Hardware-Secured System for Secure Communications and Message Exchange

Alexandre Valente Rodrigues

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Ricardo Chaves

Month 2021

Acknowledgments

I would like to thank my parents for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible. I would also like to thank my grandparents, aunts, uncles and cousins for their understanding and support throughout all these years.

Quisque facilisis erat a dui. Nam malesuada ornare dolor. Cras gravida, diam sit amet rhoncus ornare, erat elit consectetuer erat, id egestas pede nibh eget odio. Proin tincidunt, velit vel porta elementum, magna diam molestie sapien, non aliquet massa pede eu diam. Aliquam iaculis.

Fusce et ipsum et nulla tristique facilisis. Donec eget sem sit amet ligula viverra gravida. Etiam vehicula urna vel turpis. Suspendisse sagittis ante a urna. Morbi a est quis orci consequat rutrum. Nullam egestas feugiat felis. Integer adipiscing semper ligula. Nunc molestie, nisl sit amet cursus convallis, sapien lectus pretium metus, vitae pretium enim wisi id lectus.

Donec vestibulum. Etiam vel nibh. Nulla facilisi. Mauris pharetra. Donec augue. Fusce ultrices, neque id dignissim ultrices, tellus mauris dictum elit, vel lacinia enim metus eu nunc.

I would also like to acknowledge my dissertation supervisors Prof. Some Name and Prof. Some Other Name for their insight, support and sharing of knowledge that has made this Thesis possible.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

To each and every one of you – Thank you.

Abstract

Individuals with high responsibility jobs such as government officials, top level company executives and diplomats are high profile targets to digital attacks, since they manage very sensitive information. Thus, attacks can have very damaging consequences for them and organizations. To maximize security, it is in their best interest to avoid storing cryptographic keys, passwords and perform critical cryptographic operations in their personal computers. This thesis proposes a cheap, relatively efficient but highly secure physical personal system, in a client-server mode, which enables individuals to securely exchange messages and sensitive documents. The proposed system secures communication by providing confidentiality and authentication to messages. This system will be responsible for performing every cryptography operation, store and manage cryptographic keys. All operations are performed inside the device and keys are never exposed to the outside, in order to not jeopardize the security of the communications.

Keywords

Communication Security; Secure Physical Device;Confidentiality;Authentication.

Resumo

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Aliquam aliquet, est a ullamcorper condimentum, tellus nulla fringilla elit, a iaculis nulla turpis sed wisi. Fusce volutpat. Etiam sodales ante id nunc. Proin ornare dignissim lacus. Nunc porttitor nunc a sem. Sed sollicitudin velit eu magna. Aliquam erat volutpat. Vivamus ornare est non wisi. Proin vel quam. Vivamus egestas. Nunc tempor diam vehicula mauris. Nullam sapien eros, facilisis vel, eleifend non, auctor dapibus, pede.

Palavras Chave

Colaborativo; Codificaçãoo; Conteúdo Multimédia; Comunicação;

Contents

List of Figures

List of Tables

Acronyms

AES	Advanced Encryption Standard
API	Application Program Interface
AD	Associated Data
AEAD	Authenticated Encryption with Associated Data
CBC	Cipher Block Chaining
CBC-MAC	Cipher Block Chaining Message Authentication Code
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
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

MSS	Microcontroller Subsystem
NIST	National Institute of Standards and Technology
NRBG	Non-Deterministic Random Bit Generator
NVM	Non-Volatile Memory
OFB	Output Feedback
OMAC	One-key MAC
PGP	Pretty Good Privacy
PIN	Personal Identification Number
PKCS	Public-Key Cryptography Standards
PKI	Public-Key Infrastructure
RSA	Rivest-Shamir-Adleman
SEU	Single-Event Upset
SHA	Secure Hash Algorithms
SoC	System-on-Chip
SRAM	Static Random-Access Memory
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 have access to a computer, involved in many everyday tasks, such as, web browsing, communications, social networks, news, 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. This is of great concern to people with high responsibilities from their jobs, who deal with sensitive information, such as, government officials, top level company executives and diplomats. Suffering an attack to a personal computer can be highly damaging as it can carry severe consequences for companies and countries. In addition, high profile officials who deal with sensitive information are more likely to be targeted by attackers.

1.1 Problem

New attacks, targeting computers, are discovered daily. They can come from zero-day vulnerabilities, phishing scams and many others, the opportunities are endless. It is impossible to predict and protect against all. Communications security, depends on the cryptography keys and passwords used. These are usually stored, along with other sensitive information, in the user's computer. Instead of storing the data in the computer, a more optimal solution, meaning, harder to compromise the security of communications, is to separate the platform used by the user for communications (their computer), and the device responsible for managing, securing communications and storing sensitive data. The goal is to add another layer of security, to make it difficult to compromise security even if the user's computer is compromised. A secure and independent solution is needed to establish secure channels of communication, store keys and perform critical operations, even if the computer might be compromised. A possible approach is the utilization of a personal physical device that is responsible for storing digital keys and perform critical operations. These devices need to be highly secure and independent from the user's personal computer.

1.2 Requirements

In order to address the problem and using the discussed approach, the implemented solution will have several requirements, to allow secure communications between multiple entities. It should perform all critical operations to the security of the communications, as well as, store all relevant secrets to the security of the interactions. A design requirement of the system is it should be easily usable to the regular user, with no technological expertise. The system must be efficient and low-cost, as so it is more easily accessible and scalable by interested users.

1.3 Document Structure

This first chapter introduces the context, the problem and basic requirements of the system. The second chapter will cover the technical background needed to comprehend the solution and state of the art. The third chapter goes into detail about the problem, its entities, devices, extensive requirements, and possible use case scenarios. The fourth chapter covers the architecture of the solution including components and the operations. The fifth chapter covers the developed protocols and implementation. The sixth chapter evaluates the system's performance and fulfilled requirements. The last chapter outlines the conclusions of the developed system and provides guidelines for future work.

2 Background and Related Work

Contents

This section goes into detail on the necessary concepts required to perfectly understand the problem, the proposed solution and the rationalization process behind it. It starts by providing an overview of cryptographic services, primitives and protocols. Then it presents several general purpose computing systems and ends by presenting other relevant components.

2.1 Cryptography

There are several cryptographic services relevant to this work, namely: Confidentiality: used to hide the content of information from unauthorized entities; Data Integrity: ability to protect from unauthorized modification of data; Authentication: used to ascertain 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. The keys are smaller and the operations are faster than with asymmetric keys.

Asymmetric keys constitute a pair for each party, one private and one public key. The private key is personal to its owner and should never be shared. The public key may be shared widely to other parties. Asymmetric keys are bigger and the operations are slower compared to symmetric key algorithms.

There are two types of ciphers regarding the procedure: stream and block ciphers. Stream ciphers generate an infinite stream of pseudo-random bits as the key, know as key-stream. The stream is used to encrypt, usually 1 bit of plaintext at a time. The operation to combine the key-stream and plaintext is an exclusive-or (XOR). Stream ciphers are usually faster than block ciphers, have lower memory requirements and thus are more suitable to embedded devices with limited memory. However they are prone to weaknesses if not implemented correctly, in particular, using the same Initialization Vector (IV) more than once.

Block ciphers encrypt fixed-length groups of bits, called blocks, with a symmetric key. They have a higher memory usage, in order to keep the blocks in memory. Since the plaintext is encrypted one block at a time, if the plaintext length is not a multiple of the block size, the last block needs to be padded. Another caveat of blocks ciphers is, they are more susceptible to noise in transmissions. If a bit is flipped with a stream cipher, only the corresponding bit is affected. While with a block cipher, more than 1 bit is affected, depending on the mode.

Symmetric ciphers support both block and stream ciphers while asymmetric use block ciphers.

2.1.1 Hash Functions

A cryptography hash function generates a fixed dimension value (digest) based on variable 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 altered. To achieve this, hash functions must have several properties, more notably: the same input value always results in the same hash, very different outputs must be generated for very similar inputs, it should be hard to find two messages which generate the same hash and it should be hard to find n 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 several flavours. SHA-2 provides different functions which vary on the size of the digest. 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 use symmetric keys and are frequently used to achieve data integrity, authentication and confidentiality.

The Advanced Encryption Standard (AES) is one of the most popular symmetric-key algorithms and its different modes of operation. It uses 128-bit blocks for block cipher modes and the keys can be 128, 192 or 256 bits. AES has both block and stream cipher modes. Relevant modes for this work, 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 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 this is not done correctly, it can be exploited with a padding oracle attack [2]. Implementing ciphertext stealing, resolves the issue and is recommended for the security of CBC [3]. Encryption is not parallelizable, since a ciphertext block depends on all the blocks before it. Another disadvantage of CBC is it cannot precompute data to improve encryption performance. To decrypt a ciphertext block, only the previous ciphertext block is needed. Therefore random read access is supported and decryption can be parallelized. Regarding error propagation, when a bit is flipped in the ciphertext, the plaintext block will be completely corrupted and the corresponding bit of the next block will be inverted.

The **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. It is effectively a stream cipher. The downside of needing to encrypt the IV multiple times, is the encryption and decryption are not parallelizable and random read access is not possible. However, the multiple encryptions of the IV can be precomputed in order to increase the performance of both encryption and decryption. Changes to a ciphertext block, only affect the corresponding bits.

Counter (CTR) mode concatenates an IV with a counter beginning at 0. Each sequence is encrypted and applied 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, the encryption operation is exactly the same as the decryption and an error affects only the respective bits. However it does not have the performance disadvantages of OFB. Encryption and decryption are parallelizable, random read access and preprocessing are both possible. It is worth noting that due to existing no need to implement decryption, it can save hardware costs and simplify code.

For every mode with an IV, it needs to be sent along with the message to the receiver, or the receiver will not be able to retrieve the entire message.

CBC and OFB modes are proved secure, assuming the IV is random, and is unique, meaning it is only used once for each key and message. For CTR mode, the IV does not need to be random, but it cannot be reused with the same key. After the IV is used, there is no need for the value to be kept secret. It can be sent alongside the ciphertext.

Regarding performance, the paper [4] states CBC is slower than CTR mode. OFB is even slower. When efficiency characteristics matter, nothing comes close to CTR: it has better performance characteristics, in multiple dimensions, than any of CBC, and OFB.

All these cipher modes are malleable, meaning an attacker can modify a ciphertext C, the result of encrypting plaintext P, to create ciphertext C' which will decrypt to plaintext P' that is similar to P. Malleability is connected to message integrity. This is not considered a relevant weakness since the modes only goal is to offer confidentiality guarantees, not integrity. If one wishes integrity it should pair one of these modes with a Message Authentication Code (MAC), or 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, receives the message and a symmetric key, to generate a tag. Unlike digital signatures, MAC do not offer non-repudiation since it uses a symmetric key, which need to be distributed to all parties. Any of the users in possession of the key can generate a MAC for a message, as well as verify it. On the contrary, digital signatures utilize the private key from asymmetric cryptography, which is personal.

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 generating the authentication tag. CBC-MAC security deficiencies were resolved with One-key MAC (OMAC), which is secure for variable-length messages.

Hash-based Message Authentication Code (HMAC) is different from the previous techniques, by using 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 considered secure. Therefore SHA-2 is a good option. Despite CBC's inefficiencies discussed in section 2.1.2, specifically, each block is serially encrypted, HMAC is slower due to the inefficient hashing operations. However, HMAC does not have the security problems of CBC-MAC. HMAC 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 both confidentiality and authenticity using only symmetric keys. They may be more efficient than combining separate privacy and authentication techniques, such as the ones discussed in earlier sections, and are less likely to be used incorrectly. AEAD schemes also allows associated data to be included in the message, which is authenticated but not encrypted. This feature is useful, for example, for network packets. The header is visible but is authenticated. The payload is both authenticated and encrypted.

