detection (80 KB).

Regarding key management, the PUF secure storage service is limited to 1000 write cycles for a predicted twenty year lifespan. To mitigate this, a key management service was developed, which stores the keys encrypted in a non volatile memory using the PUF service. This allows the update of multiple keys with only one PUF write for each update, instead of one write per key.

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. Lastly, the implemented prototype provides solid groundwork for a secure communications service, in a low cost HSM device.

7.2 Future Work

This work can be improved by implementing fully working digital signatures. To achieve this, a lightweight big integer library, with all arithmetic operations in the ECDSA algorithm is needed. It should support 48 byte integers, and have a maximum memory footprint of around 50 KB, to fit in the 80 KB of RAM with error detection and correction disabled, while keeping the secure communication and key management services enabled.

Another potential improvement to the SmartFusion2 is to study other software and FPGA cores, compared to the AES, HMAC and SHA SoC cores, regarding their side-channel protection and performance. There is potential for including other FPGA cores, with better scaling performance in regards to increasing data sizes, which have side-channel protection.

Regarding the secure communications prototype, the data transmission performance between the computer and the SmartFusion2 SoC can be improved by using the available USB connection which provides a much higher throughput rate. Additionally, the secure data exchange service only acts as a middleman to secure communications. Ideally, the secure data exchange service would secure a real time connection between two individuals. This can be achieved, e.g., by including a TCP or TLS library directly in the board, in order to allow two similar devices to establish a direct and secure connection.

Bibliography

[1] Q. H. Dang, "Secure hash standard," Tech. Rep., 2015.

[2] J. Rizzo and T. Duong, "Practical padding oracle attacks." in *WOOT*, 2010.

[3] P. Rogaway, M. Wooding, and H. Zhang, "The security of ciphertext stealing," in *International Workshop on Fast Software Encryption*. Springer, 2012, pp. 180–195.

[4] P. Rogaway, "Evaluation of some blockcipher modes of operation," *Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan*, 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] 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.

[8] M. Fiskiran *et al.*, "Workload characterization of elliptic curve cryptography and other network security algorithms for constrained environments," in *2002 IEEE International Workshop on Workload Characterization*. IEEE, 2002, pp. 127–137.

[9] D. Pointcheval and J. Stern, "Security arguments for digital signatures and blind signatures," *Journal of cryptology*, vol. 13, no. 3, pp. 361–396, 2000.

[10] U. Maurer, "Modelling a public-key infrastructure," in *European Symposium on Research in Computer Security*. Springer, 1996, pp. 325–350.

[11] S. Delaune, S. Kremer, and G. Steel, "Formal analysis of pkcs# 11," in *2008 21st IEEE Computer Security Foundations Symposium*. IEEE, 2008, pp. 331–344.

[12] T. Feller, *Towards Trustworthy Cyber-Physical Systems*. Wiesbaden: Springer Fachmedien Wiesbaden, 2014, pp. 85–136.

[13] S. Drimer, "Authentication of fpga bitstreams: Why and how," in *International Workshop on Applied Reconfigurable Computing*. Springer, 2007, pp. 73–84.

[14] U. Farooq, Z. Marrakchi, and H. Mehrez, "Fpga architectures: An overview," *Tree-based heterogeneous FPGA architectures*, pp. 7–48, 2012.

[15] T. Dorta, J. Jiménez, J. L. Martín, U. Bidarte, and A. Astarloa, "Overview of fpga-based multiprocessor systems," in *2009 International Conference on Reconfigurable Computing and FPGAs*. IEEE, 2009, pp. 273–278.

[16] 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.

