Hochschule Offenburg
offenburg.university

ivESK Institut für verlässliche Embedded Systems und Kommunikationselektronik

E+I Elektrotechnik und Informationstechnik

Extension and Integration of an Abstract Interface to Cryptography Providers

Report

Steve Wagner
EI-3nat

February 18, 2016

Prof. Dr. Axel Sikora
Dipl.-Phys. Andreas Walz
Institute for Embedded Systems and Communication Electronics (ivESK)
Hochschule Offenburg

Statutory declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Offenburg, _____ _____

Date Signature

Abstract

English

Abstract in english

German

Abstact in german

Contents

Notes

List of Figures

List of Tables

1 Introduction

1.1 Cryptography

Cryptography uses mathematical techniques for the security of data transmitted over an insecure network.
Cryptanalysis is the complementary of the cryptography with the focus on the defeat of the cryptographic mathematical techniques.
The security of the information is defined into:

- Confidentiality or privacy
  No one, except whom is intended, can understand the transmitted data
- Integrity
  No one can alter the transmitted message without the alteration is being detected
- Authentication
  The sender and the receiver can identify the destination of the data and identify themselves
- Non-repudiation
  The sender cannot deny at a later stage the transmission of the datas

The cryptographic mathematical techniques are grouped into several algorithms:

- Hash algorithm
- Signature algorithm
- Symmetric cipher algorithm
- Asymmetric cipher algorithm

Rewrite introduction cryptography

1.1.1 Hash algorithm

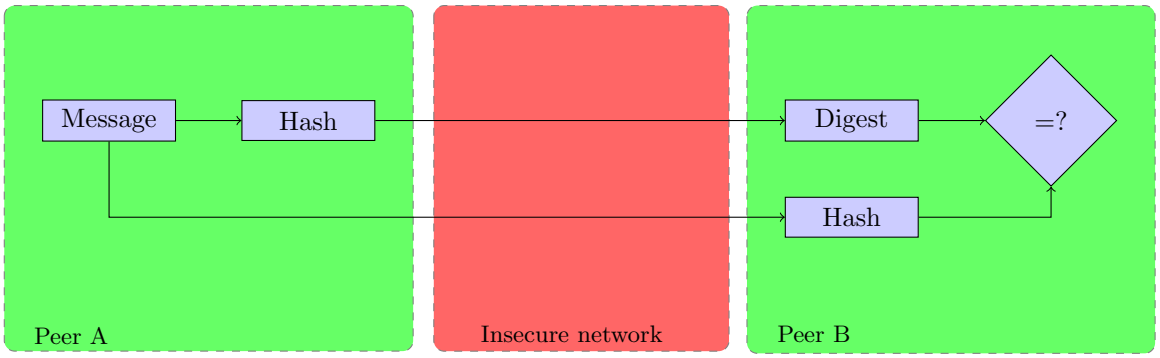

Figure 1.1: Introduction of a hash operation

A cryptographic algorithm is considered practically impossible to invert, meaning that impossible to recreate the input data (message) with the digest (output of the hash). The main properties of a hash function are:

- it's quick to compute the digest for any message
- it's infeasible to generate a message from its digest
- it is infeasible to modify a message without changing the digest
- it is infeasible to find two different messages with the same digest.

On Figure 1.1, the message is hashed and the result (digest) is sent to the other peer with the original message.
Then the other peer hashes the message too (with the same algorithm) and compares the two digests to know if the message has been changed during the transmission.
This is the principle of integrity.

Add table with example of hash with same word but one with capital letter at the beginning (to show the difference)

1.1.2 Signature algorithm

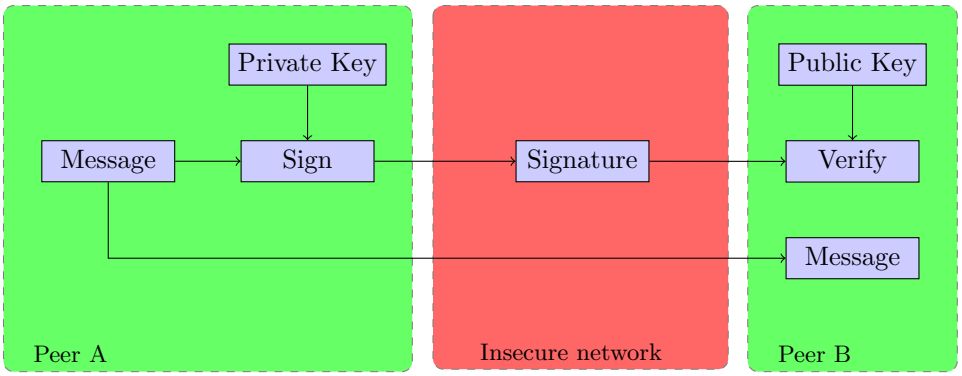

Figure 1.2: Introduction of a signature operation

A digital signature is a mathematical principle to demonstrate the authenticity of a message.

It infeasible to generate the original message with the signature. A valid digital signature allows to be sure that the incoming message comes from the peer we are communicated with and not from someone else (authenticity).
With digital signatures the sender cannot deny having sent the message (non-repudiation).
It allows too to be sure that the message has not been corrupted during the transmission (integrity).

The principle of a digital signature is shown figure 1.2.
The message is signed with the private key (no one has this key too) and the result (signature) is sent to the other peer (peer B).
The public key of the peer A has already been sent previously. With this key, peer B can verify the signature (only with the public key of peer A) and be sure that the message sends with it comes from peer A and not from someone else.