CCM is an AEAD mode that combines CBC-MAC for authentication and CTR for encryption. CCM uses a MAC-then-Encrypt approach. First, CBC-MAC is computed on the message to obtain the tag. Then the message and the tag are encrypted with CTR mode. Due to performing two encryption operations, CBC-MAC and then CTR, it is a less efficient mode 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. It is only considered secure for fixed-length messages. Despite being a slower mode, it is secure and achieves its goals, so it is widely supported, 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. Evidently, like in normal CTR mode, it needs a different IV for each encrypted message. 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 64GiB 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 utilizes a pair of public and private keys. It is commonly used to provide confidentiality, data integrity, authentication and non-repudiation. The private keys must always remain secure with the owner. Public keys may be distributed as it does not compromise security. 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 on account of only the owner is in possession of the private key. These two different concepts can be combined to provide confidentiality, authentication and non-repudiation, through digital signatures, to a message.

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. The scheme 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, it 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. Gupta & Silakari, 2011 [6], give as an example a 160-bit ECC key has similar security to a 1024-bit RSA key. ECC operations are also faster, this makes elliptic curve based schemes more suited for less powerfull and memory constrained devices [7]. With the threat of quantum computers, both ECC and RSA could become obsolete in the future, as it is vulnerable to brute force attacks from such devices.

2.1.6 Elliptic-curve 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 key to secure communications. Both parties compute the secret from from publicly exchanged integers, and private integers. Attackers listening on the exchanged public integers cannot compute the same secret, since both parties 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 is a standard scheme for authenticating documents or digital messages and ensuring the signer cannot repudiate the signature. The digital signature is generated by a combination of asymmetric keys and hash functions. The 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 the message, which has a fixed size. Third parties can validate the signature with the author's public key. Only the author, in possession of their private key, could have generated the signature. They are a digital version of handwritten signatures [9], commonly used anywhere forgery detection is essential, for instance in financial transactions or software distribution.

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 never leave it. 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 Public Key Infrastructure

Asymmetric cryptographic 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 certain identity. A PKI has several components e.g. a registration authority, a certification authority and a central database of stored keys. A user can also submit other entities' public keys. Other entities that trust the user responsible for the submission, can use the public keys to authenticate messages. There are alternative approaches to PKI, such as a web of trust. This mechanism self-signs certificates and third parties attest these certificates. This approach is implemented in Pretty Good Privacy (PGP) [10].

2.1.9 Transport Layer Security

TLS is a cryptographic protocol that aims to provide confidentiality and data integrity, during transmission, over TCP/IP [11]. It uses symmetric cryptography to encrypt data. A new symmetric key 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, unlike PGP, assuring any past connections are secure, if in the future encryptions keys are disclosed.

2.2 Other Important Concepts

There are some important concepts to consider involving cryptography and secure cryptoprocessors.

2.2.1 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 almost all 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 microscopic physical conditions. These conditions are completely unpredictable. For this reason, TRNGs are perfect for use in cryptography and secure cryptoprocessors.

2.2.2 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 defines an Application Program Interface (API) designed to interface between applications and cryptographic devices such as smart cards and hardware security modules [12]. 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, it allows devices to take advantage of another device's API. It allows applications to access cryptographic devices through slots. The slots represent a socket or device reader. A session can be established through the slot so the application can authenticate itself to a token with a default PIN. The token holds private and public objects which can be keys and certificates among others, and can be accessed by user's.

2.3 Secure Cryptoprocessors

In this section we discuss some computing systems that may be pertinent to this work.

Secure cryptoprocessors are dedicated physical computational devices for performing cryptographic operations. Some secure cryptoprocessors namely, Smart Cards, Trusted Platform Modules and Hardware Security Modules will be discussed next.

2.3.1 Hardware Security Modules

A Hardware Security Module (HSM) is a high grade computational device responsible for storage, management and generation of cryptographic keys, as well as performing 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, support several cryptographic operations, random number generators and have fail-safe mechanisms in place, in case of an attack, e.g., deletion of keys. Some devices have accelerated Central Processing Unit (CPU) and optimizations to improve operations' performance. These modules are usually costlier than other computational systems i.e. Trusted Platform Module (TPM)s, but are more advanced in processing power and available operations.

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 of the device and easy replacement if needed. However, the disadvantage is it makes it easy for attackers to acquire many devices to try to tamper with. They have a low processing power, and small memory which only allows to store a small amount of data. To be authenticated, the cards need readers that are either contact or contactless (RFID technology). These characteristics make them extremely popular, used in many industries, such as, retail, healthcare, communication and government.

2.3.2 Smartfusion2 SoC

An example of such device is the SmartFusion2 System-on-Chip (SoC) from Microsemi that delivers more resources in low-density devices with the lowest power, proven security, and exceptional reliability [13]. Smartfusion2 integrates a non-volatile Field-Programmable Gate Array (FPGA) with a SoC and an internal Non-Volatile Memory (NVM) of 512 KB for storing Phase 0 boot code. Since they are non-volatile, the bitstream is at a lower risk of being probed during boot [14]. It has a Static Random-Access Memory (SRAM) with 64 KB + 80 KB, and is protected against Single-Event Upset (SEU). Smartfusion2 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-256, SHA-256 and ECC, among others.

The AES system service supports encryption and decryption with ECB, CBC, OFB and CTR modes. It offers two services, for 128 bit and 256 bit keys. It provides a SHA-256 hashing service, and a HMAC service using the same SHA-256 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 from a root key and a salt or in challenge-response protocols.

The Non-Deterministic Random Bit Generator (NRBG) service provides a raw entropy source based upon a noisy ring oscillator, measured against a system clock. It can be generate true random numbers from the unpredictable physical conditions of the board.

The device includes several relevant ECC accelerators. It supports the National Institute of Standards and Technology (NIST) defined P-384 elliptic curve. The device can generate new 48 byte private ECC keys. The corresponding public key can be calculated with scalar point multiplication service, by multiplying the scalar (private key) with a point (base point). This service also allows the generation of secrets 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 an elliptic curve digital signature algorithm such as ECDSA.

The SRAM-PUF service allows the user to store extrinsic keys (value supplied by the user) and intrinsic keys (randomly generated in the device). 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 or protect cryptographic keys with 256 bit security strength [15]. When power is removed, the secret disappears. There is not technology to detect the start-up behaviour of an SRAM. 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 flash memory. It has a size of 512KB, with the top 64 pages reserved for design security (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 [16].

The side-channel Differential Power Analysis (DPA) is an advanced power analysis technique to compute values from statistical analysis of multiple cryptographic operations [17]. Not all services in the smartfusion2 board are DPA-resistant. ECC point multiplication, NRBG, SRAM-PUF and KeyTree are. On the other hand, AES, SHA-256 and HMAC accelerators only 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 cipher text. For the non DPA-resistant services, there is a danger of key extraction if the same key is used repeatedly. Although to setup up this type of attack, physical access to the device is needed with an oscilloscope to measure the voltage [18]. Even then, the device has tamper detection mechanisms so its services can be blocked, or the device is completely reset and deleted of any sensitive information through zeroization.

Secure boot

Not validating code before its execution leads to potentially executing untrusted code. This causes problems such as hackers inserting malware to hijack systems, download intellectual property, spy on users or perform any number of attacks. 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 can be write protected. Another reason is 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 TPM, compliant with version 1.2, into the processor. 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 allows the customer to specify a public key as a root of trust to be fused into the device. This key is used to authenticate other keys that can be used to authenticate software components.

2.4 State of the Art

In general, HSMs have been used in specific applications, exploiting their cryptographic services, key storage and physical protections.

Lesjak et al. [19] proposes a system to secure remote snapshot acquisition, between the vendor and customers, by attaching a hardware security module to the customer product. 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, the diffie-hellman based ECMQV algorithm for key establishment using ECC keys. This controller also contains protections against side-channel attacks such as DPA and physical manipulation.

Seol et al. [20] isolates critical operations and data from cloud administrators by implementing a HSM to isolate the cryptographic keys and data.

Wolf et al. [21] implemented a HSM on a 663€ Xilinx Virtex-5 FPGA to secure e vehicles network communications. The cryptographic algorithms were implemented on the FPGA by the authors, AES-128 bit and ECC 256 bit. The board was connected to a microcontroller running linux with additional algorithms available from a cryptographic library. Secure coprocessors have also been used to store and manage symmetric keys for data encryption in systems which manage biomedical data [22]. Existing HSMs capabilities and functionalities have been studied by some studies. Kehret et al. [23] studied two specific boards. VaultIC460 is 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 secure memory for key storage and a USB connection.

ATECC508A from microsemi is a tiny security controller with asymmetric key cryptography algorithms: ECDSA and ECDH, along with SHA, a TRNG and storage of up to sixteen 256 bit keys. These type of controllers are not suitable for this work, it is very small and limited with only a single GPIO, and no symmetric key algorithms, preventing encryption of large amounts of data. They are designed to be added to IoT devices.

3 Problem Definition

Contents

This chapter will define the problem, the client requirements and list the necessary functionalities of the solution, to successfully address the problem. First some context surrounding the problem is given. Next, the profile of the target clients will be described, and some examples will be given. The following section contains a list of client requirements the solution must abide by. Then it will shed the light on some essential concepts in order to understand the operations that need to be implemented. It will end by detailing the client use cases it must provide.

3.1 Context

As discussed before, the same computers commonly used for communications and information storage are exploitable by attackers, and can cause a minor inconveniences, to possibly severe repercussions, such as, losing your confidential data to malicious parties.

An interesting approach to improve security is to add another layer of security to confidential data and communications through the addition of an device, independent of the user's personal computer. The device is responsible for the security of sensitive data and communications.

3.1.1 Entities

These type of devices are especially relevant to people high responsibility jobs, that handle very sensitive information, which have dire consequences if they are lost, corrupted or leaked. Some examples are government officials who handle confidential information pertaining to a country, company executives, such as the CEO who have access to company secrets, diplomats who manage confidential treaties, and military officers who have access to information critical to a countries' security.

Additionally, not just individuals have interest in these systems, a device can be assigned to a group of people representing an entity. For example, in the armed forces, a device can be assigned to the navy, one to the infantry, and every other faction. Any ranked officer, or people with a certain level of authority, could use the entity device, to communicate with other people or entities, in behalf of the group.

3.1.2 Devices

There are currently on the market some dedicated devices designed to secure communications and save private data. These type of devices have physical tamper-resistant measures against attackers who wish to read the device's information. They also provide fail-safe mechanisms in case of an attack. HSM is a high grade device, with more computational power and larger storage capacity for the user's secrets.

Smart Cards, provide secure and portable tamper-resistant storage. They have lower processing power, and smaller memory which only allows to store a small amount of data. They have a low-cost, so can be produced in bulk and easily replaced. Only an RFID card reader is needed to read its information, and verify the owner's identity. Because of these features, they are widely used in the retail, healthcare, communication and government industries.

3.2 Requirements

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

- Devices should be distributable to individuals or entities with one or more individuals;
- The system must allow communications between individuals representing themselves or an entity;
- The system must be responsible for securing all communications against any sort of attacks;
- The device should be independent from user's personal computers;
- Users should be able to create secure communications with available and new entities;
- It should provide an easy-to-use interface by everyone, including 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 client requirements need to be translated into slightly more technical and tangible requirements a solution must follow. In order to secure communications, the following services must be guaranteed: confidentiality, integrity and authentication. With asymmetric keys the system can provide non-repudiation to documents or files, by means of qualified digital signatures.