[17] Microsemi. (2019) User guide smartfusion2 and igloo2 fpga security and best practices. Last visited 2021-05-27. [Online]. Available: https://www.microsemi.com/document-portal/doc_download/132037-ug0443-smartfusion2-and-igloo2-fpga-security-best-practices-user-guide

[18] ——. (2018) Ds0128 datasheet igloo2 fpga and smartfusion2 soc fpga. Last visited 2021-05-27. [Online]. Available: https://www.microsemi.com/document-portal/doc_download/132042-igloo2-fpga-datasheet

[19] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*. Springer, 1999, pp. 388–397.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

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

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

[28] J. E. Martina, T. C. S. de Souza, and R. F. Custodio, "Openhsm: An open key life cycle protocol for public key infrastructure's hardware security modules," in *European Public Key Infrastructure Workshop*. Springer, 2007, pp. 220–235.

[29] 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.

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

[31] 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

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

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

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

[35] 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.

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

[37] 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.

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

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

A

Implementation Details

PKCS#11 Function	**Arguments**	**Description**
`C_EncryptInit`	Key Object handle	Initializes continuous AES encryption and HMAC generation
`C_EncryptUpdate`	Plaintext Chunk Chunk Length Ciphertext Buffer Output Chunk Length	Encrypts and generates MAC from data block
`C_EncryptFinal`	Buffer Output Chunk Length	Finishes encryption operation, returns the generated MAC
`C_DecryptInit`	Key Object handle	Initializes continuous AES decryption and HMAC verification
`C_DecryptUpdate`	Ciphertext Chunk Chunk Length Plaintext Buffer Chunk Output Length	Decrypts and generates MAC from data block
`C_DecryptFinal`	Buffer Output Chunk length	Finishes decryption operation, returns MAC verification status
`C_DeriveKey`	Peer's Public key Salt value	Generate and derive new symmetric key, store it in device
`C_UnwrapKey`	Key Set Set Size	Import set of symmetric keys into device

