TÉCNICO LISBOA

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

List of Algorithms

Listings

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
CFB	Cipher Feedback
CTR	Counter
CCM	Counter with CBC-MAC Mode
DH	Diffie-Hellman
EAX	Encrypt-then-Authenticate-then-Translate
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
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
SHA	Secure Hash Algorithms
SoC	System-on-Chip
TLS	Transport Layer Security
TRNG	True Random Number Generator
TPM	Trusted Platform Module
USB	Universal Serial Bus
XTS	XEX-based Tweaked-Codebook mode with ciphertext stealing
CPU	Central Processing Unit

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 goes into detail about the problem, its entities, devices, full requirements, goals of the system and the services it must provide. The third chapter will cover the technical background needed to comprehend the solution and state of the art. The fourth chapter will give context on the related work and existing solutions to the presented problem. The fifth chapter introduces the solution and its architecture. The sixth chapter will describe the system protocol and implementation.

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:

1. They must be deterministic, meaning the same input value must always result in the same hash value;
2. They must generate very different output values for similar inputs;
3. They must be collision resistant, meaning it should be hard to find two input messages that generate the same hash value;
4. The hash value should be computed relatively quickly;
5. Given a hash value, it should be hard to find an input text that produces that hash value.

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.

Cipher Feedback (CFB) is a combination of OFB and CBC. The xor operation is applied on the encrypted IV with the plaintext block. The result is then used for the next block. Similarly to OFB, CFB is effectively a stream cipher. Yet, akin to CBC, and for the same reasons, encryption is not parallelizable, opposed to decryption which is. Random read access is possible, contrary to preprocessing which is not. A specific condition of CFB is it can become self-synchronous, meaning it will recover if s bits are lost. If s=1, it can recover from slips of any number of bits, if s=8 it can recover from slips of any number of bytes. However for every blockcipher call, CFB only processes s bits, compared to 128 bits (block size), which is a major performance cost.

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, OFB and CFB 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. CFB (even with s =

128) and OFB are slower still. When efficiency characteristics matter, nothing comes close to CTR: it has better performance characteristics, in multiple dimensions, than any of CBC, CFB, 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.

XEX-based Tweaked-Codebook mode with ciphertext stealing (XTS) is intended for encrypting data on a storage device. The mode is not approved for any other use. Instead of an IV, it uses the sector number and block index in the sector. It uses a ciphertext stealing to avoid padding in the last block. As expected, it provides no integrity. Therefore is susceptible to data tampering, any ciphertext will be decrypted, regardless of being modified or not.

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.

Encrypt-then-Authenticate-then-Translate (EAX) mode aims to improve on CCM by replacing CBC-MAC with OMAC. Similarly to CCM, it first generates the authentication tag with OMAC, then encrypts with CTR. By using OMAC instead of CBC-MAC, it supports variable-length messages and is online.

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 has the advantages EAX has over CCM. It is online and the AD can be preproccesed. 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. For the same level of security, ECC keys are smaller. Gupta & Silakari, 2011 [6], give as an example a 160-bit ECC key has similar security to a 1024-bit RSA key. They also state that smaller key sizes may result in faster execution timings for the schemes, which is beneficial to systems where real time performance is a critical factor. 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 [7].

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 [8], 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) [9].

2.1.9 Transport Layer Security

TLS is a cryptographic protocol that aims to provide confidentiality and data integrity, during transmission, over TCP/IP [10]. 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 [11]. Applications can use, create and modify objects, without exposing them to the application's memory. The standard has been widely used, promoting interoperability between devices. By using the same standard, 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. Only public objects are available to an unauthenticated session. The normal user has access to both private and public objects but cannot change the authentication PINs. The privileged user can change both his and the normal user's PIN.

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 Field-Programmable Gate Array System-on-Chip

A Field-Programmable Gate Array (FPGA) is an integrated circuit designed to be programmed and configured after manufacturing. FPGAs are often used to prototype and for high specialized systems produced at low scale. One of the major advantages is their agility and flexibility to be customized for special use cases [12]. However, the reconfigurability may introduce certain weaknesses to the system. FPGAs are generic and its bitstream is vulnerable to cloning if no additional protection is applied. Cloning is simple and is considered the worst security vulnerability of volatile FPGAs [13]. The configuration data of these devices is stored in non-volatile memory and may be directly copied if no authentication mechanism is implemented [14].