The device must store all keys related to the entity who owns the device. The device must support secure storage in order to store the user's sensitive information, such as the cryptographic keys. These keys must never be exposed to the outside environment of the device to ensure the security of communications and independence of the system. All cryptographic operations must also be performed inside the device. 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 extremely difficult or even impossible to extract any information from it. The solution should work with a plethora of devices, which will increase the adoptability of the solution among clients.

The system should provide an application on the user's device, which will communicate with the physical device, and make the operation's available to the user through a simple interface for the regular non-tech savvy user. Another related requirement is the usage of a common connection solution, e.g. Universal Serial Bus (USB) cable, to further increase the pool of supported devices. In addition, the system should perform the operations in a reasonable time to minimize the user's wait, and improve the user experience.

3.3 Use Case Scenarios

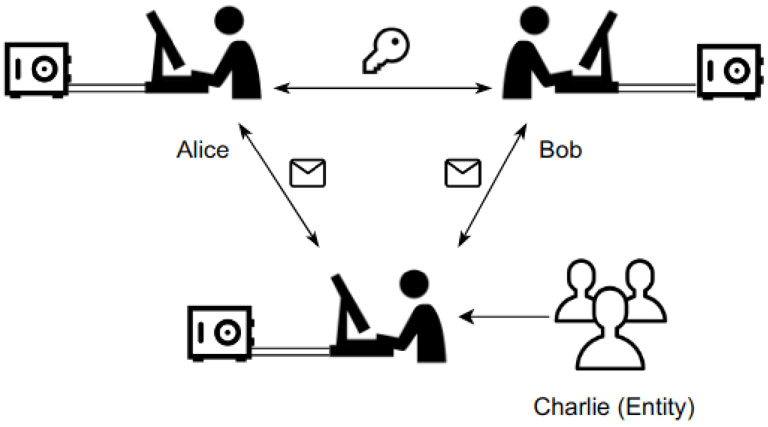


Figure 3.1: System Usage Example

This section will define and detail the use cases the solution must satisfy and provide to the user. The combination of the use cases satisfy the client requirements in Section 3.2. Figure 3.1 will be used as an example and illustration for the different scenarios.

3.3.1 Authenticating the User

Every user must authenticate himself, before using the device. This can be done by providing a PIN, which the device will verify before unlocking the session for the user. The device will come from fabric with a default authentication PIN. The users should be allowed to change this number. For **personal** devices, there is only one user, the owner, as illustrated by Alice and Bob in figure 3.1. There is only the single authentication PIN, which when sent to the device, unlocks the session, and the user can access its services. For **groups** and entities, there can be multiple users, illustrated by Charlie. In this case, there are different scenarios for authentication. The simplest is when it is not needed to authenticate the individual user, only the entity. There is only a single authentication PIN for the entity. All the user's with permission to communicate in behalf of the entity, must know the PIN. They only need to send the number to the device to be allowed access to its operations.

The second scenario, is when an entity wishes to authenticate each individual user with access to the device. This would entail a more complex process. A user will be assigned the role of administrator. For example, in the military, this could be assigned to the top ranking general. The administrator, using the administrator PIN, is able to register new users with their own private PIN, access the logs of which user logged in and when, as well as, which messages they sent and received. The registration of a new user can be done physically with both administrator and user present. Afterwards, the registered user can use their credentials to authenticate himself, and use the entities device to communicate. It is important to note that in all scenarios, only the users or 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 setup between two or more entities. For each configured communication, the same symmetric key is saved in secure storage on 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 data content.

These devices have a certain amount of secure storage, so there is a limit to the number of symmetric keys which can be stored there. In the simplest scenario, the system allows secure communications between a limited amount of entities. Each entity, upon ordering the device, can specify which entities it wants to communicate with. For example, Alice can request a device which enables two separate channels with Bob and Charlie. Alice can also request a single channel with all three entities, in this case, only one 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. This approach is not very flexible. It has a limit on the number of communications and does not allow for the users to communicate with new entities without returning the device to manufacturing.

Including asymmetric keys, allows an approach with improved features. Each device has one pair of asymmetric keys stored in secure storage, generated inside the device from fabric. These keys allow the storage of a practically unlimited number of symmetric keys in non-volatile memory. This gives the flexibility entities might need to communicate with the number of entities they wish. This feature is useful in other situations. There are cases where existing symmetric keys might need to be revoked, due to suspicion of being compromised, with new ones generated and shared. Another case is if the symmetric key has an expiration date, when it expires, new keys need to be exchanged.

The asymmetric keys generated inside the device, and never exposed to the outside, make the generation of qualified **digital signatures** possible. These strong signatures can legally represent the entity.

Finally, when using asymmetric keys, beyond providing authentication to the communications, the data sender can be authenticated to the receiver, using the entities private key stored in the device.

3.3.3 New Communications

Adding to the previous scenario with asymmetric keys, users will be able to establish secure communications with new entities. This is achieved by distributing a list of available entities to communicate with. When and individual, e.g. Alice, wants to establish communications with a new entity, such as Bob, she needs to get Bob's key from the received list. Then they can securely trade a new key, and begin communications.

The list can be provided at the manufacturing control station, when the device is delivered. Afterwards, there are two options to distribute the list. When the list is updated with new entities, a member of the entity will physically go to the station, to retrieve it. Alternatively, the entity will receive the device with the necessary keys to securely communicate with the control station. When an entity needs the list, it will be supplied for them by the control station. In this scenario, the control station can be thought of as a special entity. The list can be sent using an offline service such as email. Every time a new entity is added to the list, the station sends a new email with the updated list. Each entity can get access these updates when they need it.

4

Architecture

Contents

The objective of the system was to develop a device in a box format to enable users to establish safe channels of communication. This is achieved with a safe and secure device which is personal to each individual. In order to secure the communications between users, the device saves the user's sensitive data, such as keys, and performs all security critical operations. The system is designed so that each user has it's own physical box.

4.1 Components

The system architecture, depicted in figure 4.1, is composed of two main components: the physical box which responsible for securing the data, and the client software on the user's computer, which communicates with the box.

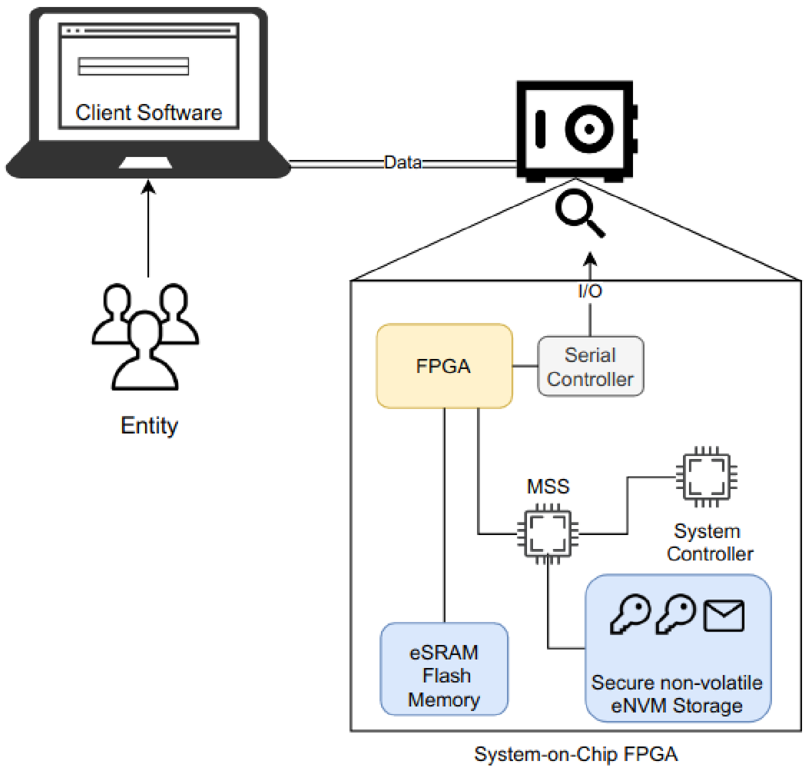


Figure 4.1: System Components

The client software sends and receives data through the device's serial I/O port, which exposes an API to access its operations. With this connection the entity can signal the device to perform the desired operations, through the client software. The device integrates a FPGA, Microcontroller Subsystem (MSS), secure eNVM storage and flash memory for configuration. The MSS has an embedded ARM processor, connected to the system controller which provides several cryptographic services. The secure eNVM allows storage of keys, data and secure boot code. The MSS uses the keys stored in the eNVM, with the cryptographic algorithms in the system controller.

4.2 Operations

This section will define and describe the system architecture. It is structured starting with the authentication and then the system operations, divided by types. The architecture will be explained, using the scenarios in section 3.3.

For the user to be able to perform operations, he first must authenticate himself to the device. The device will come from fabric with a default authentication PIN. To authenticate himself to the device, the user sends a PIN through the software, the device then compares it to the authentication number stored inside. To protect the number it can be either stored in the secure eNVM if there is enough space, or in other non-volatile memory, encrypted with the device's public key. Once authenticated, the session will be unlocked, and the user will be able to perform operations. If only the entity is authenticated, not the individual, a single PIN is used for all authentications. If instead the individual is authenticated, the user will send the registered name and the number, and the device will use the name to identify the correct number.

After authentication, the operations that can be performed, are split in three types:

- Administration operations configure authentication and communication parameters;
- The secure communication operations provide the cryptographic services to secure communications;
- Communication management operations manage the keys used to secure connections.

4.2.1 Administration

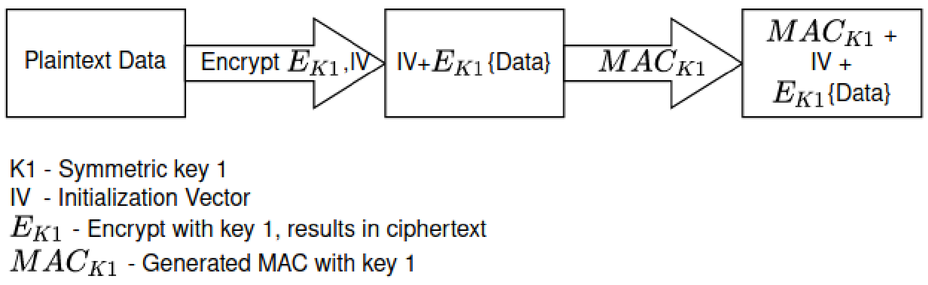
The administration operations will allow the user to manage the authentication related parameters. For the first scenario, the only operations of this type is to change the authentication PIN. The user sends the number to the device, and it will be securely saved inside it. The device will be initialized from fabric with a default PIN which must be supplied to the user. Before performing any operation the user should change his PIN to begin secure communications.

The second scenario has an administrator role and a user role. Each user has its own PIN. The administrator, beyond changing the authentication number, can register new users. To register a new user, both administrator and new user need to be physically together with the device. The administrator authenticates himself to the device, begins the registration process and allows the user to insert their name and PIN. After this the user can login with name and number, and access all secure communication operations, as well as changing their own PIN.