1.1.3 Cipher

The cipher is an algorithm uses for encryption of data (plaintext) and decryption of data (ciphertext). It's uses on internet, on the e-mail, on cellphone when calling, etc.
Two main cipher algorithms exists, which are:

- Symmetric
- Asymmetric

1.1.3.1 Symmetric cipher algorithm

Symmetric-key algorithms are algorithms for cryptography that use the same cryptographic keys for both encryption of plaintext and decryption of ciphertext in a communication.
The key is often named shared secret key.
There are two kinds of symmetric encryption:

1. Stream ciphers, which encrypts bits individually
2. Block ciphers, which encrypts an entire block of plaintext bits at a time with the same key

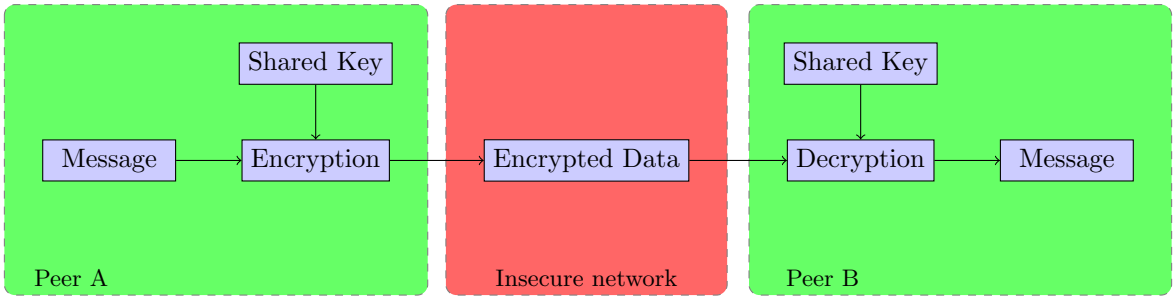

Figure 1.3: Introduction of a symmetric cipher operation

Figure 1.3 represents the process for encryption and decryption with a symmetric key. The shared key has already been exchanged between the two peers.
Peer A encrypts the message (plaintext) with this key and sends the result (the encrypted data or ciphertext) to peer B.
Then peer B uses the same key but to decrypt the encrypted data (ciphertext) and then reads the plaintext sent by the peer A.

No one who doesn't have this key can understand this message over the insecure network represents in red figure 1.3.

1.1.3.2 Asymmetric cipher algorithm

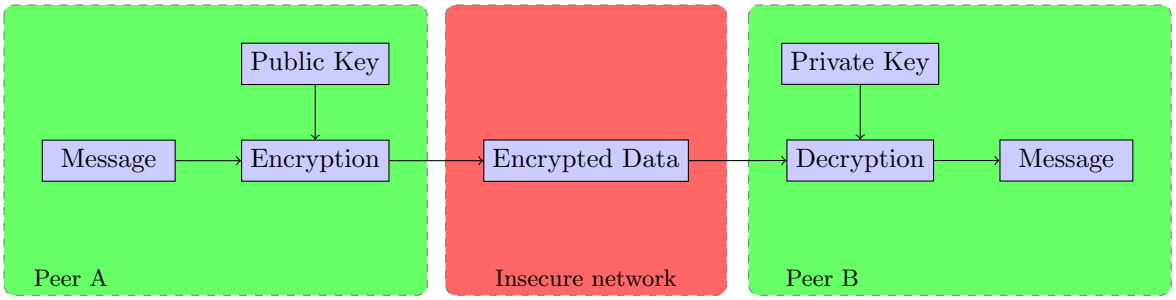

Figure 1.4: Introduction of an asymmetric cipher operation

Asymmetric algorithm is a method for encryption and decryption of messages with public and private keys.
One peer generates the private and public key together, keep the private key (meaning that he is the only one to have it) and sends the public key to every one he wants to communicate with.
With the public key everyone can encrypt a message, which no one can understand through the insecure network and no one can decrypt it, except this one who has the private key.

Figure 1.4 shows the principle of encryption and decryption with asymmetric algorithms.
Peer B is this one that has generated the public and private keys and has already sent the public key to peer A.
Peer A encrypts the message with the public key of peer B. This message is than sends to peer B over the insecure network.
Thanks to the private key can peer B decrypts the message of peer A.

Through the principle of private and public keys, asymmetric algorithm allows privacy (like symmetric algorithm) but integrity of data too, because only this one who creates the keys has the private key for decryption.

