COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

DIGITAL SIGNATURES

SECURITY PROPERTIES, CONSTRUCTIONS AND APPLICATIONS

Autor: Javier Borrachero Prieto

Director: Ante Derek

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Digital Signatures. Security properties, constructions and applications.

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2019/20 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

[illegible]

Fdo.: Javier Borrachero Prieto Fecha: 20/07/2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

[illegible]

Fdo.: Ante Derek Fecha: …20…/ …07…/ …2020…

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

DIGITAL SIGNATURES

SECURITY PROPERTIES, CONSTRUCTIONS AND APPLICATIONS

Autor: Javier Borrachero Prieto

Director: Ante Derek

Madrid

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1. INTRODUCTION

1.1 TOWARDS DIGITAL SIGNATURES

In the analogic world, the only and unique way to identify a person with a document is through a signature. A signature is a handwritten representation that a person uses in documents as a proof of identity.

With the arrival of digital world, arise a need to proof documents digitally. To satisfy this urgent need many different valid methods have been developed gathered in electronics signatures. An electronic signature is any electronic equivalent of a handwritten signature which allows to validate the content of an electronic message. Within this wide field, appear biometric signatures, user-password systems and, digital signatures as well.

A digital signature is a cryptographic mechanism that provides a recipient assurance over both integrity of a message and, of the person who claims to have generated the message.

With the digital world explosion, digital signatures have become one of the most important tools in cybersecurity and is widely used to sign documents. Application for digital signatures are uncountable inside the digital world, highlighting cryptocurrencies, digital certificates in e-commerce security or legal contracts for software updates, among others.

Digital signatures employ public-key cryptography techniques (asymmetric cryptography), whose main characteristic is that the person who sends the message generates two keys, one private that only he knows and with which to sign the message, and one public, in reach of anyone, that allows the recipient to validate and authenticate the received message.

There are many public-key encryption algorithms among which two stands out: RSA and DSA. These algorithms are today considered secure if used correctly with sufficient key sizes. [1]

1.2 PRESENT AND FUTURE

Digital signatures and its encryption methods are complex, they require wide mathematical knowledge based on modular arithmetic and primes factorization.

This complexity is necessary in order to avoid cyberattacks being, thus, completely reliable. Nowadays, the security of these digital signatures lies in public and private key generation through prime numbers. [2]

However, the arrival to the world of a brand-new technology is threatening any cryptography method, including digital signature. This technology is named Quantum Computing.

Quantum computing combine binary information technology world along with quantum physics allowing to develop a new technology of several orders of magnitude higher, in terms of power, that the current one. This means that quantum computing will be able to break many encryption methods, such as digital signatures, since its power will be such, that it could solve primes' factorization.

This new technology is today in development and there are already few computers based on quantum physics, such as **IBM** Q 53, the greatest computer from IBM, which employs 53 qubits (notation for quantum bits). It is a fact that quantum computing represents the future of computer science, and hence, greatest companies all over the world are already investing in its development. [3]

Still, it is important to highlight that this new world is yet in an initial phase and requires years of development to become a reality for commercial purposes and a threat to current cryptography.

1.3 MOTIVATION

Cybersecurity, as many other fields in information technology, is one of the most demanded fields in the world. The capacity of development is still huge, and the complexity of digital signature encryption methods requires a deep look and comprehension of itself, as well as the mathematical background.

The ability to develop digital signature allows the acquisition of several capabilities from the technology that is surrounding us every day, something that is still unknown for the vast majority of the population. It also provides a greater understanding of the coming world in terms of IT development. Along with the preparation needed for the arrival of new technologies that sooner or later will substitute the currents one.

It is apparent that to advance and go one step further, it is under urgent necessity to master current developed concepts and the steps that have been taken until today.

1.4 OBJECTIVE

The aim of this project is to investigate and understand concepts, properties, and applications of digital signatures, highlighting RSA and DSA algorithms.
The project also provides a better understanding of why the arrival of quantum computing is threating all encryption algorithms, especially digital signatures.