4.2.2 Secure Communication

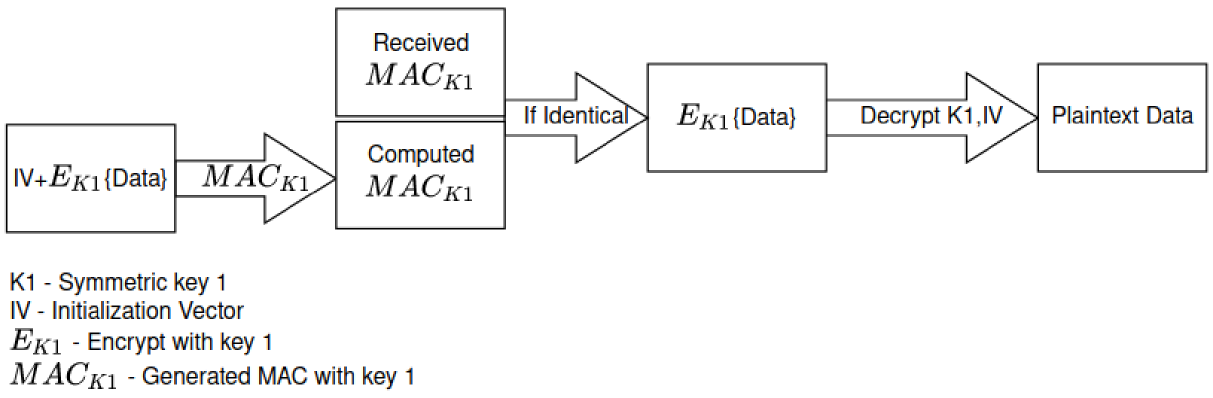
The main operations will be responsible to secure the communications between users. These operations will grant the confidentiality, integrity, authentication and non-repudiation services to communications.

Th objective of communications with **confidentiality** and **authentication** is to send and receive data securely, to and from the device. The user sends plaintext data, and the device returns it encrypted and authenticated with the symmetric key stored inside the device, of the corresponding connection. In the equivalent operation reversed, the ciphertext is sent to the device, and the plaintext is returned. Both operations are pictured in figures 4.2(a) and 4.2(b).



Plaintext Data
Encrypt E_{K1},IV
IV+E_{K1}{Data}
MAC_{K1}
MAC_{K1} + IV + E_{K1}{Data}
K1 - Symmetric key 1
IV - Initialization Vector
E_{K1} - Encrypt with key 1, results in ciphertext
MAC_{K1} - Generated MAC with key 1

(a) Encrypt and authenticate data with K1 key



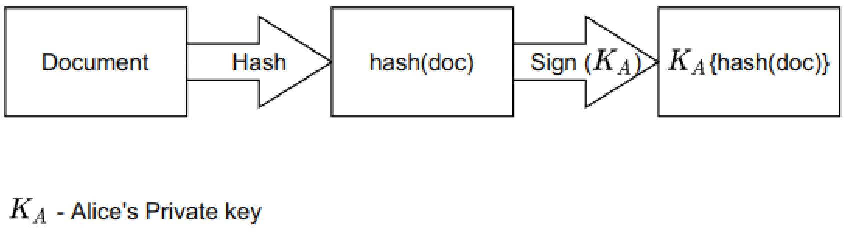
(b) Decrypt data and verify authentication with K1 key

Figure 4.2: Procedure to secure data with authentication and confidentiality

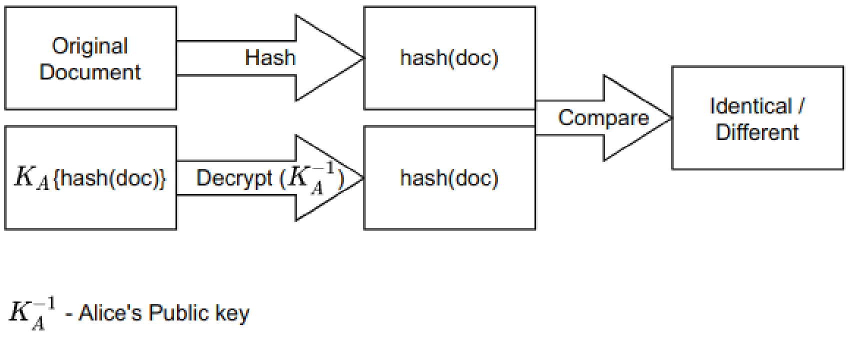
Beginning with figure 4.2(a). The plaintext data sent to the device is first encrypted with the symmetric key *K1*, using a randomly generated IV. Then a MAC is generated from the encrypted data and IV, using the symmetric key. All fragments of information (MAC, IV, ciphertext) are returned to the user, and the user can then send it to other entities in possession of the used *K1* key.

Following with figure 4.2(b), when a device receives a encrypted message, it computes the MAC of the received IV appended with the decrypted data. If the computed and received MAC are identical, the ciphertext is decrypted using the same *K1* key and the received IV.

Qualified digital signatures provide **non-repudiation** to a piece of data, using the private key generated inside the box. The user sends the data to the box, and the subsequent signature will be returned as pictured in figures 4.3(a) and 4.3(b).



(a) Alice generates signature of a document



(b) Bob verifies the signature of the document

Figure 4.3: Procedure to generate and verify a qualified digital signature

The signature is generated by calculating the hash of the data, and signing the digest with the device's private key. To verify a signature (figure 4.3(b)), the device decrypts the hash with the signer's public key. Next, it computes the hash of the original document, and compares both hashes. If they are identical, the qualified signature is valid.

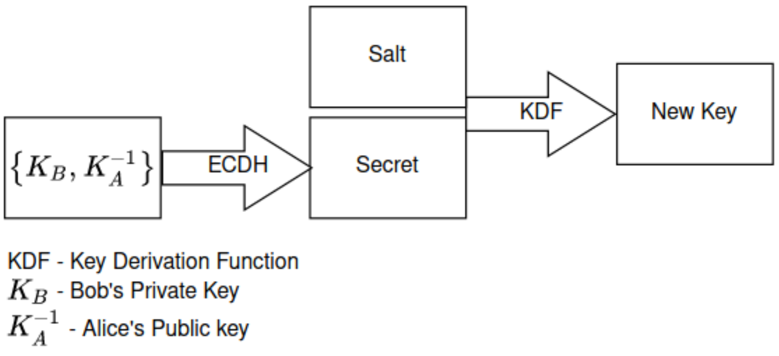
4.2.3 Communication Management

These operations will manage the necessary symmetric keys for secure communications and digital signatures. Supported management operations include generation of new symmetric keys for communication and revocation of existing keys saved in the device, if communications are suspected to be compromised. These operations are only available in the scenario where each device has a pair of asymmetric keys, and there is a protocol in place to distribute public keys.

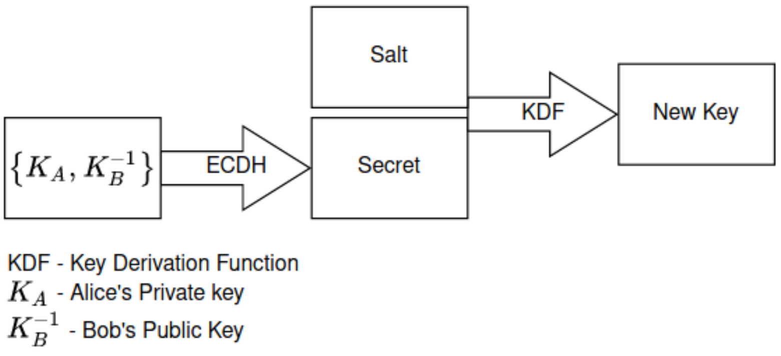
An entity receives their device, prepared to communicate with other entities, and a list of information of other entities, available to create communications. This information, namely the entities public key, can be imported into the device to generate new keys. In order for an entity to communicate with a another entity, with no previously established communications, their device will generate a new symmetric key and store it in secure storage, or in non-volatile memory, encrypted with the device's public key. Figures 4.4(a) and 4.4(b) illustrate the procedure Alice and Bob devices go through to generate new communication keys.

The procedure is the same from the perspective of both devices. Alice's device has its private key and the Bob's public key. Bob's device has its private key and Alice's public key. Both generate the same secret from these values using the ECDH algorithm, introduced in section 2.1.6. This secret then serves as input to a key derivation function to derive a new key. This function receives the secret and a public know parameter, a salt, to generate a new key. Alternatively a hashing function such as SHA-256 can be used, but a key derivation function which takes a salt parameter is preferred, so multiple keys can be derived from the same secret.

When a symmetric key is revocated, due to reaching its expiration date, or from being compromised, entities can generate a new one, using the aforementioned procedure, with an additional parameter for the key derivation function, to generate a completely new and unique key.


Salt
KDF
New Key
$\{K_B, K_A^{-1}\}$
ECDH
Secret
KDF - Key Derivation Function
K_B - Bob's Private Key
K_A^{-1} - Alice's Public key

(a) Alice generates key


Salt
KDF
New Key
$\{K_A, K_B^{-1}\}$
ECDH
Secret
KDF - Key Derivation Function
K_A - Alice's Private key
K_B^{-1} - Bob's Public Key

(b) Bob generates the same key

Figure 4.4: Alice and Bob generate new key from both asymmetric key pairs

5 Implementation

Contents

This section presents the implementation details of the HSM's services and implemented solution. It will start by describing the communication protocols for each operation between the client interface and the device. The next section describes the implementation details such as the standards and libraries, the board's services and implemented operations.

5.1 Protocol

All data and operations will flow through a serial connection between the client software in the individual's computer and the physical box. A communication protocol which sets the communication rules was defined. This section will describe this protocol between the client application and the hardware device. Each operation's goal, stage, what data is traded in each transmission and why is detailed. The serial port connection transmits data one block at a time, each block contains a maximum of 16 bytes. Data is organized in fields, each field has a fixed number of bytes to hold a certain piece of data. The protocols are organized in transmissions. One transmission is two messages, one message from the client interface to the device, and another in the reverse direction. In the protocol diagrams which will be presented next, data flowing in the same direction consecutively may be shown separately in separate messages. This is only for visibility purposes, in practice, the data is sent together, each in its own field with a defined number of bytes.

5.1.1 Authentication Protocol

Before executing any operation the user must authenticate himself to the device. Figure 5.1 depicts the login protocol.

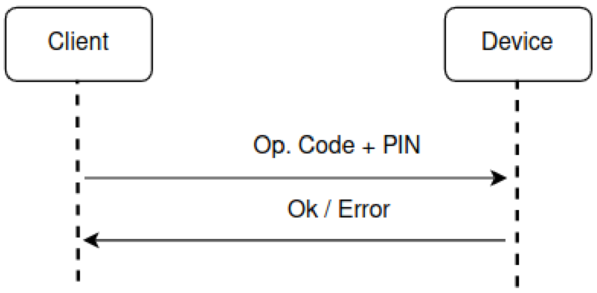


Figure 5.1: Communications protocol to login and change login PIN

If the user is not authenticated, only the authentication operation is allowed. The communication is initiated by the client interface. It sends the operation code and the login PIN. The device responds with an appropriate failure or success status. If successful, the session will be unlocked, allowing the user access to all other operations.

The user is allowed to change the login PIN. The protocol is exactly the same as the login, with the exception that the supplied PIN is the new PIN which will overwrite the current. The user will receive an error status if not logged in.

5.1.2 Secure Communications Protocol

The protocol for operations which secure communications are defined here. It covers the operation which enables secure data exchange by providing the confidentiality and authentication services. It also covers the qualified digital signatures which provide non-repudiation. The protocol which enables secure data exchange, illustrated in figure 5.2, consists of two transmissions.