1.1.4 Diffie-Hellman

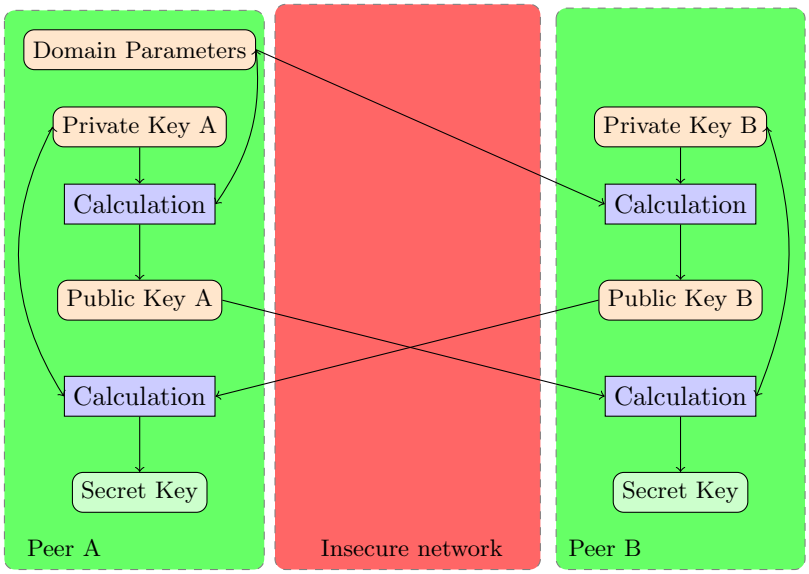

Figure 1.5: Introduction of a Diffie-Hellman operation

Diffie-Hellman Key Exchange establishes a shared secret key between two peers that can be used for secret communication for exchanging data over an insecure network.
The principle of Diffie-Hellman Key Exchange is to begin with asymmetric keys and to finish with symmetric key.
It makes sure that both parties participate in the generation of the symmetric key.

Figure 1.5 shows the principle of the Diffie-Hellman key exchanges.
Peer A creates the domain parameters and a private key only (not the public).
With the domain parameters and the private key can peer A calculates his public key.
He sends than the domain parameters and his public key to peer B.
Peer B generates a private key too and calculates his public key with the domain parameters from peer A and its own private key.
Peer B sends than his public key to peer A.
To finish, each one calculates the shared secret (which is the same for the twice) with the public key of the other peer and it's own private key.

The shared secret is the same for the two peers and can be used for encryption and decryption.

1.2 SSL/TLS protocol

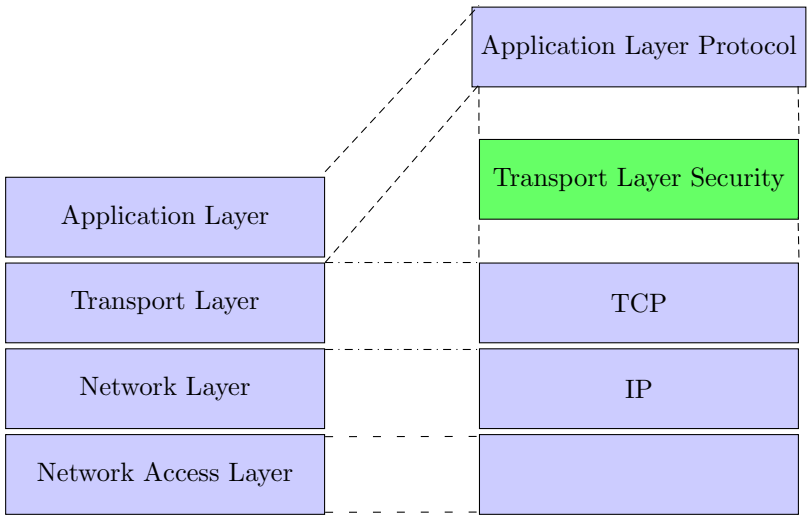

Figure 1.6: TLS protocol in OSI model

Transport Layer Security (TLS) is a client/server protocol that provides different basic security services for the communication between peers:

- Authentication (both peer and data origin authentication) services
- Connection confidentiality services
- Connection integrity services (without recovery)

This security layer is situated between the transport and the application layer on the OSI model (see figure 1.6) This security protocol is often uses in:

- E-commerce website for secured transaction and client authentication access
- Remote access
- Web browsers to browse the Internet
- Simple Mail Transfer Protocol (SMTP)
- Virtual Private Network (VPN)

1.2.1 Handshake protocol

Explain Handshake protocol

Client Hello

Server Hello

ChangeCipherSpec

Application Data

2 Motivation

In the institute of reliable Embedded Systems and Communication Electronics (ivESK) is a project named emb::TLS which has the goal to use the TLS protocol (see chapter 1.2) in embedded systems.

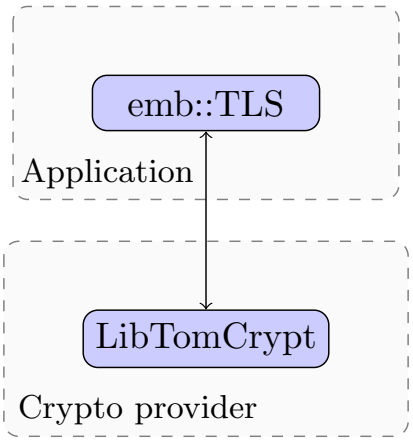

Figure 2.1: emb::TLS's project

For this, a cryptographic provider (LibTomCrypt) is used for the part of cryptographic calculation needed in the application.
This cryptographic provider is an open-source cryptography software library.

Problems with this implementation is that only LibTomCrypt is supported as cryptographic providers, meaning that no other libraries can be used without changing the complete implementation for emb::TLS.