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 [15]. Smartfusion2 integrates a non-volatile FPGA with a SoC and an internal secure eNVM for storing Phase 0 boot code. Since they are non-volatile, the bitstream is at a lower risk of being probed during boot [16]. The eSRAM flash memory is protected against single event upsets. 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.

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.

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

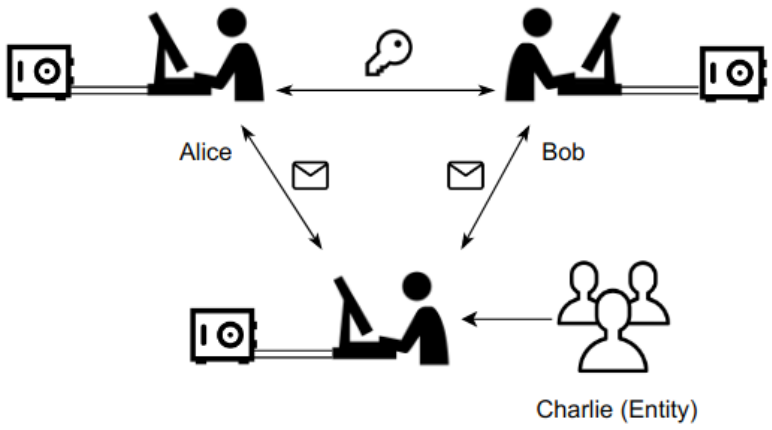


Figure 3.1: System Usage Example

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.

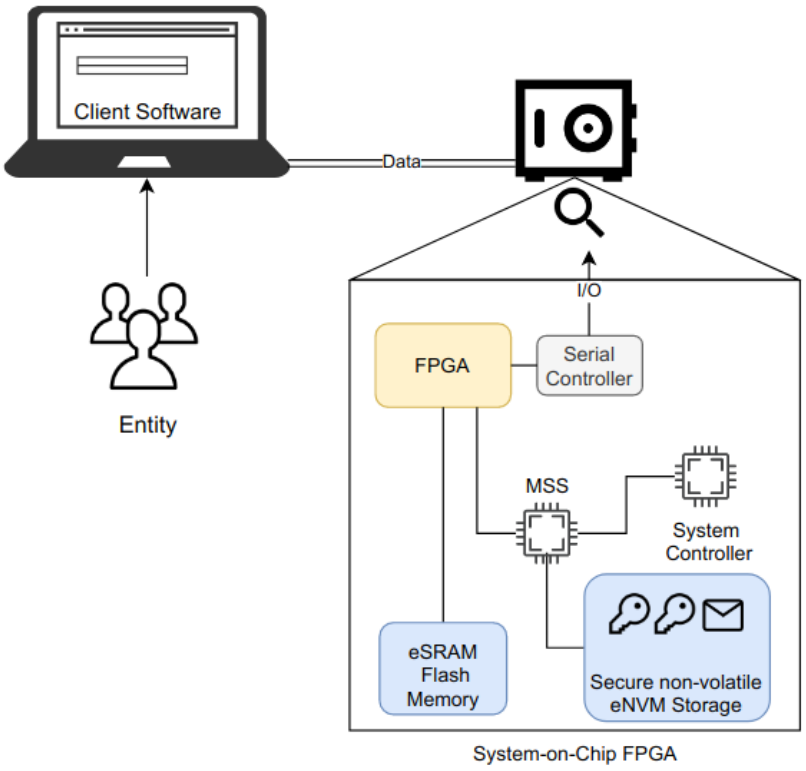


Figure 4.1: System Components

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, signed 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.

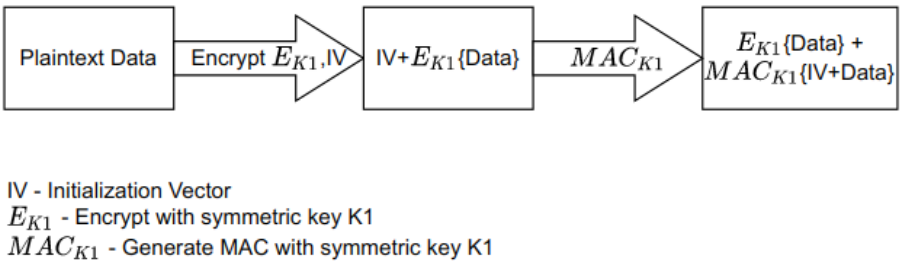
The second scenario has an administrator role and a user role. Each user has its own PIN. The administrator, beyond changing the authentication number, can register new users. To register a new user, both administrator and new user need to be physically together with the device. The administrator authenticates 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.