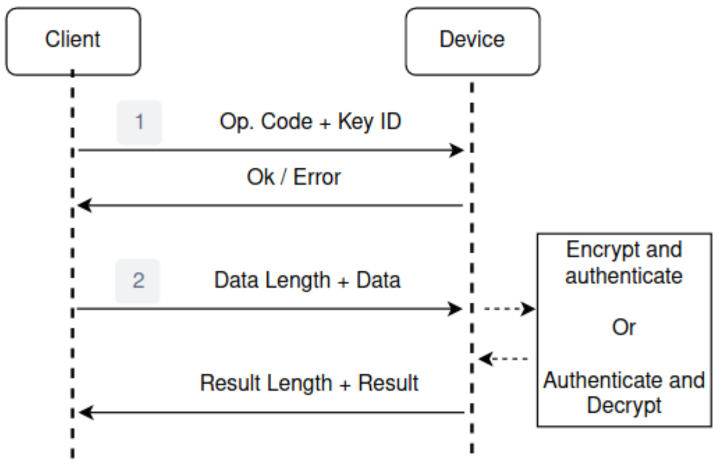


Figure 5.2: Communication protocol for the encryption and authentication service using internal HSM keys

The first transmission is started by the client sending the operation code and key ID, which identifies the internal HSM key used to encrypt and authenticate. The device returns a status message. If the key ID was invalid or the user is not authenticated, the status is an error. In the second transmission the client application sends the data length, followed by the data to be encrypted and authenticated internally. Afterwards, the device start the encryption process and MAC generation. When finished, the devices outputs the result length and result data. If there is a processing error, the result length is 0.

The protocol for decryption and encryption is identical. The only difference is the transmitted data. For the encryption service, the device receives the plaintext data and returns a message with the generated MAC, IV and encrypted data. For decryption, the device receives the MAC, IV, ciphertext, and returns the original plaintext.

The encryption service is pictured detailed in the formula:

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

First, the plaintext data is encrypted with a symmetric key and a randomly generated IV. Next a MAC is generated from the concatenated IV and ciphertext.

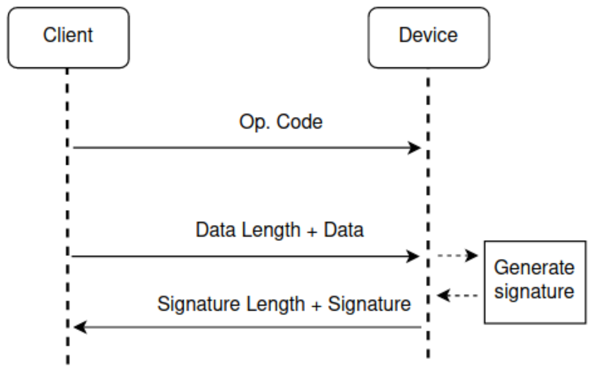
The decryption service is pictured in the following formula:

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

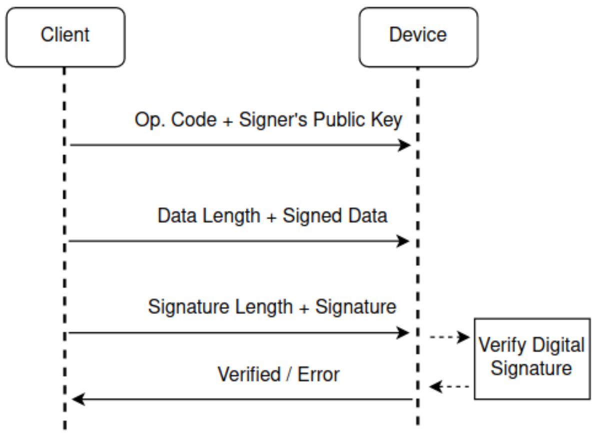
A new MAC is generated from the concatenated IV and ciphertext received. Then the received and computed MACs are compared. If identical, the data is authenticated, and the ciphertext is decrypted with the same symmetric key used for encryption and the IV, to obtain the plaintext.

In the final stage, after the box has finished the cryptographic operations, the device sends the result size to the client, waits for an OK message and returns the result.

The next protocols are relating to the generation and verification of digital signatures. The communication protocol for generation is represented in figure 5.3(a).



(a) Generation


Client
Device
Op. Code + Signer's Public Key
Data Length + Signed Data
Signature Length + Signature
Verify Digital Signature
Verified / Error

(b) Verification

Figure 5.3: Communication protocol for generating digital signatures and verifying the signatures using the internal HSM asymmetric key pair

The protocols only have one transmission. They are separated only for readability. The client application sends the operation code, data length and data to be signed. The qualified digital signature will be generated using the internal private key and the data. Afterwards the device responds with the signature length and generated signature.

The device will generate the digital signature using the device's private key and a hash function according to the formula:

$$Sign_K\{Hash\{Data\}\}$$

The protocol for verifying digital signatures is pictured in figure 5.3(b). The client sends the operation code, signature length and generated signature, as well as the data length and data from which the signature was generated. The last piece the client sends is the public key of the device where the signature was generated.

After verifying the signature, the device responds with the operation status. The data is verified using a hash function and the board's ECC services.

$$\{Hash\{Data\}\} == Decrypt_{K_{-1}}\{Signature\}$$

The hash is generated from the data using the same hash function as the signer. The result is compared with the value obtained from the decryption of the signature using the signer's public key. The signature is verified if the values are identical.

5.1.3 Key Management Protocols

The protocols for the communication management operations are detailed next, namely for symmetric key generation. The protocol to generate a new symmetric key with another entity using asymmetric cryptography is detailed in figure 5.4.

The user forwards the operation code, public key and salt value to the device. The device will then generate a symmetric key with:

$$KDF\{ECDH_K\{K^{-1}\}, Salt\}$$

A secret is generated from the ECDH algorithm, with the device's private key, and another entity's public key. The same secret can be generated with the device's public key and the other entity's private key. This secret is run through a key derivation function in conjunction with a salt which can be public, to generate a symmetric key. If both entities generate the same secret and agree on the same salt value, the generated key is identical.

The new key is then stored in non-volatile memory encrypted with the dedicated symmetric encryption key for storage, saved in its PUF slot. A new unique ID is generated for the key, which is stored alongside the key in memory. This ID is returned to the user along with the operation success status.

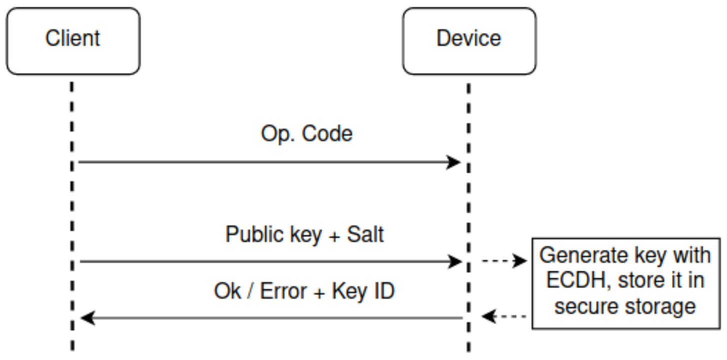


Figure 5.4: Communication protocol to generate symmetric keys, with the HSM internal private key, and stored them internally

5.1.4 Import Keys into HSM Protocol

This section details the protocol to import a list of encrypted symmetric keys, into the device. The imported keys must be encrypted with a predefined symmetric key, stored in a dedicated PUF slot inside the device. This service is useful so a central management section with a similar device can distribute keys to the enrolled entities, on a regular schedule, such as monthly. The protocol for this service is pictured in figure 5.5.

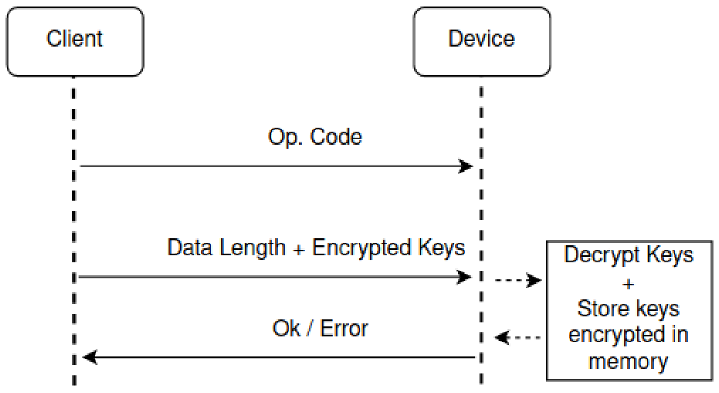

Client
Device
Op. Code
Data Length + Encrypted Keys
Ok / Error
Decrypt Keys + Store keys encrypted in memory

Figure 5.5: Communication protocol to import encrypted symmetric keys, and store them encrypted in the HSM's memory

The only transmission consists in the client sending the operation code, keys length and key data.

Then, the device decrypts and authenticates the keys using a predefined symmetric key stored in a secure PUF slot. The protocol used is the same for secure data exchange described in **??**.

Each key consists of 1 byte for its ID, 1 byte for its size and remaining bytes for the key data. The decrypted list of keys is encrypted again, using AES CTR mode with another predefined symmetric key for storage of data in non-volatile memory. The ciphertext, randomly generated IV and ciphertext length is stored in memory. To provide authentication and thus protection against tampering, a MAC is generated from the three pieces of data and stored in a PUF slot.

In order to fetch a stored symmetric key to use in one of the implemented services, the MAC is generated and compared to the PUF slot. If identical, the key can be decrypted from memory using the same key used for its encryption and identified with its ID.

5.2 Implementation

This section contains all details about the developed implementation on the Hardware Security Module device and client interface.

5.2.1 Libraries and Tools

The cryptographic software is running on the SoC of a SmartFusion2 board, version M2S090TS from Microsemi, with the specifications described in section 2.3.2. It is an adequate device due to several components such as the required cryptographic functions in its system controller, a TRNG essential for cryptography, secure storage for keys and anti-tampering protections. The application was implemented using the C programming language, with the SoftConsole v3.4 Integrated Development Environment (IDE) and libraries of the board functionalities, provided by Microsemi. The configuration was generated using Libero v11.7. The device functions as a HSM, connected through a USB connection to a computer. It was programmed using the external FlashPro4 programmer required to develop and debug embedded applications with SoftConsole [15].

The inclusion of a cryptography library, such as Mbedtls, designed for embedded systems was considered. The library provides algorithm implementations: ECDH, ECDSA, AES encryption, HMAC and support for generation of X.509v3 certificates. The library can be integrated with the boards functionalities and have unnecessary features disabled to lower the used memory [14]. The paper compares the performance of operations on the smartfusion2 with the mbedtls library and the board's security cores. It concludes AES encryption has a higher throughput of 1.8 Mbit/s on the mbedtls implementation compared to the 730 Kbit/s on the embedded core. For the elliptic curve scalar multiplication, used for ECDH, the software implementation is significantly slower, from 28.4s per operation compared to 0.57s with the embedded core. Ultimately, the security services offered by the device provide the necessary functionality to fulfill the requirements defined previously, with the exception of generating the device's certificate, but this was done by exporting the public key, generating the certificate and loading the certificate on the device. Beyond increasing the memory footprint, the library does not offer significant performance benefits, or security measures against side-channel attacks similar to the one's offered by the device's services.

The client interface implemented the PKCS#11 standard for a higher device interoperability. The client interface was implemented in C++ for the open source MinGW compiler in Windows 10, and is composed of a simple command line interface which calls the implemented PKCS#11 interface functions based on the user's input.

5.2.2 Communications

The device implementation uses the available Universal asynchronous receiver-transmitter (UART) communication controller and corresponding drivers for communications. It allows the board to send and receive messages of 16 block bytes at a time. An abstraction layer on both ends was implemented to send and receive byte strings of any size with all the logic hidden. When data of variable size is exchanged, such as plaintext and ciphertext, before transmission, the number of bytes is sent, so the receiver can validate and know how many bytes to expect. In the case of fixed size data such as public keys and hashes this is not needed. For the remaining messages, which is usually a status message, all strings are terminated by the carriage return byte, so the receiver knows when the sender is done transmitting the message.