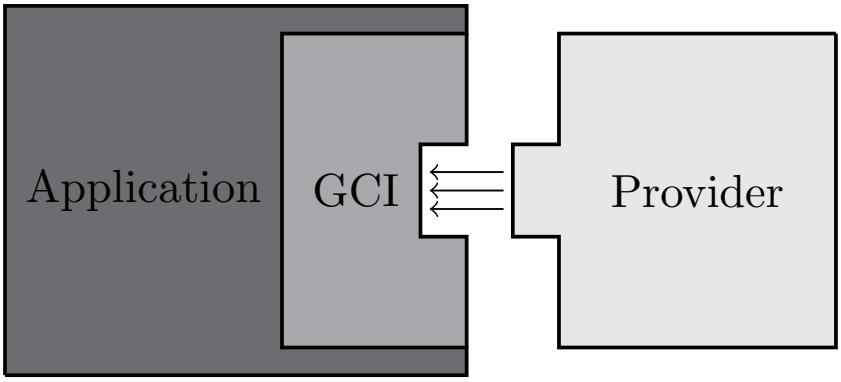

Figure 2.2: Goal of the new implementation

The goal of the project is therefore to create an interface, a Generic Cryptographic Interface (GCI), to have a base of the existing cryptography services and to have the possibility to easily add other providers only by changing some lines in the interface, instead of the complete application.
Through to this new interface other new cryptographic algorithms may be easier to add in the interface and to use in the application.
As shown in figure 2.2, the interface is implemented in the application and nothing has to be changed, except if other mathematical calculation has to be changed in a cryptography algorithm, like another kind of hash.

The requirements for this Generic Cryptographic Interface (GCI) are listed below:

1. No hidden states shall be used in the interface, meaning that the behavior of functions should only be effected by the input parameters.
   All parameters written in input of the function will be used for the cryptographic algorithm and nothing else.
2. Different cryptographic providers may be used for the cryptographic calculation.
   That could be open-source cryptographic software libraries or hardware-based cryptographic modules.
3. Interaction between the provider and the application shall be enabled for the key by a key management services, meaning that the key generated by the provider should be stored to have the possibility to use it by the application (like sending it to another peer) and keys coming from another peer shall be stored too, meaning that this key shall be used by the provider.
   to interact between the application and the provider.

3 Generic Cryptographic Interface

As explained in chapter 2, this interface should be implemented between the application and the cryptographic provider in the project emb::TLS for example. The interface has to be optimized for other projects too.
That's why this interface should have a basis of cryptography which can be easily adaptable for other projects and easily modifiable for other providers.
Some contraints are added for the interface, which are:

- The link between the keys which could coming from the application and needed in the provider and inversely.
- No hidden states should be written in the function of the interface, meaning that no default parameters for an algorithm has to be written internally of a function.

In several books and on the internet (see Bibliography) the most important part of cryptography are listed below:

- Hash, see 1.1.1 for the definition
- Signature, see 1.1.2 for the definition
- Cipher (symmetric (see 1.1.3.1 for the definition) and asymmetric (see 1.1.3.2 for the definition),
- Diffie-Hellman, see 1.1.4 for the definition

What is it - why it's use - how to use - advantage - disadvantage

3.1 Context

As explained above, no hidden states should be written in the function of the interface, meaning that no default parameters for an algorithm has to be written internally of a function.

For this constraint the principle of context is used.

Through the context, a configuration of an algorithm could be done, meaning that in the application part, different parameters for this algorithm can be choosen and this configuration is saved internally in the interface (see figure 2.2).
An ID of where is the configuration saved is returned to the application.
When an update has to be done, the ID is given and the interface knows that the update has to be done with this configuration.

The cryptographic algorithms which the principle of context is used are:

- Hash algorithm
- Signature algorithm
- Cipher algorithm (symmetric and asymmetric)
- Diffie-Hellman

Through the context, only the parameters added in the configuration are used. No hidden states is therefore implemented.

The disadvantage of the context is that the user should have good knowledges of cryptography to configure correctly an algorithm.

The others cryptographic services which don't need a context is because no configurations are needed to be saved.

What is it - why it's use - how to use - advantage - disadvantage

3.2 Cryptographic services

3.2.1 Hash

The hash algorithm is split into 3 functions in the interface:

1. Configuration of the algorithm
2. Update of a data
3. Calculation of the digest (finish)

What is it - why it's use - how to use - advantage - disadvantage

Configuration

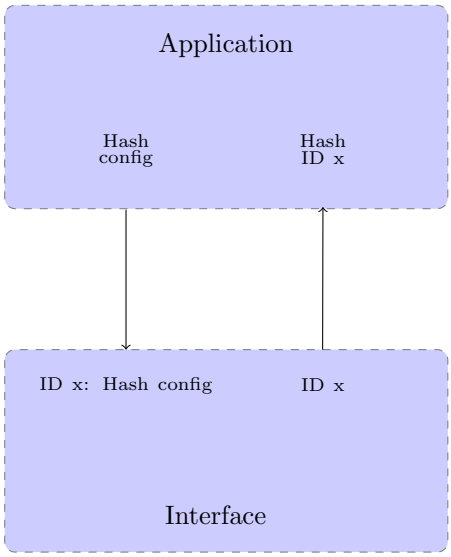

Figure 3.1: Hash - configuration

First part of the hash, it allows to configure the algorithm by only passing which hash algorithm will be used like:

- MD5
- SHA1
- SHA224
- SHA256
- SHA384
- SHA512

As shown in figure 3.1, this configuration is than saved in the interface and an ID of where it's saved is returned to the application.