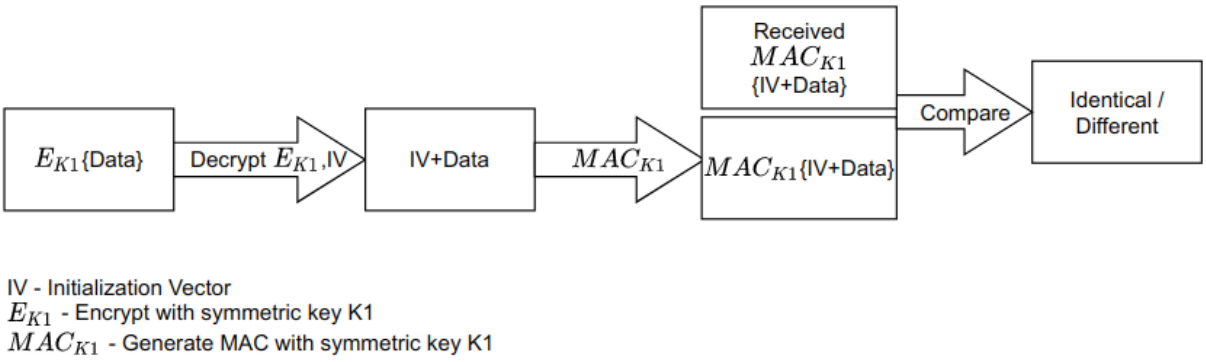
4.2.2 Secure Communication

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).



(a) Encrypt and authenticate data with K1 key



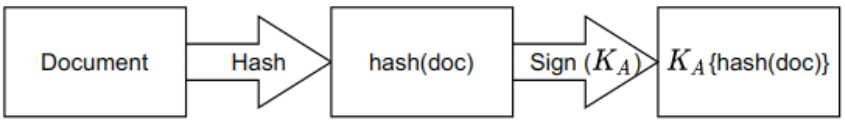
(b) Decrypt data and verify authentication with K1 key

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 (IV, ciphertext, MAC) 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 the ciphertext, it decrypts the data using the same *K1* key and received IV. Then

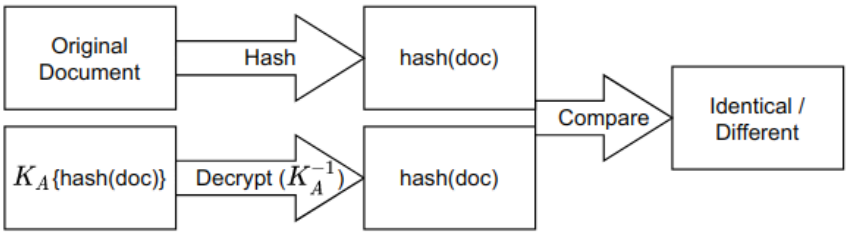
it computes the MAC of the received IV appended with the decrypted data. If the computed and received MAC are identical, the data is authenticated.

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).



K_A - Alice's Private key

(a) Alice generates signature of a document



K_A^{-1} - Alice's Public key

(b) Bob verifies the signature of the document

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

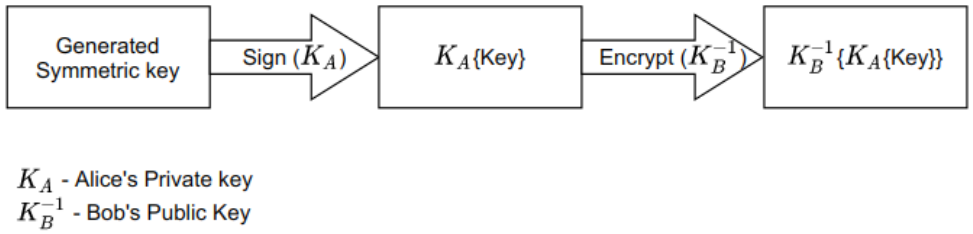
The signature is generated by calculating the hash of the data, and signing the digest with the device's private key. To verify a signature (figure 4.3(b)), the device decrypts the hash with the signer's public key. Next, it computes the hash of the original document, and compares both hashes. If they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