5.2.3 Smartfusion Implementation

The application in the device was implemented using several of the security services of the board: SRAM-PUF, AES, HMAC-SHA-256, ECC scalar point multiplication, KeyTree, NRBG and tamper monitoring.

5.2.4 Initial State

The device will come from fabric configured and prepared with the necessary keys depending on two scenarios. In the simpler scenario, the device comes with the symmetric keys already shared and stored in each entities' device. They can begin to communicate immediately, with no setup necessary. In the other scenario, the entities will receive the device with a pair of asymmetric keys, a private and public, generated inside the device from fabric. Each device will have the user's public keys, whom he wishes to communicate. The entity can request whose public keys he wants, before the device is initialized in fabric. This allows the users to share symmetric keys between them, which they can user to begin trading data securely.

All communications start with the client sending the operation code and receiving an OK message, meaning the user has access to the operation. If the user is not authenticated he receives an appropriate error message. The authentication operation can always be be accessed by the user, all other operations can only be accessed after the user is authenticated. Sending the operation code acts as a acknowledge message to check if the device is connected and powered to the computer.

Authentication

The authentication PIN is saved in the SRAM-PUF service in a predefined and static slot. When authentication is performed, the PIN is fetched, compared with the client's supplied PIN and erased from memory. If the user is authenticated the PIN can be changed, by enrolling and overwriting the value in the PIN assigned SRAM-PUF slot.

Secure Communications

As previously mentioned, the security of AES-GCM in hardware is considered to be unsurpassed by any authenticated-encryption scheme [4]. Unfortunately, the SmartFusion2 board, does not support this encryption algorithm. Therefore the solution implements a combination of an encryption algorithm with an authentication scheme. Among the supported AES modes, as concluded in chapter 2, CTR mode is the most favourable option because of its efficiency and security, assuming the IV is unique for each message.

For authentication, the board supports HMAC with the SHA-256 hash function, which uses a 256 bit symmetric key and generates a 256 bit code. As concluded in section 2.1.3, it is a well-designed construction, even though it is not the most efficient.

For combining these algorithms, studies have shown that a combination of secure encryption and secure MAC must use the encrypt-then-MAC method [24]. Considering these guidelines, the data encryption/decryption and authentication implementation follows the protocol presented in figures 4.2(a) and 4.2(a). The key used in HMAC should be different from the one used in encryption, to ensure the best security practices. So in practice, a key used for securing communications is split into two keys, one for encryption and one for authentication. When choosing key sizes, taking into account the limited storage capacity of the board, a smaller, but still secure, key size is preferred. The AES 128 bit or 256 bit services guarantee both 128 and 256 bit security. According to the NIST recommendations [25], algorithms which guarantee both 128 and 256 bit security, are expected to be secure from 2031 and beyond. Thus AES with 128 bit keys is preferred, since it guarantees adequate security for the foreseeable future, and is the more conservative option. Even though the HMAC service needs a 256 bit key, a 128 bit key can be used with the upper 128 bits padded with zeros [15]. Therefore for every communication a 256 bit key is used, the lower 128 bits for encryption and upper 128 bits for authentication.

On encryption, a unique 16 byte IV is generated using the NRBG service. The encryption and authentication keys, both 128 bit, are fetched from the SRAM-PUF service, using the slot (key ID) requested by the client application. The returned string is either the original plaintext, or the MAC, IV and ciphertext, concatenated in that order.

Key Generation

The supported ECC P-384 curve guarantees 192 bit security. The implemented new key generation operation protocol is defined in figures 4.4(a) and 4.4(b). The peer's public key is supplied by the client application, and the device's private key is fetched from a predetermined PUF slot where it is saved.

For the ECDH algorithm the ECC scalar multiplication board accelerator is used. It multiplies a scalar value of 48 bytes, the private key, with a point with a x and y coordinate of both 48 bytes (96 bytes total), the public key. The result is a 96 byte point, which is effectively the shared secret. Then to generate the 256 bit key, the KeyTree service is applied as a key derivation function on the shared secret combined with a salt, which can be public. The generated key is enrolled in the first available PUF slot, which is provided by a function in the API, and is returned to the user as the newly generated key ID.

Key Revocation

Key revocation only consists of deleting the key from the SRAM-PUF slot it is saved. This is a simple call to the API function.

Tamper Detection

Tamper and failure events detection is implemented. When certain events occur, flags are set which prevent the user from performing operations, to protect all possible sensitive data. On tamper detection events, zeroization is performed on the device, erasing all keys and data from the device. This process either makes the board unusable or reset to the default fabric state. These measures effectively have a denial-of-service effect but is a trade-off in order to avoid successful attacks and potential leaks of sensitive information, e.g. DPA.

Using the AES and HMAC services, in the secure communications operation, which are not fully DPA-resistant, makes the keys and information vulnerable to these attacks, as discussed in section 2.3.2. Attackers need physical access to the board and voltage measurement tools to perform them. With the developed implementation using tamper detection, which blocks attackers from using the device and zeroization which erases all sensitive data from the device, helps mitigate this type of attacks. All this is invalid, if the attacks somehow gain access to the user's authentication PIN. Which is why regular replacement of symmetric keys used in the vulnerable operations is recommended, and is a missing feature of the implementation. For future work a feature could be developed where a central entity, once every month, sends a encrypted message using pre-distributed keys, with a new key set, to replace the existing set.

5.2.5 PKCS#11 Interface

The application only supports one token and one session for the token. PKCS#11 defines two types of users, the regular user and security office. The implemented application does not make that distinction. Both types of user can login and perform operations. The implemented PKCS#11 API is presented in appendix **??**, table A.1.

The implemented application initializes the cryptoki application and opens the session with the device through the `C_Initialize` and `C_OpenSession` functions. The opposite `C_CloseSession` and `C_Finalize` functions close the session and terminate the cryptoki application. The interface handles the input from the user and uses the data to call the functions in the implemented PKCS#11 API. First the user inputs the operation code he wishes to perform, and the application handles the correct API calls to the PKCS#11 interface. The authentication operation calls the `C_Login` function. The authentication PIN can then be changed with the `C_SetPIN` function. The `C_Logout` function logs out the user. Only the operation code is needed for this operation.

To encrypt and authenticate some data, a local object for the secret key is created with `C_CreateObject`. It does not contain the key data, only the key ID, so it can be passed of to the PKCS#11 API. The encryption is performed through calls to `C_EncryptInit` and then `C_Encrypt`. The same logic applied to decryption with `C_DecryptInit` and then `C_Decrypt`

For deleting a key on the device, first a secret key object is created with its ID, like before, and then a call to `C_DestroyObject` handle the rest. A new key is generated with the ECDH mechanism and SHA-256 key derivation function, with a call to `C_DeriveKey`.

Only one custom which is called directly is implemented. The `HSM_C_GetKeyList` returns the number of enrolled keys in the device.

6 System Evaluation

Contents

This chapter concerns the system and board evaluation. The system and board were evaluated regarding its perfromance and the fullfilled requirements outlined in the previous project report and the problem definition chapter 3.

6.1 Performance Tests

The test objectives, configuration, results and conclusions are detailed for every tested component. The communication channel, smartfusion2 board's security services and implemented services were tested. The measured performance metrics were the processing time for every test, and the tested component's throughput.

6.1.1 Testing configuration

The tests were all performed on a Windows 10 computer, connected to the HSM device through a UART serial port. The implemented PKCS#11 program interface was used to run the tests. Two programs were running on the computer while performing the tests, the client program on powershell and the SoftConsole IDE to run the code on the smartfusion board. For all tests, the serial port UART connection was configured with a 115200 bit/s baud rate, 8 data bits, no parity bits and one stop bit. The elapsed time was measured in the client application using the function `gettimeofday()` 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, only for components where it is logical and can potentially have a performance impact. Since the board does not provide a clock and API to measure elapsed time, the time has to be measured on the client's side. Time measurement starts in the client interface right before sending a message to the device which triggers the operation, and stops when the client receives a message from the device, after the operation is finished. Tests were performed in two different configurations. For the first, the measured operation is performed once in the device each transmission. This transmission is repeated multiple times for every set of values, until an acceptable variance is achieved. For most components the variance is well bellow 1%. The more volatile test results have a variance below 4%. Obvious outlier values were excluded from the experimental calculations. The adopted rule was that values which are significantly above or below the average, and are never repeated were eliminated from the sample set. For the second scenario, tests on components where the time to transmit messages needed to minimized to more accurately assess the corresponding component's performance, for each transmission the operation was performed 50 times. The resulting time was divided by 50 to obtain the processing time per operation.

6.1.2 Communications

In order to assess the communication channel performance, the average time to transmit data was measured. The client sent the data and the device sent the data back immediately without performing another operation. The test, in accordance with the first scenario, was repeated at least 30 times for each data length. The highest variance did not go above 0.2%. The average transmission times for each value are displayed in figure 6.1.

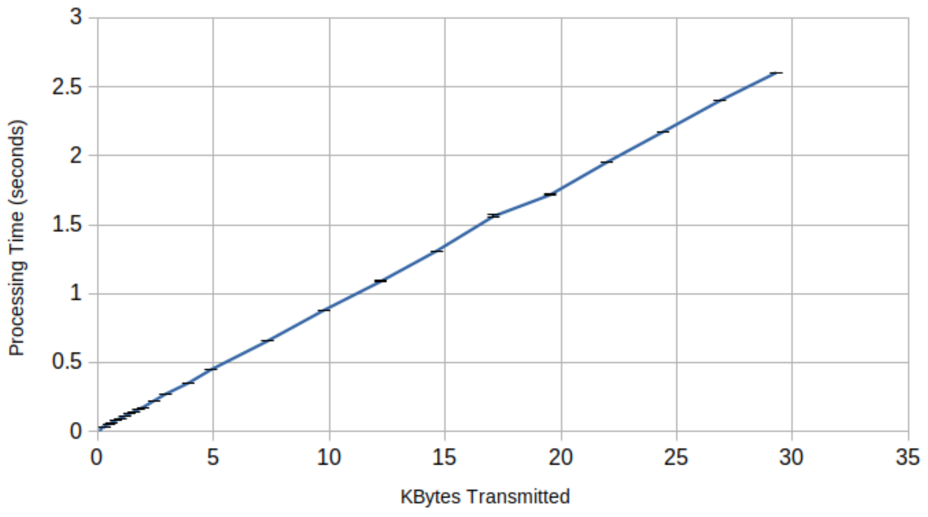


Figure 6.1: Average Data Transmission Times

The values range from 0.016 seconds for 50 bytes, to 2.6 seconds for 29 KBytes. From the graph we can conclude the performance has linear scalability which is ideal for a system.

For the subsequent graphic, the throughput was calculated from the transmission tests, for every repetition. Figure 6.2 plots the experimental throughput and theoretical throughput. The theoretical throughput was calculated from the baud rate $115200/8 = 14.06KBytes/s$.

We observe the experimental throughput is 3.05 KB/s for data below 100 bytes and starts stabilizing around 12 KB/s to 11 KB/s as data size increases. Thus we can conclude the practical throughput is close to the theoretical, and as expected stabilizes as the sample size increases, meaning, more bytes are transmitted.