Table A.1: PKCS#11 API to access the SmartFusion2 implemented services

B

Performance Graphics

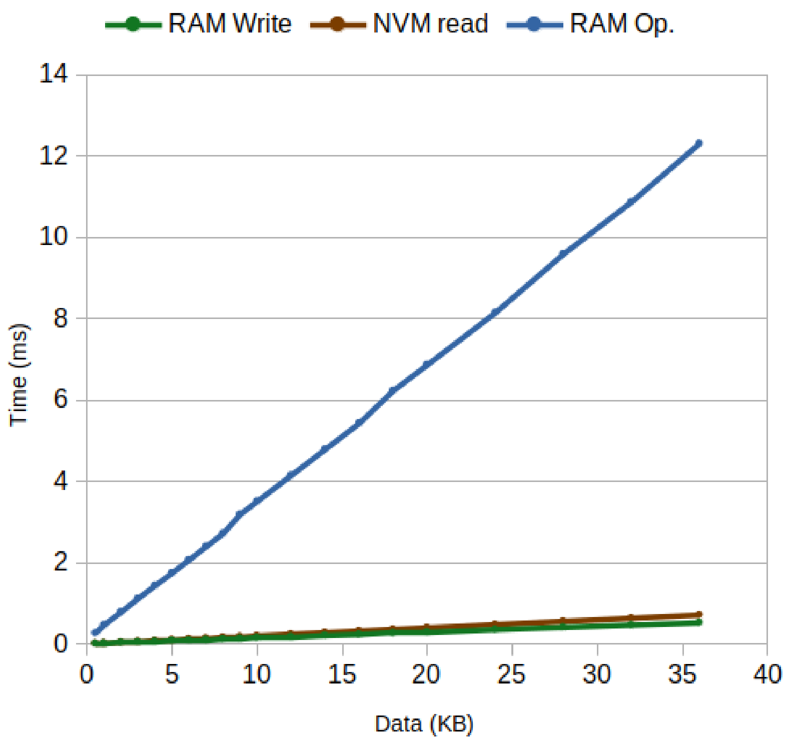


Figure B.1: Performance comparison of RAM and eNVM read operations up to 36 KB

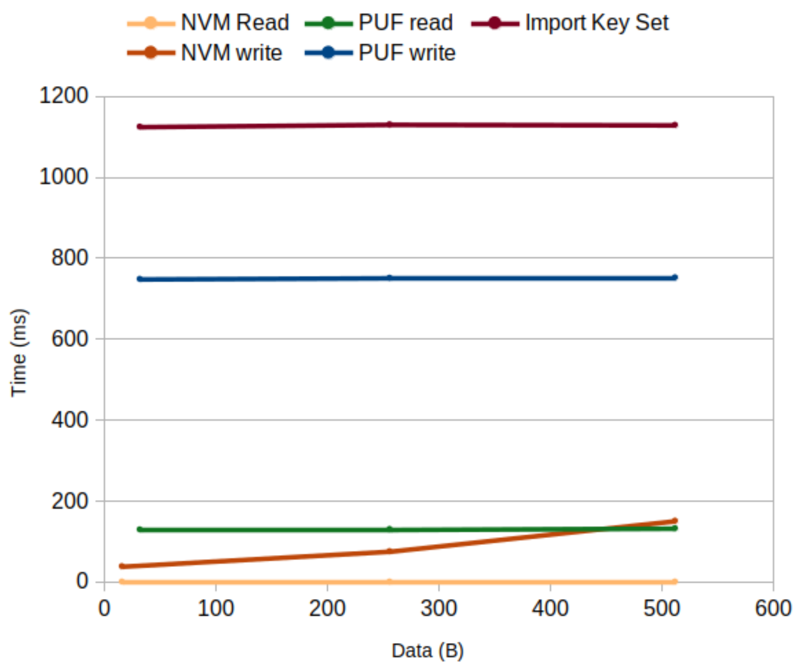


Figure B.2: Performance comparison of PUF and eNVM read and writes, along with the key importation service

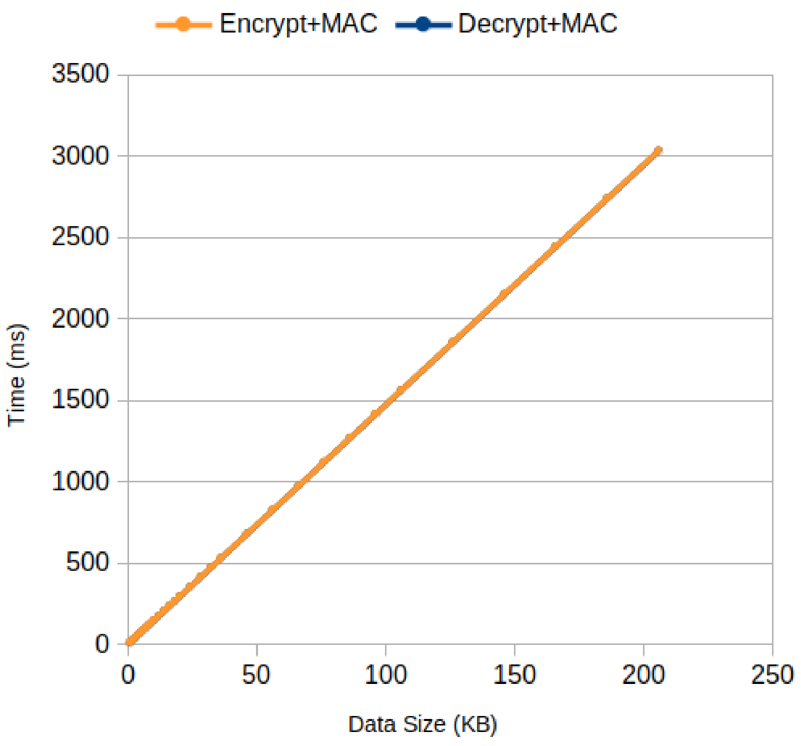


Figure B.3: Performance comparison of continuous encryption and MAC and continuous decryption and MAC