4.2.3 Communication Management

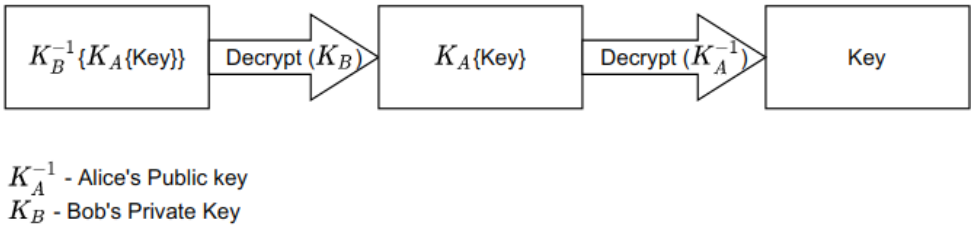
These operations will manage the needed keys to support secure communications, digital signatures and create new secure connections. Supported key management operations are: symmetric key generation, symmetric key revocation, if communications are suspected to be compromised, and importation of other entities' keys. These operations are only available in the scenario where each device has a pair of asymmetric 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, needs to be imported into the device's non-volatile memory. It does not need to be secure storage. After the key is imported, a secure connection with a new entity can be established, by sharing symmetric keys. For an entity to communicate with a new entity, their device will generate a new symmetric key and store it either in secure storage, or in non-volatile memory, encrypted with the device's public key. Figure 4.4(a) illustrates the procedure Alice's device goes through to share a new key with Bob.



(a) Alice generates key



(b) Bob saves key

Figure 4.4: Alice generates a new key to share with Bob

The key is first signed with the sender's private key, which is stored in the device's secure storage, and then ciphered with the receiver's public key, which was imported before. Portrayed in figure 4.4(b), Bob's device decrypts the message using its private key, and decrypts it again with Alice's public key. Finally the key is securely stored in the device, and both entities can subsequently establish secure communications.

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.

5

Implementation

Contents

This section will build on the system architecture. It describes the communication protocols, standard and libraries chosen to implement the operations. It will start by clarifying important assumptions. Then it will start with the initial state specifications, and finish by defining the protocol for each operation, introduced in chapter 4.

5.1 Libraries and Tools

The cryptographic software is running on the SoC of a SmartFusion2 board from Microsemi, discussed 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 Integrated Development Environment (IDE) and libraries of the board functionalities, provided by Microsemi. The device functions as a HSM, connected through a USB connection to a computer. The client software was also implemented in C, and is composed of a simple interface which allows communication through the cable, to the device.

5.1.1 Cryptographic Algorithms

As previously mentioned, AES is a popular symmetric-key standard. 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. The board supports the AES encryption modes for confidentiality, either 128 or 256 bits: ECB, CBC, OFB and CTR. 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. For combining the CTR and HMAC algorithms, studies have shown that a combination of secure encryption and secure MAC must use the encrypt-then-MAC method [17]. When choosing key sizes, taking into account the limited storage capacity of the board, a smaller, but still secure, key size is preferred. AES guarantee both 128 and 256 bit security, with 128 and 256 bit keys. According to the National Institute of Standards and Technology (NIST) recommendations [18], algorithms which guarantee both 128 and 256 bit security, are expected to be secure from 2031 and beyond. AES-128 is chosen, since it guarantees adequate security for the foreseeable future, and is the smaller option.

The system provides a single hash algorithm, SHA-256, which has a security strength of 128 bits. The HMAC-SHA-256 algorithm needs a 256 bit key, different from the one used with encryption, to ensure the best security practices. A 128 bit key can be used, if padded with zeros. Thus, for every

communication, a 128 bit key for encryption and a 128 bit key for authentication is used. In total, a 256 bit shared secret. The used algorithm for public-key cryptography is ECC with 384 bit private and public keys, the only one supported by the board, which guarantees 192 bit security.

The qualified digital signatures operation combines ECC and the SHA-256 hash algorithm, and provides 128 bit security, calculated by taking the smallest security strength value of both algorithms. For the encryption and authentication operation combining AES-128 and HMAC-SHA-256, 128 bit security is provided.