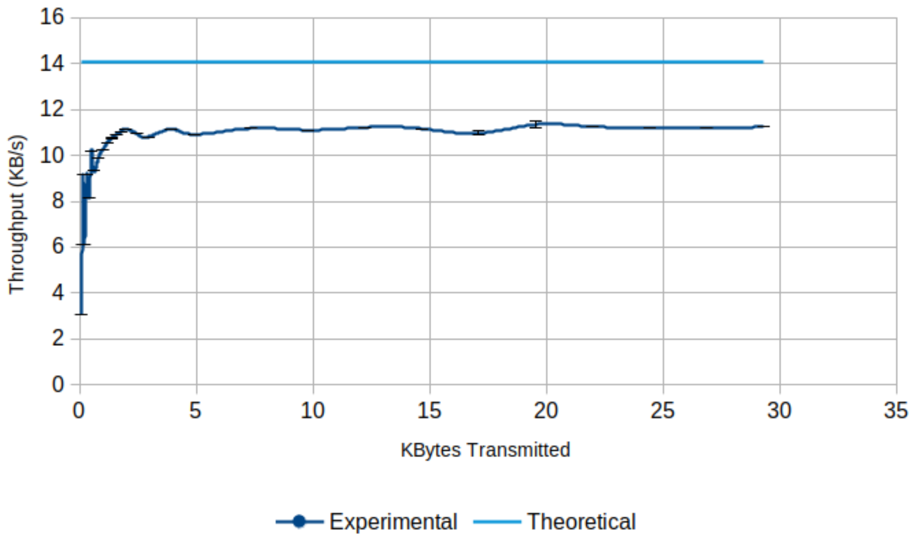


Figure 6.2: Serial Port Communications Throughput

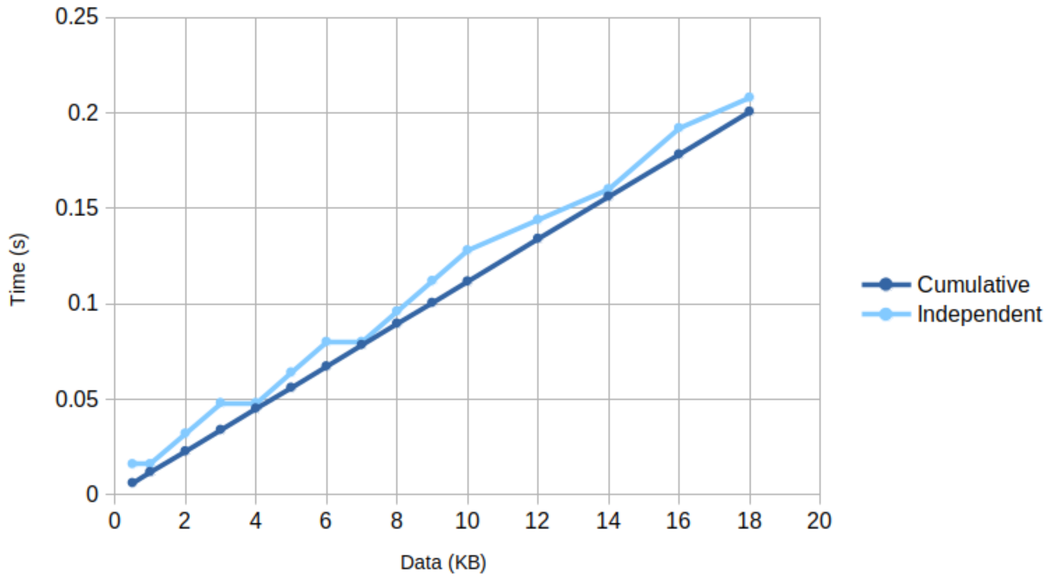


Figure 6.3: AES 128 processing time

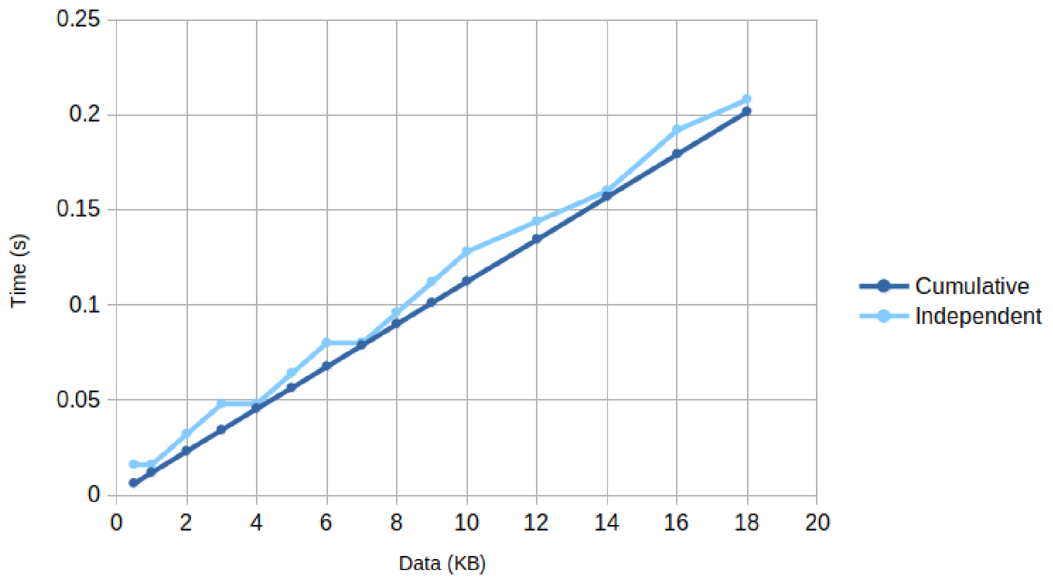


Figure 6.4: AES 256 processing time

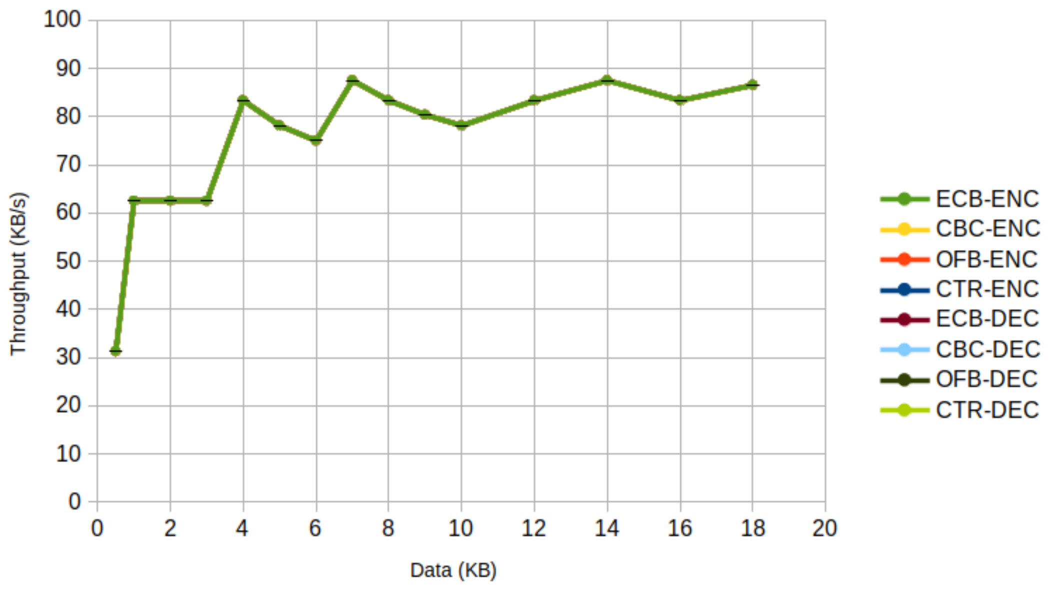


Figure 6.5: AES Throughput

6.2 Smartfusion2 Security Services

6.2.1 AES-128 and AES-256

6.2.2 SHA-256 Based Services

6.3 Implemented Services

The same library used to measure the time in the communication tests was used. The time was measured at the client application before sending the message which will trigger the service at the HSM, and

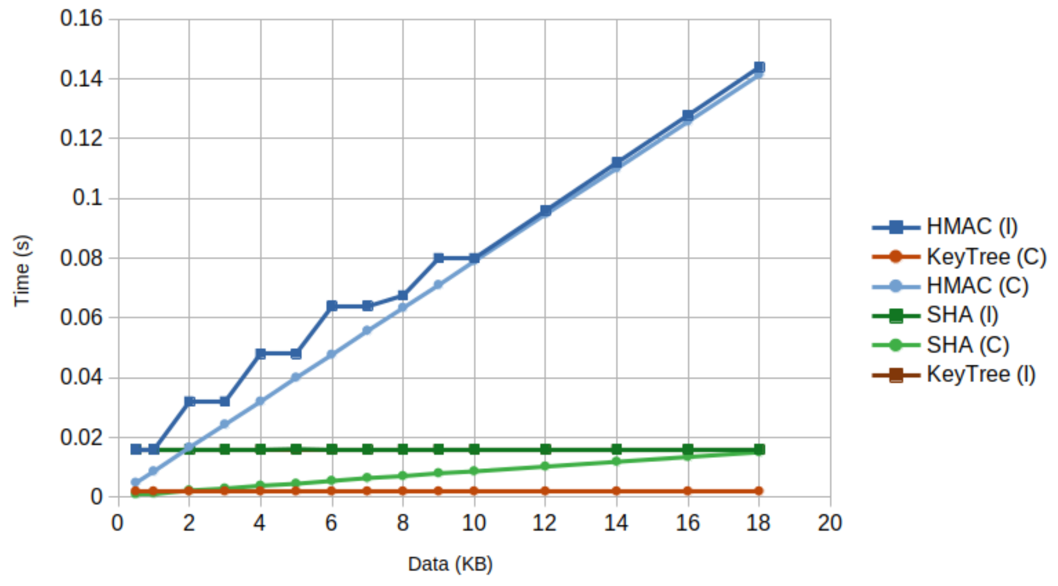


Figure 6.6: SHA based services processing time

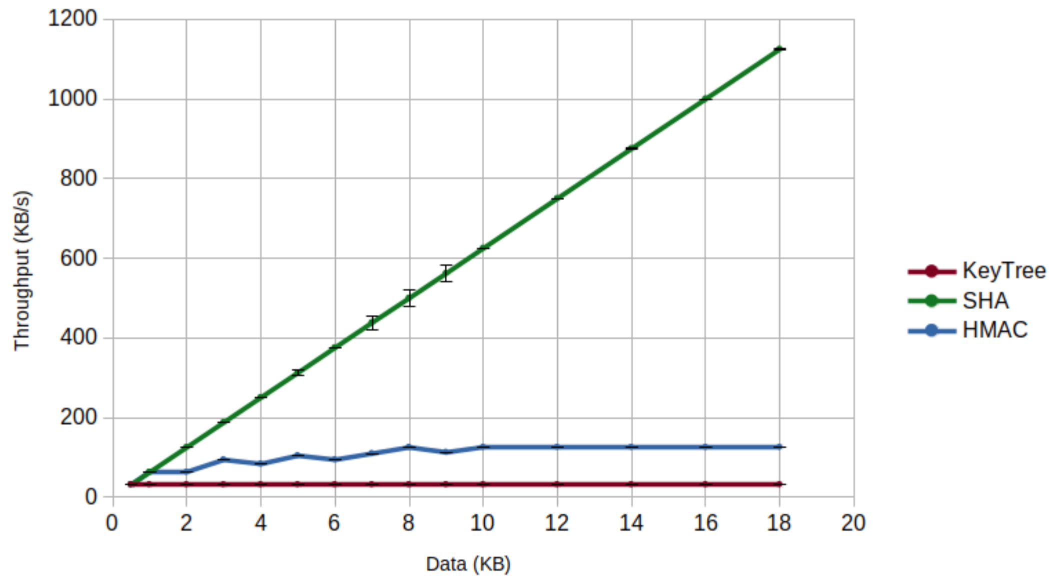


Figure 6.7: SHA based services throughput

after it has received the result.