With all the gathered knowledge, put together a digital signature the prototype for some purpose.

A brief description of the protype is the following:
Build a command line tool to verify digital signatures on EU official electronic journal documents.

1.5 PROJECT STRUCTURE

The structure of the project is as follows:

- Firstly, digital signatures basic principles and its security properties are explained.
- Secondly, RSA and DSA algorithms are presented including how they work currently.
- Thirdly, quantum computing concepts and how they could forge digital signature methods is discussed.
- Later there is an analysis of a prototype, its necessary means of construction, such as formats, libraries, and the different python's commands.
- Consequently, the python code will be included, along with the results.
- Finally, a conclusion is added.

1.6 MATHEMATICAL BACKGROUND

Modular arithmetic:

To have a better understanding of digital signature methods, it is necessary to understand its mathematical background. The mathematics behind it are basically based on Integer factorization, prime generation, and Modular arithmetic. Both prime generation and integer factorization follow easy concepts, however modular arithmetic can be unknown for many.

In mathematics, modular arithmetic is a system of arithmetic for integers, where numbers ‘wrap around' when reaching a certain value, named modulus. It is widely known as clock arithmetic because 12-hour clocks use modular arithmetic of modulus 12.

This arithmetic is built upon the congruence of relationships of integers: “a and b are in the same "class of congruence" module n, if both leave the same remainder if we divide them by n, or, equivalently, if the difference a - b is a multiple of n.” [4]

$$a \equiv b \ (mod\ n)$$

$$63 \equiv 83 \ (mod\ 10)$$

As both 63 and 83 leave the same remainder (3) when divided by 10.

As mentioned before, the most famous application of modular arithmetic is 12-hour clocks. If the time is 8:00 now, then 5 hours later it will be 1:00. Simple addition would result in 8 + 5 = 13, but as clocks "wrap around" every 12 hours, the clock starts over and displays 1:00.

CHAPTER 2. TECHNOLOGY DESCRIPTION

The main technology used for this project was Python 3. All programming tools were developed within this software.

The interface employed for python was Jupyter Notebooks.

Apart from python, XMLdsig syntax was used for digital signature documents. It is commonly applied to sign data, typically, XML documents.

CHAPTER 3. DIGITAL SIGNATURES

3.1 HISTORICAL BACKGROUND

The history of digital signatures comes from cryptography. The word cryptography might be associated with modern encryption such as e-mail encryption, cryptocurrencies, or the famous attack against the German Enigma encryption machine during the World War II.
However, the origins of cryptography go back to about 2000 B.C. when non-standard secret hieroglyphics were used in ancient Egypt. Since then, cryptography has been used in many ways throughout time, with civilizations that developed their own cryptography, from ancient Greece to the Roman Empire or Sparta with its famous *stycale*. [1]

Cryptography is the science of secret writing with the aim of hiding the content of a message. This field can be split in three types of cryptography: Symmetric ciphers, asymmetric ciphers, and protocols. [1]

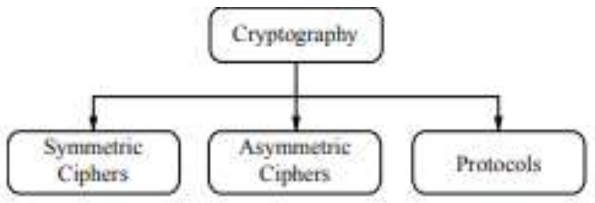


Figure 1 *Overview of cryptography field [1]*

Symmetric ciphers: define the idea that comes up to mind when thinking of cryptography: Two parties have a secret encryption-decryption method for which they share a secret key. All cryptography mentioned before and until 1976 was exclusively based on symmetric cryptography. They are widely used today for data encryption and integrity checking of messages. [1]