5.1.2 Device Standardization

The solution should support a plethora of devices, as it will increase the adoptability of the solution among clients. This entails the use of a widely established protocol, which clearly defines a set of functions and standards the system should follow. The PKCS #11 standard will be utilized to fulfill this requirements. It allows operations to be standardized across different devices, increasing the range of supported devices. By implementing the system in accordance with these guidelines, it will have a higher device interoperability. Additionally, it allows the application to use, create and modify objects, without exposing them to its memory.

5.2 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 communications protocol needs to defined, in other for this communication to occur. This section will explain and define the communication protocols between both the client application and the hardware device in detail. For each operation, it will describe its goal, the different phases, what data is traded in each phase and why.

5.2.1 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.

5.2.2 Authentication Protocol

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

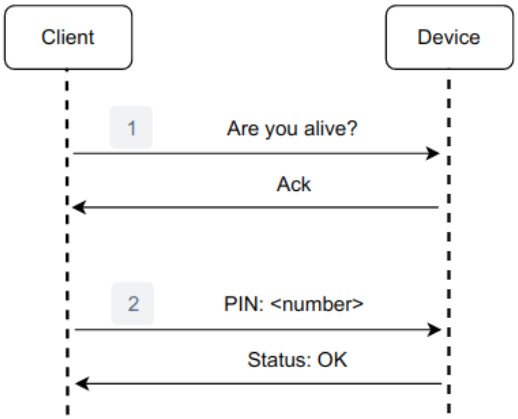


Figure 5.1: Authentication Protocol

The user initiates by sending a message to check if the device is alive and connected to the computer. After the device responds with an acknowledge, the user software sends the authentication PIN. The device will respond with a status parameter indicating failure or success. When it is successful, the session will be unlocked, allowing the user access to the main operations.

5.2.3 Administration Protocol

There is only one administration operation, changing the authentication PIN, pictured in figure 5.2. The user initiates by sending the change PIN operation code, and the new PIN number. The device responds with the return status of the operation.

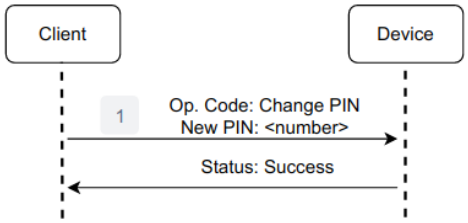


Figure 5.2: Protocol to Change Authentication PIN

5.2.4 Secure Communications Protocol

The protocol for operations which secure communications will be defined here. It covers the encryption and authentication operation, which enables secure communications and non-repudiation through qualified digital signatures. The protocol to encrypt and authenticate data illustrated in figure 5.3 consists of four phases.