In order to get a real time measurement, each operation was performed 1000 times on the HSM, with the communications only done once. From the resulting time, the predicted time spent on communications was subtracted using the data from the previous test. The result was then divided by 1000.

The data encryption and decryption operations were ran with different data sizes, in order to asses the data size impact on performance. The ECDH key generation always uses a private and public key of the same size, so no message size variation is possible.

6.3.1 Secure Communications

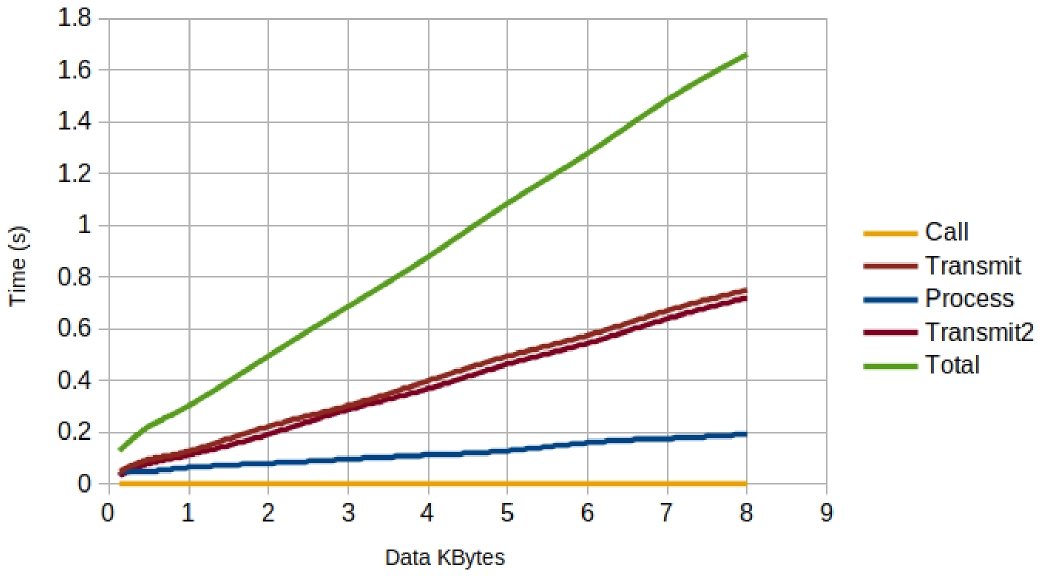


Figure 6.8: Secure Communications Average Time

The results in table 6.8 for the encryption and decryption operations are very similar due to both using the same board services, AES encryption and HMAC, but in a different order. It is also important to note AES encryption and decryption in CTR mode is essentially the same operation due the mode's characteristics. Relating to the variation in data size, the values vary between approximately 0.1284 and 0.1825 seconds, which is a very insignificant difference. Thus we can conclude, the data size has a negligible impact on the operations performance.

6.3.2 Import Keys

6.3.3 Key Generation

Regarding the key generation operation results in table 6.11, two values were obtained through different methods. Due to the operation using SRAM-PUF services to enroll new keys in the eNVM memory, with limited write cycles and key slots, this operation cannot be repeated enough times to get a relevant enough sample size. So a trade off was achieved. The operation was performed 1000 times without the key enrollment operation, meaning only the ecdh key generation algorithm and key derivation function (SHA-256). Since the enrollment phase is presumed to be expensive, due to writing in eNVM memory,

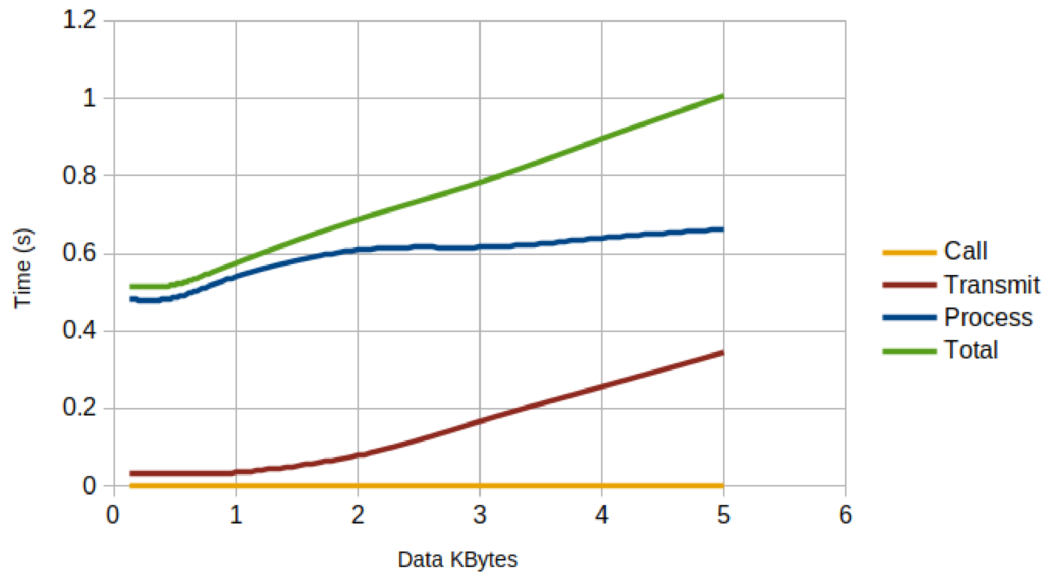


Figure 6.9: Import Keys Average Time

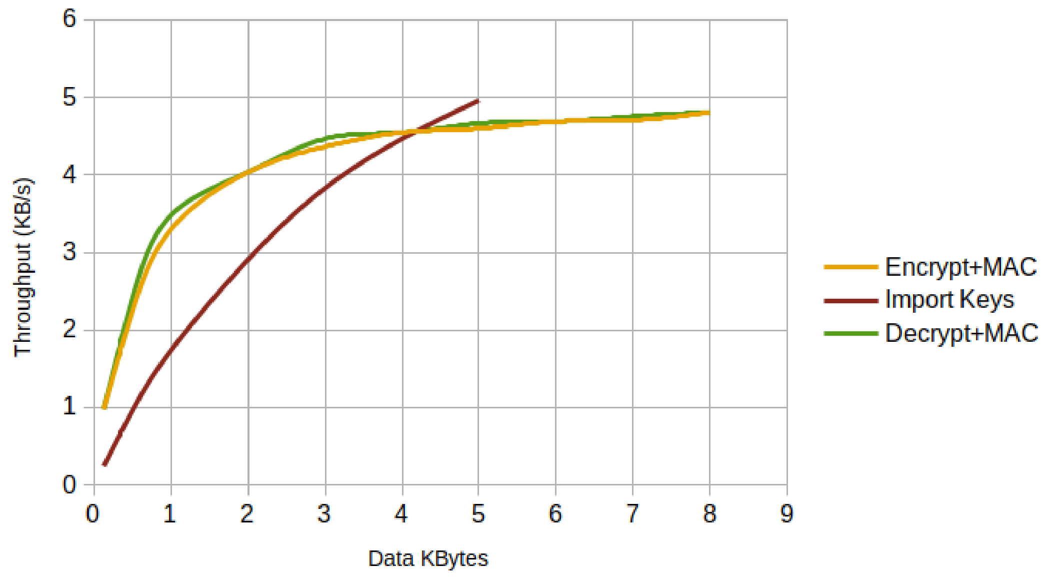


Figure 6.10: Implemented Services Throughput Comparison

the test was also performed 10 times with key enrollment, to get an idea of its potential performance cost.

A higher time of 0.577 seconds without enrollment and 1.764s with, compared to the previous operations is expected due to the higher cost of operations with asymmetric keys. However, comparing both values we can conclude saving the key in memory, has most likely the higher performance impact on the operation. This result is congruent with the one obtained by [14] of 0.57s per ECC scalar multiplication.

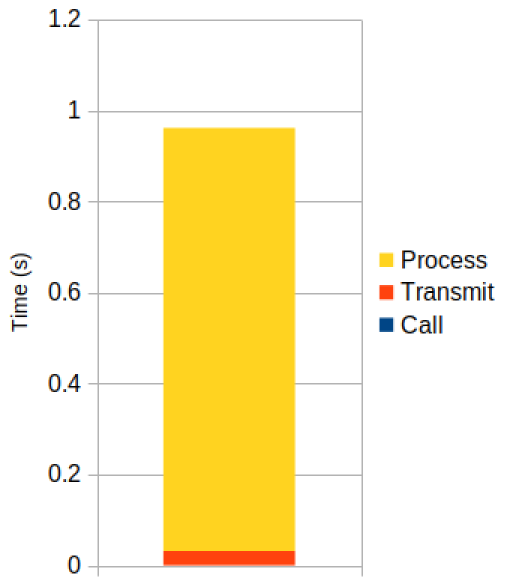


Figure 6.11: Key Generation Average Time

Thus we can conclude, the scalar multiplication is the second most expensive operation, after the key enrollment, since the key derivation function has a negligible impact on the processing time (0.577s with the function vs 0.57s without).

6.4 Requirements

A M2S090TS smartfusion2 evaluation kit is priced at 384€ [26].

7 Conclusion

Contents

7.1 Summary

7.2 Future Work

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] E. Rescorla and T. Dierks, “The transport layer security (tls) protocol version 1.3,” 2018.

[12] 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.

[13] Microsemi, "Specify and program security settings and keys with smartfusion2 and igloo2 fpgas," 2013.

[14] D. Parrinha and R. Chaves, "Flexible and low-cost hsm based on non-volatile fpgas," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2017, pp. 1–8.

[15] Microsemi, "User guide smartfusion2 and igloo2 fpga security and best practices," 2019.

[16] ——, "Ds0128 datasheet igloo2 fpga and smartfusion2 soc fpga," 2018.

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

[18] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, 2011.

[19] 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.

[20] 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.

[21] 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.

[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] 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.

[24] 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.

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

[26] "M2s090ts-eval-kit pricing," *Mouser Electronics*. [Online]. Available: https://eu.mouser.com/ProductDetail/Microchip-Microsemi/M2S090TS-EVAL-KIT/?qs=HNBw3F7vE2zzRkt03XBdWg%3D%3D

A

Implementation Details

PKCS#11 Function	**Description**
`C_Initialize`	Initializes the cryptoki application
`C_Finalize`	Closes the cryptoki application
`C_OpenSession`	Opens the session and communication port
`C_CloseSession`	Close session and communication port
`C_Login`	Authenticates users with a PIN
`C_Logout`	Logout the user
`C_SetPIN`	Change authentication PIN
`C_CreateObject`	Creates local objects like symmetric keys and public keys
`C_DestroyObject`	Deletes remote object like a secret key
`C_EncryptInit`	Initialize AES encryption operation
`C_Encrypt`	Encrypt data with AES CTR mode
`C_DecryptInit`	Initializes decryption operation
`C_Decrypt`	Decrypts data with AES CTR mode
`C_DeriveKey`	Generate and derive new symmetric key, store it in device
`C_UnwrapKey`	Import symmetric keys into device
`HSM_C_GetKeyListize`	Custom function to get list of avaible remote symmetric keys

Table A.1: PKCS#11 Implemented Interface API