Update

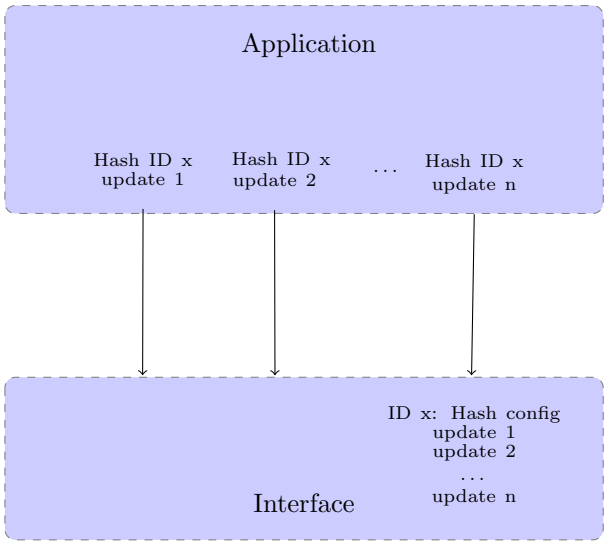

Figure 3.2: Hash - update

When the configuration is done, several updates can be done.

The principle on an update is to add a data which we want to hash.

As shown in figure 3.2 the ID received in the configuration part has to be used to add a data. Through this ID, the interface knows that the data has to be hashed with the configuration saved at this ID.

Finish

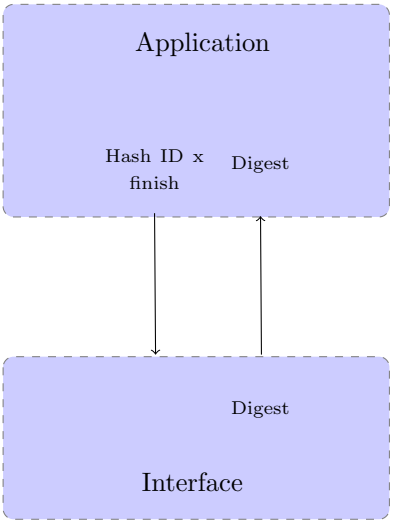

Figure 3.3: Hash - finish

Last part of the hash is the calculation of the digest

As shown in figure 3.3, by passing the ID which contains the configuration and all the updated data, the interface will, through the provider, calculate the digest.

One of the disadvantage of this part is when the digest is calculated, all the updated data and the confguration are lost, meaning that we cannot used them again to calculate another hash with other data.

This problem is solved in chapter 3.3

3.2.2 Generate key pair

To use other parts of cryptography like signature and cipher, keys should be generated.
Only key pairs are generated, meaning that only public and private keys and no symmetric/shared key.

To get a symmetric key see chapter symmetric cipher (**??** or diffie-hellman (3.2.5).

The type of key pair could be:

- RSA key pair
- DSA key pair
- ECDSA key pair (Elliptic curve)

For the configuration of a key pair the type of key pair has to be choosen with one listed above. Each one have different configuration:

- For a RSA key pair, the size of the key should be configured
- For a DSA key pair, the domain parameters should be configured (see the documentation of the interface in the Appendix for more details)
- For ECDSA key pair, the type of elliptic curve has to be given

The keys are returned with an ID. For more details to how to get the keys see the key management 3·4

What is it - why it's use - how to use - advantage - disadvantage

3.2.3 Signature

As some spefications of certain provider, the signature has two possibilities of use:

1. generate a signature
2. verification of a signature

Each one is split into three functions:

- Configuration of the signature
- Update data
- Generate a signature/Verify a signature

Through this algorithm the Message Authentication Code (MAC) can be used too.

What is it - why it's use - how to use - advantage - disadvantage

Configuration

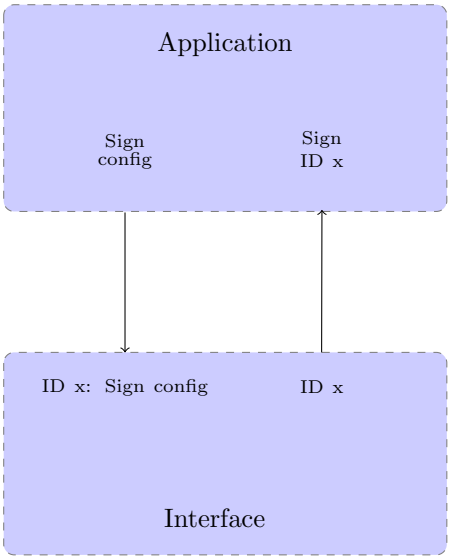

Figure 3.4: Signature - configuration

For the generation and verification of a signature this part is the same (only the name of the function changes).

First should the signature be configured.
Several parameters has to be configured which are:

- the signature/MAC algorithm, which could be:
  - RSA
  - DSA
  - ECDSA
  - MAC
  - CMAC
  - HMAC
- A hash algorithm, if in the updated data should be first hashed before signed, or if the HMAC algorithm is used
- The padding, if the RSA algorithm is used
- The block mode, the padding and the initialization vector, if the CMAC algorithm is used
- The private key, if RSA, DSA or ECDSA is used to generate a signature or the public key if the same algorithm are used to verify a signature

As shown figure 3.4, when the configuration is done, this one is saved in the interface and an ID of where is this configuration saved, is returned to the application.