Asymmetric ciphers (public-key encryption): Here is where digital signatures take place. An asymmetric cipher or public-key algorithm was introduced by Whitfield Diffie, Martin Hellman, and Ralph Merkle in 1976. It is a completely different type of cipher. In public-key cryptography, there are two keys. A private key, that only the owner of the key knows, and a public key that is available for anyone. The application for public-key algorithms are mainly digital signatures and key establishment. [1]

Cryptography protocols deal with the application of both symmetric and asymmetric ciphers. It is used for secure Internet communication among parties. An example of protocols is The Transport Layer Security (TLS) scheme, widely used in almost every web browser. [1]

Back again with Digital signatures, soon after its discovery in 1976, Ronald Rivest, Adi Shamir and Len Adleman invented RSA algorithm, which became the basis of primitive digital signature schemes. The first widely marketed software package to offer digital signature was Lotus Notes 1.0, released in 1989, which used the RSA algorithm. Since then, more complex, and reliable digital signature algorithms have been developed such as Rabin signatures, GMR signature scheme or DSA algorithm. [2]

3.2 PRINCIPLES OF DIGITAL SIGNATURES

Symmetric algorithms arise under the necessity of encrypt messages and one can believe that by just encrypting data, a message will be absolutely secure and safe. However, in practice there are other security prerequisites that need to be satisfied such as **authentication** and **integrity** that will be discussed in section 3.3.

Thus, asymmetric algorithms were born to provide security prerequisites where symmetric algorithms failed. To have a better outlook the following example would help:

Assume we have two communication parties, Alice and Bob. Alice wants to a sell gaming computer $2000 worth with "no return" policy online. Bob is looking for a new computer, so he contacts Alice. To make a formal order, they both share a secret key K_{ab} with which they encrypt/decrypt the buying order. If only Alice and Bob know the key, they can be reasonably sure that an attacking third party has not changed the message in transit. Until now, symmetric algorithms would perfectly work.

But now imagine that before the computer delivery takes place, Bob finds from a different seller the same computer $500 cheaper. Then, as Bob wants this computer, he claims he never ordered Alice's computer, so Alice has no choice but to sue him. In front of the judge, Alice's lawyer uses as evidence the order sent from Bob. However, Bob's lawyer has an ace up his sleeve, he can ensure that Alice is also in possession of the private key K_{ab}, so she could have made a fake order and, in fact, she has a high incentive to do so. Consequently, the judge has no choice but to uphold Bob´s claim as there are not enough evidences to proof that Bob ordered the computer.

The problem lies in the symmetric algorithm idea: as both parties know the private key, everything Bob does, Alice could do too. To solve this problem, asymmetric algorithm appeared. If we repeat the same example, now when Bob wants to make the computer order, he must sign it using his private key; thus, the lawyer can proof with no-doubt that the order indeed, belongs to Bob.

The basic idea of digital signatures is that the sender uses a private key (k_{pr}) to sign a message (x), then with the private key he also generates a public key (k_{pub}). The recipient uses the matching public key to verify the signed message (s).

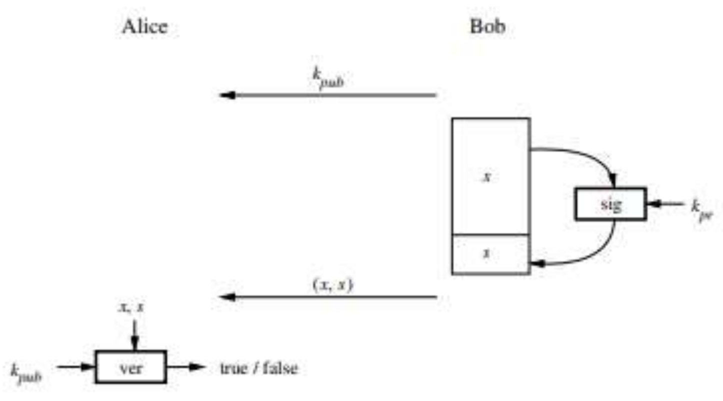


Figure 2 *Principle of digital signatures [1]*

3.3 SECURITY PROPERTIES AND TYPES OF ATTACKS

Security properties: [2]

Cryptography ciphers are basically defined by their security properties. Digital signature, as mentioned in the previous section, provides certain properties that make them very reliable and, therefore, widely used in e-commerce world. Those are:

- **Integrity**: Ensures that the message has not been modified in transit. Confidentiality provides that the content of the message is hidden, but an attacker could change the encrypted message without understanding it. Therefore, when a message is signed, any change in the message will automatically invalidate the signature verification.