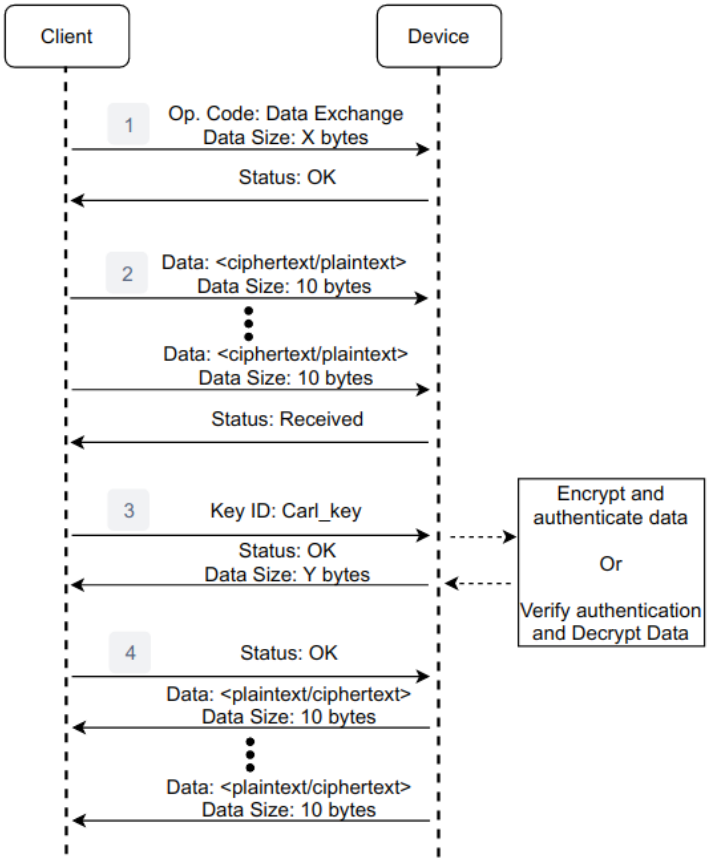


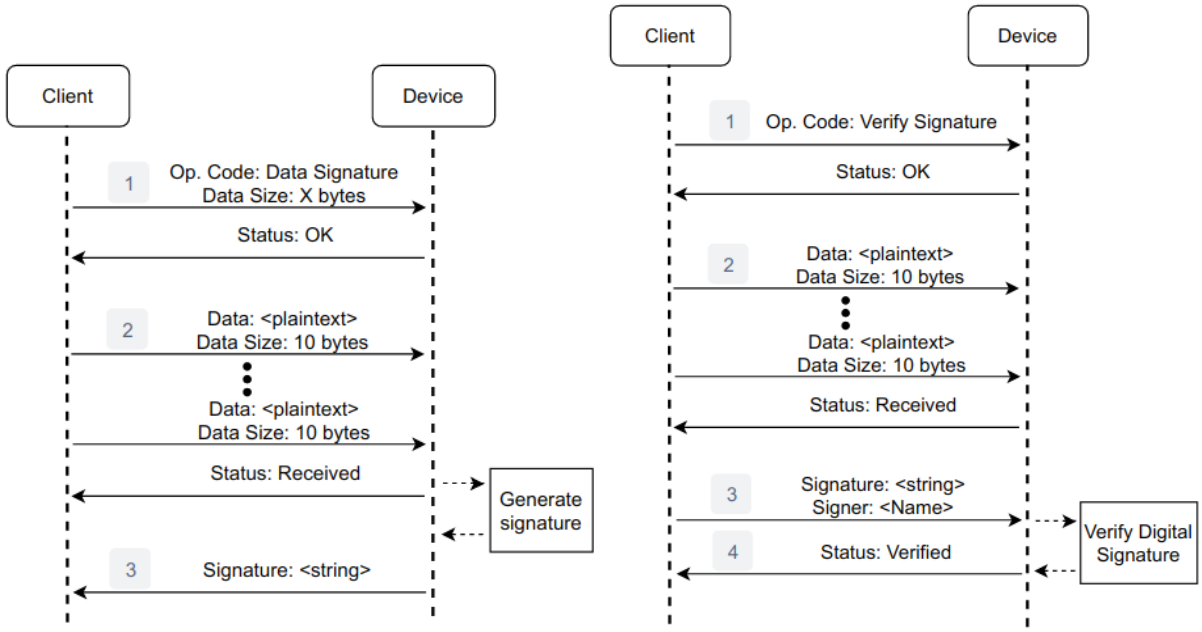
Figure 5.3: Encryption+Authentication Protocol

It starts with the user sending the operation code and the data size. The box will respond with an OK message signaling the client application can begin to transmit the data. A maximum of X bytes per message can be transmitted. Each message contains a part of the data and the size of the data in that packet. When the transmission ends, the device confirms its reception. In the third phase, the client software sends the ID of the symmetric key used to encrypt and authenticate communications with. The box will execute the cryptographic operations and return a status message and the encrypted data size. When the client acknowledges its reception, the encrypted data with the MAC and IV parameters appended, will be sent back, one message at a time.

The protocol to decrypt and verify data authentication is very similar to the previous one, and is also

pictured in figure 5.3. The differences are in phase three, after the ciphertext and symmetric key ID are transmitted, the device will decrypt and authenticate the data. It returns the operation status and the decrypted plaintext size if it was successful. It ends by transmitting the plaintext in the fourth phase to the client software.

The next protocols are relating to the generation and verification of digital signatures. The designed protocol for generation is represented in figure 5.4(a).



(a) Generation Protocol **(b)** Verification Protocol

Figure 5.4: Digital Signature Generation Protocols

The user initiates by sending the operation code and the plaintext data size. When the box responds with an OK message, the user transmits the data to be signed, one message at a time. In possession of the data, the device will generate the digital signature using the device's private key. When finished the signature is sent back to the user.

The protocol for verifying digital signatures is pictured in figure 5.4(b). After the user sends the operation code, and the box responds with an OK message, the user transmits the data, used by the signer to generate the signature, one message at a time. When done, the user also sends the signature and the name of the signer, so the device knows what public key to use to verify the signature. Then, the device will verify the digital signature using the signer's public key, the data and the signature. The result will be sent back to the user.