Update

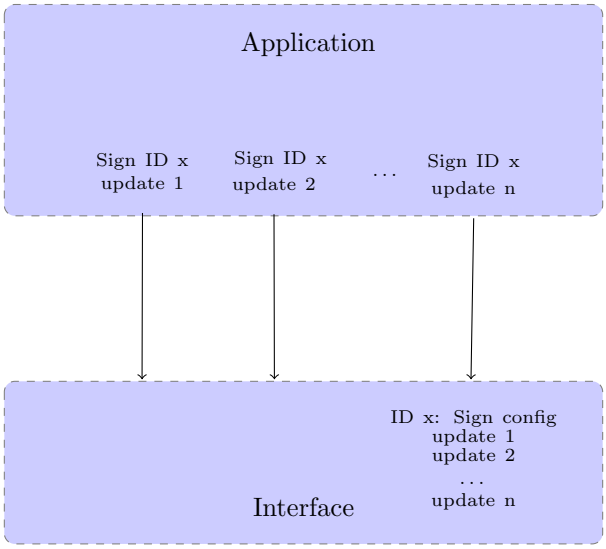

Figure 3.5: Signature - update

When the configuration is done, several updates could be done.

The principle of the update is to add data which will be used to generate or verify a signature.

As shown in figure 3.5, to use the correct configuration, the ID returned when the configuration is saved, should be used.

Finish

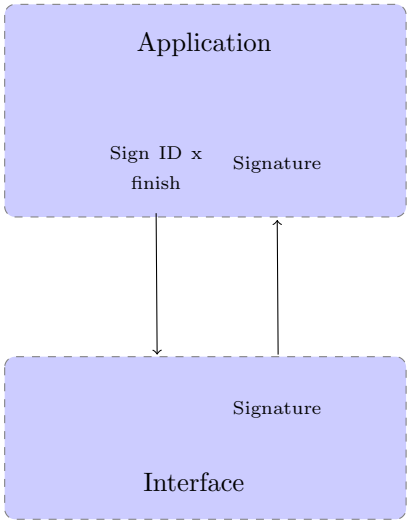

Figure 3.6: Signature - finish

In this part the generation anf the verification are different.

For the generation , the whole updated data and the configuration will be signed with the private key added in the configuration.

For the verification, the signature we want to verify should be add to the function.
Than the updated data will be signed, but with the public key.
Than the added signature (which is done with the private key) could be compared with the “signature” computed to verify.

The most important part of the verification is that the private key, which the signature is done and the public key for the verification, should be generated together.

3.2.4 Cipher (symmetric and asymmetric)

A cipher, as explained in **??**, which could be symmetric or asymmetric, is an algorithm for encrypting and decrypting data.

This concept is therefore used in the interface and split into three main functions which are:

- Configuration of the cipher
- Encryption of a plaintext
- Decryption of a ciphertext

Configuration of the cipher

Several parameters are to be configure for the cipher algorithm and depends particularly of which cipher algorithm is used.
The cipher algorithm is split into three main algorithm:

- Symmetric stream cipher algorithm
  Today very deprecated but is however implemented in the interface if comparaison has to be done.
  Only the RC4 stream cipher is implemented in the interface.
  Other stream cipher can however be easily added in the interface if needed.
  Nothing more as the algorithm has to be configured for the use of it.

- Symmetric block cipher algorithm
  Three kinds of symmetric block cipher algorithm are used today and therefore implemented in the interface:

  - Data Encryption Standard (DES)
  - 3DES, three subsequent DES encryptions
  - Advanced Encryption Standard (AES)

  Each symmetric block cipher algorithm needs a mode of operation, named block mode in the interface, which depends of the size of data we want to encrypt.
  The block mode implemented in the interface are:

  - Electronic Code Book mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - Cipher Feedback mode (CFB)
  - Output Feedback mode (OFB)
  - Counter mode (CTR)
  - Galois Counter Mode (GCM)

- Asymmetric algorithm
  Only the RSA algorithm is implemented for this part of the interface.
  In practice to use the RSA algorithm this one should use a padding to increase the security of it.
  The most known padding in the Public-Key Cryptography Standard (PKCS) and is of course implemented in the interface.

The most important thing for a cipher is of course the key!
For a symmetric cipher, stream or block, the key is only a shared key.
For an asymmetric cipher, if an encryption will be done, a public key should be added. If a decryption will be done, a private key should be added to the configuration.

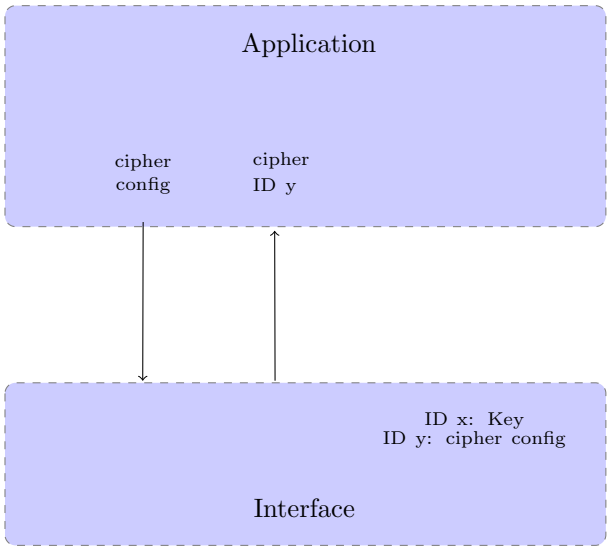

Figure 3.7: Cipher - configuration

Thenceforth the configuration is done this one should be sent to the interface which will saved it in a context.
The interface returned an ID of the context, which corresponds of where is saved the configuration, if this one should be use in the future.
The key added to the function is an ID of a the key which is already saved in the interface.
This principle is shown in figure 3.7