- **Message authentication:** Ensures that the sender of the message is authentic. A valid signature shows that only the creator of the private key could sign the message, then, it is the proof of authentication.

- **Nonrepudiation:** As explained before, it ensures that the sender of the message cannot deny the generation of it. Furthermore, access to the public key only does not enable a fraudulent party to fake a valid signature. This property defines the main difference among digital signatures and other cryptography encryption methods.

Types of attacks: [2]

The aim of an attacker is to forge signatures, and to do so, the attacker attempts to produce signatures which would be accepted. The set of criteria to break a digital signature scheme is the following:

1. **Total break**: An attacker is able to compute the private key.
2. **Selective forgery**: An attacker is able to create a valid signature for some type of messages
3. **Existential forgery**: An attacker is able to forge a signature for at least one message. In this case, the adversary has no control of the forged message

There are basically two attacks against digital signature encryption schemes:

1. **Key-only attacks:** The adversary only knows the public key.
2. **Message attacks:** In this case, the adversary examines signatures corresponding to chosen messages. There are three types of messages attacks:
   a. **Known-message attack:** The attacker has signatures for messages which he knows but are not chosen by him.
   b. **Chosen-message attack:** The attacker obtains a valid signature for a message he chose before.
   c. **Adaptive chosen-message attack:** In this case, the adversary uses the signer as an oracle. He requests signatures of messages which depend on signer's public key. He may also request signatures of messages which depend on previously obtained signatures or messages.

3.4 HASH FUNCTIONS

[1] Hash functions are cryptographic functions used in protocols to map data of arbitrary size to fixed size values. They compute a digest value, which is a short representation of the original message.
In cryptography, hash functions play an important role in the practical use of digital signatures. Digital signature algorithms are based on large primes factorization and modular arithmetic. Signing or verifying large messages based on these two concepts requires a great number of bits and enormous amount of energy and time. This problem becomes double because not only the message is sent but also the signature.

Furthermore, hash functions deal with digital signature's security limitations. Without hash functions, the only way to send large messages would be by signing sequence of blocks individually. Although an adversary cannot modify individual blocks, this process would allow him to remove individual blocks or re-order them.

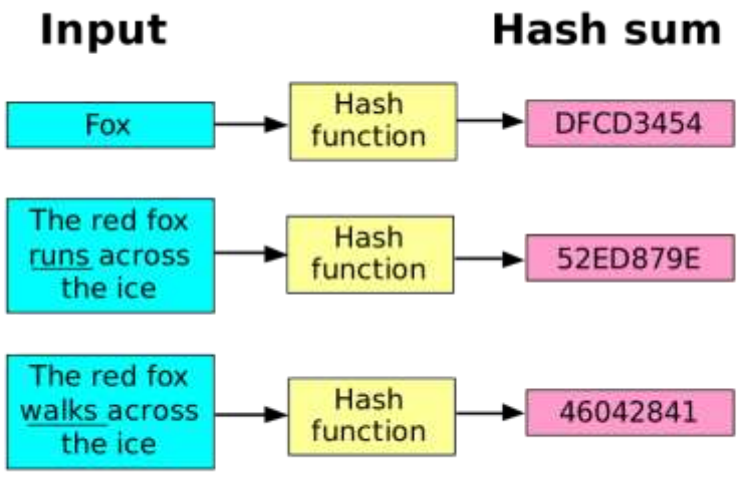


***Figure 3** Hash function example*

Therefore, in practice, thanks to its efficiency and security properties digital signatures algorithms are mainly applied to hash values instead of original messages.