5.2.5 Communication Management Protocol

The protocols for the communication management operations are detailed next. The operations are importation of public keys from the list provided to entities, generation of new symmetric keys and storage of newly received symmetric keys. The protocol to import public keys is pictured in figure 5.5.

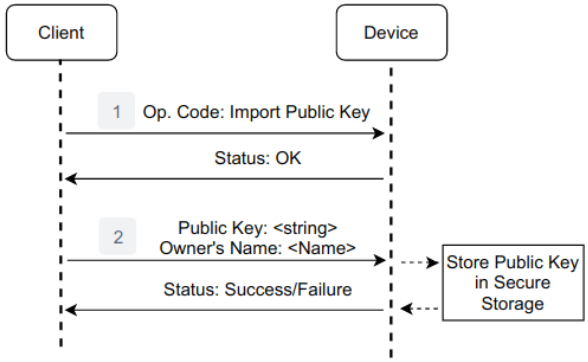
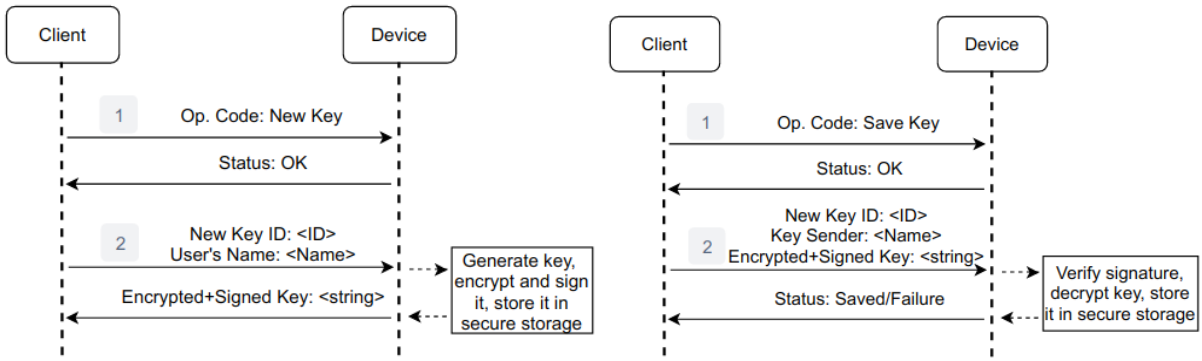


Figure 5.5: Import Public Key

Like previous protocols, it starts with the user sending a message with the operation code, indicating he wants to import someone's public key. After the device responds with an OK signal, the user sends the public key, and the name of the public key's owner. The device stores the key, associated to the owner's name, and informs the client software of the operation's success or failure.

Just like digital signatures, entities can share symmetric keys between each other if they each others public keys are stored in their devices. If not, they must first import them into their respective devices. The protocol to generate a new symmetric key, and secure it to be shared with other entities is detailed in figure 5.6(a).



(a) Generate key

(b) Save key

Figure 5.6: Protocols for sharing keys

After both sides trade the operation code and OK status, the user sends the key ID, which will identify the symmetric key, and the name of the entity it will be shared with, so the device knows which public key to use to secure the key. A new symmetric key will be generated and saved in the device's secure storage, with the key ID sent by the client application. The device encrypts and signs the key with public-key cryptography, and sends it back to the application, one message at a time. The user can then securely share the key with the other entity.

The protocol for the other user to save the newly received symmetric key, and store it inside their device is in figure 5.6(b). After the operations code is sent and the OK signal is returned, the user sends the key ID, the name of the key sender, and the encrypted and signed key. Then, the device verifies the signature and decrypts the key, subsequently saving it in the device's secure storage along with other symmetric keys already present.

6

System Evaluation

Contents

6.1 Performance

6.2 Requirements

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] 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.

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

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

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

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

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

[13] S. Drimer, “Volatile fpga design security–a survey,” *IEEE Computer Society Annual Volume*, pp. 292–297, 2008.

[14] ——, “Authentication of fpga bitstreams: Why and how,” in *International Workshop on Applied Reconfigurable Computing*. Springer, 2007, pp. 73–84.

[15] Microsemi, “Specify and program security settings and keys with smartfusion2 and igloo2 fpgas,” 2013.

[16] 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.

[17] 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.

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

Code of Project

B

A Large Table