Encryption of a plaintext

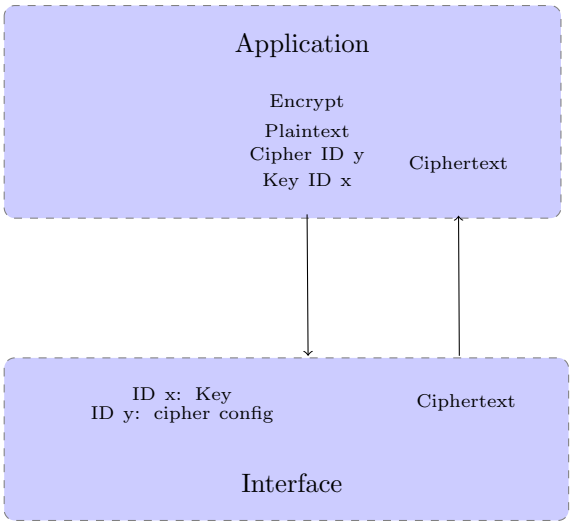

Figure 3.8: Cipher - Encryption

Thenceforth the configuration is done, a encryption can be done.
To encrypt a data, the ID of the context (where is the configuration saved) should be added to the function with the data to encrypt (plaintext).
The interface will, through a provider, calculate the ciphertext of the plaintext with the configuration saved previsously in the context.

This principle is shown figure 3.8

Decryption of a ciphertext

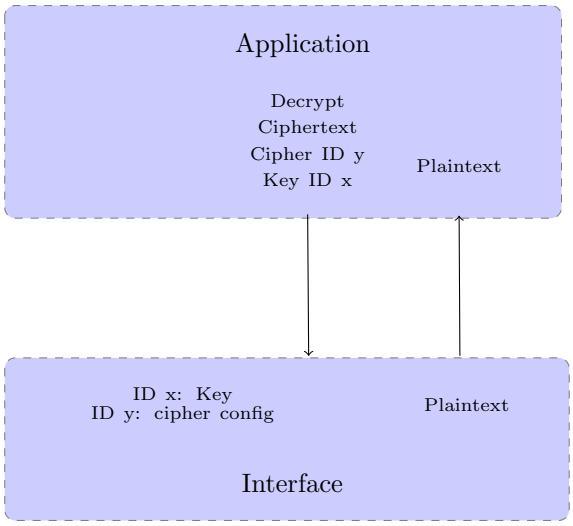

Figure 3.9: Cipher - Decryption

Thenceforth the configuration is done, a decryption can be done.
To decrypt a data, the ID of the context (where is the configuration saved) should be added to the function with the data to decrypt (ciphertext).
The interface will, through a provider, calculate the plaintext of the ciphertext with the configuration saved previously in the context.

This principle is shown figure 3.9

3.2.5 Diffie-Hellman

What is it - why it's use - how to use - advantage - disadvantage

3.2.6 Random number generator

What is it - why it's use - how to use - advantage - disadvantage

3.3 Clone of context

As explained in 3.2.1 and 3.2.3 when the digest (for the hash) and the signature (for the signature) is calculated, no more data can be added to the context.
This is a problem for the use of this interface in TLS projects (emb::TLS for example).
Several solutions was proposed which are:

1. Use two contexts at the same time.
   This wasn't very efficient, because we should know at the beginning the number of times a digest will be calculated, which determines the amount of context we have to create at the beginning.

2. Create a context when the digest is calculated.
   The disadvantage of this idea was that the whole data use previously has to be saved. For systems, like embedded systems, with memory constraints, this is not practicable.

3. Clone the context. This is the kept solution.

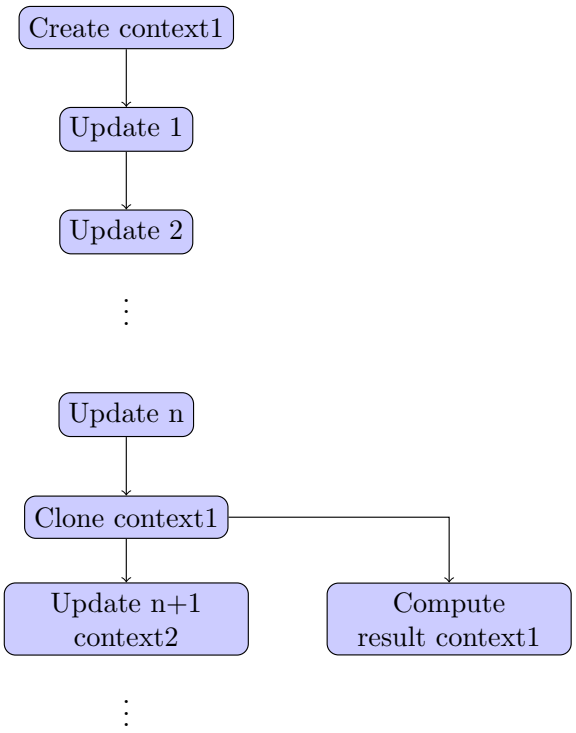

Figure 3.10: Context - clone example

As shown figure 3.10, when we need to compute a result, but the whole data added previously are needed for a future result, the solution is to clone the context, meaning that the whole data added and the configuration is copied in another context.
Then one context could be used to compute the result and the other one to add other data when needed.

What is it - why it's use - how to use - advantage - disadvantage

3.4 Key management

Add a short introduction of key management

What is it - why it's use - how to use - advantage - disadvantage

Generate a key

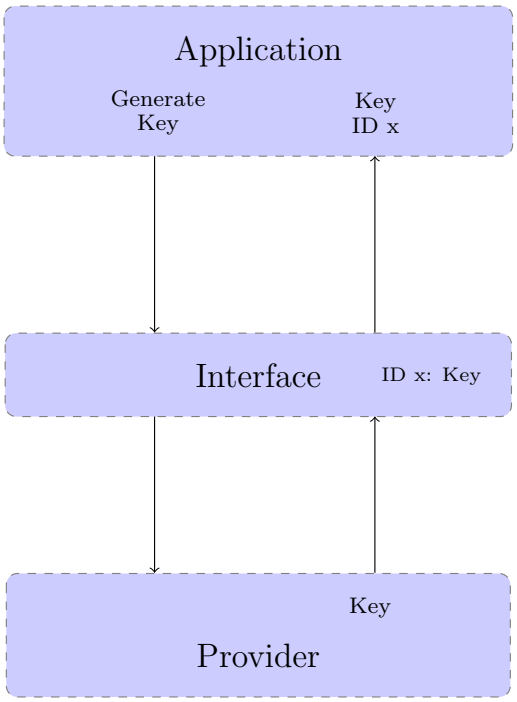

Figure 3.11: Key management - generate a key

Explain figure 3.11

Get a key

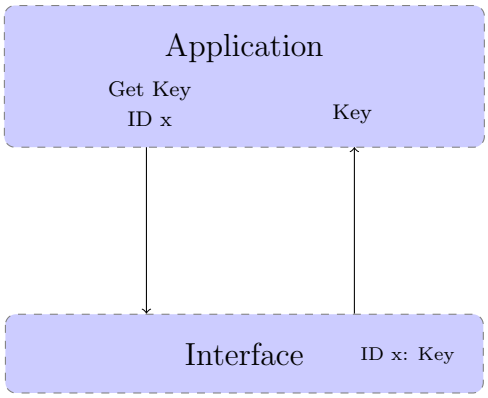

Figure 3.12: Key management - get a key

Explain figure 3.12

Delete a key

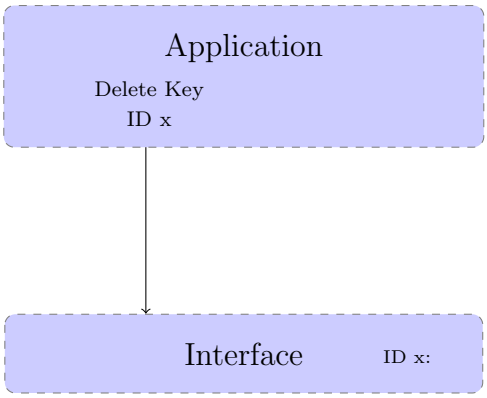

Figure 3.13: Key management - delete a key

Explain figure 3.13

Put a key

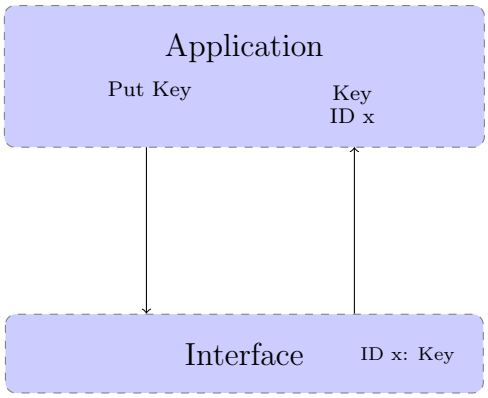

Figure 3.14: Key management - put a key

Explain figure 3.14

4 Cryptography in the TLS protocol

4.1 Handshake Protocol

4.2 Cryptographic parts in the Handshake Protocol

5 Implementation

5.1 Interface for an application

5.2 Provider for the interface

6 Results

6.1 embetterTLS as client

6.2 embetterTLS as server

7 Conclusion

7.1 Achieved work

7.2 Future work

Bibliography

Bibliography

Books

[1] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Berlin Heidelberg, Berlin; Heidelberg [u.a.], 2. corr. printing edition, 2010.

[2] Ph.D. Rolf, Oppliger. *SSL and TLS: Theory and Practice*. Artech House, eSECURITY Technologies; Beethovenstrasse 10; CH-3073; Gümligen; Switzerland, 2009.

Internet

[3] Wikipedia. Cryptographic hash function. URL: `https://en.wikipedia.org/wiki/Cryptographic_hash_function`.

[4] Wikipedia. Diffie-hellman key exchange. URL: `https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange`.

[5] Wikipedia. Digital signature. URL: `https://en.wikipedia.org/wiki/Digital_signature`.

[6] Wikipedia. Public-key cryptography. URL: `https://en.wikipedia.org/wiki/Public-key_cryptography`.

[7] Wikipedia. Symmetric-key algorithm. URL: `https://en.wikipedia.org/wiki/Symmetric-key_algorithm`.

Appendices

Appendix A

Documentation Interface

This documentation lists all functions use in the Generic Cryptographic Interface (GCI) and explains step by step the working of each cryptographic services definied in the interface.