The mathematical background of hash function's computation is based on block cipher techniques. One possible method of building hash functions is the following:

The message *m* is divided into m_i blocks of fixed size. Every m_i block is encrypted with block chipper *e*. Then the encrypted block $e_{(mi-1)}$ is XOR with the original message:

$$H_i = (e_{Hi-1}) * (m_i) \oplus (m_i)$$

The most famous hash functions for digital signatures are SHA family such as SHA-1 or SHA-2 (SHA-256 or SHA-512 with message digest lengths of 256, and 512 bits, respectively).

3.5 RSA SIGNATURE SCHEME

[1] RSA signature scheme was invented at MIT in 1977 by Ronald Rivest, Adi Shamir and Leonard Adleman. The security of this algorithm relies on the **integer factorization** problem. As mentioned in the introduction 1.2, the factorization problem becomes irresolvable for today's computers when choosing large primes. This scheme has three processes which work as follows:

Recovering the example used in section 3.2, there is a communication between two parties, the sender Bob, and the recipient, Alice. Bob wants to send a message *m* to Alice.

1. **Key generation for RSA:**

   Bob first of all, creates an RSA public key and its corresponding private key. To do so he chooses randomly two large primes numbers p and *q*. The size of these two numbers should be large enough, approximately between 512 and 1024 bits inclusive being 768 bits the recommended size [2].

   Then Bob must compute:

$$n = p * q$$
$$\phi = (p-1)(q-1)$$

   Thirdly, he must select a random *e*, 1 < *e* < ϕ, such that gcd(*e*, ϕ) = 1.

   Now Bob chooses his **private key**, it must be an integer ***d***, 1 < *d* < ϕ such that:

$$e * d = 1\ (mod\ \phi)$$

   Bob's **public key is *n* and *e*, and his private key is *d*.**

2. **Signature generation for RSA:**

   Once Bob has generated both private and public key, he signs his message m as follows:

$$s = m^{d}\ (mod\ n)$$

   S is the signed message that Bob sends to Alice.

3. **Signature verification for RSA:**

   Alice must be able to verify this message using the public key (*e*, *n*). To do so, Alice compute [2]:

$$m = s^{e}\ (mod\ n)$$

   Proof of RSA scheme works:

   Since: $e * d = 1\ (mod\ n) \rightarrow$

   And following Euler's Theorem [5]:

$$s = m^{d}\ (mod\ n) \rightarrow s^{e} = m^{ed} = m\ (mod\ n)$$

An easy example to see RSA scheme is the following:

Public key: p = 47; q = 59; e = 17.

m = 0805

$$n = 47 * 59 = 2773$$

$$\phi = (47-1)(59-1) = 2668$$

$$e * d = 1\ (mod\ \phi) \rightarrow d = e - 1\ mod\ \phi\ = 17 - 1\ mod\ 2668 = 157$$

Private key: d = 157.

$$s = 0805^{157}\ (mod\ 2773) = 542$$

Verification:

$$\mathrm{m} = 542^{17}\ (\mathrm{mod}\ 2773) = 805$$

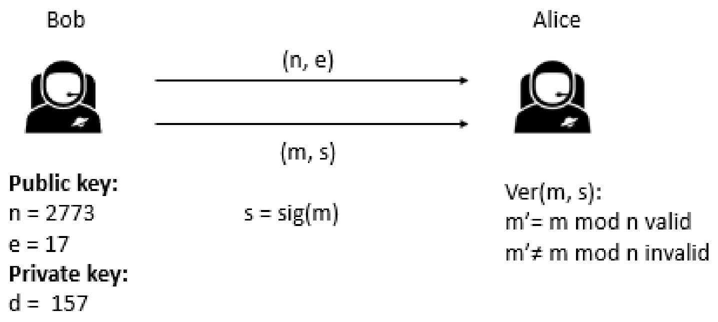


***Figure 4** RSA signature scheme*

Possible attacks on RSA signatures:

Total break:

If the attacker is able to factor the public number *n* into *p* and *q*, then he could compute ϕ and, thus, compute also *d*. If the adversary does so, he would completely break the system. As mentioned before, the signer must choose *p*, *q* large enough so no-one is able to factor *n* [2]

Existential Forgery:

The other possible attack is that the adversary is able to forge a signature for at least one random message *m*. However, to do so, the attacker must compute first the signature and then the message, which means that he is not able to generate a meaningful message. [2]

3.6 DSA SIGNATURE SCHEME

[1] The Digital Signature Scheme, proposed in August 1991, is a Federal Information Processing Standard for digital signatures. The National Institute of Standards and Technology (NIST) proposed DSA for use in their Digital Signature Standard (DSS). The algorithm has similitudes with RSA scheme, but while RSA is based on integer factorization, DSA is based on **discrete logarithm**. This scheme has the same three processes which work as follows:
Recovering again the example used in section 3.2, there is a communication between two parties, the sender Bob, and the recipient, Alice. Bob wants to send a message *m* to Alice.

1. **Key generation for DSA:**
   Bob first of all, creates an RSA public key and its corresponding private key. Selecting first a prime number q such that $2^{159} < q < 2^{160}$. Then he chooses t so that $0 < t < 8$ and selects a prime number p where $2^{511+64t} < p < 2^{512+64t}$, with the characteristic that q divides (p− 1).
   Consequently, Bob must create a generator α. To do so, he computes:

$$\alpha = g^{(p-1)/q} \ (mod \ p)$$

   In case α = 1; Bob must choose a new *g*.
   Finally, Bob chooses an integer a such that $1 \leq a \leq q - 1$, and computes:

$$y = \alpha^{a} \ (mod \ p)$$

   Bob's **public key is *p, q, α, y* and his private key is *a*.** [2]

2. **Signature generation for DSA:**

Once Bob has generated both private and public key, he signs the message *m* following these steps:

I. Select a random secret integer *k*, $0 < k < q$
II. First, compute $r = (\alpha^{k} \, mod \, p) \, mod \, q$
III. Second, compute $k^{-1} \, mod \, q$
IV. Third, compute $s = k^{-1}\{h(m) + a * r\} \, mod \, q$
V. Finally, Bob signature for *m* is the pair (r, s).

- *h(m)* corresponds to the hash function applied to message *m*. The DSS explicitly requires use of the Secure Hash Algorithm (SHA-1) explained in section 1.5. [2]

3. **Signature verification for DSA:**

Alice must be able to verify this message using the public key (*p*, *q*, α, *y*). To do so, first of all, she must verify that $0 < r < q$ and $0 < s < q$; If not, she rejects the signature [2].

Alice computes:

$$w = s^{-1} mod \, q \;\; and \;\; h(m)$$

$$u1 = w * \, h(m) \, mod \, q \qquad\qquad u2 = r * w \, mod \, q$$

$$v = (\alpha^{u1} * \, s^{u2} \, mod \, p) \, mod \, q$$

Then if $\boldsymbol{v = r}$, Alice accepts the signature, otherwise she must reject it.

Proof that the DSA scheme works:

$$s = k^{-1}\{h(m) + \, a * r\} mod \, q \quad \rightarrow \quad h(m) \equiv -a * r \, + \, k * s \, (mod \, q)$$

By multiplying both sides by *w* and then raising to the power of α yields:

$$w * h(m) \equiv w * \big(-a * r \, + \, k * s \, (mod \, q)\big) \rightarrow$$

$$w * (h(m) + \, a * r) \equiv \, k \, (mod \, q) \rightarrow$$

$$u1 \, + \, a * u2 \, \equiv \, k \, (mod \, q) \rightarrow$$

$$(\alpha^{u1} * \, y^{u2} \, mod \, p) \, mod \, q \, = \, (\alpha * k \, mod \, p) \, mod \, q$$

$$\boldsymbol{v = r}$$

An easy example to see DSA scheme is the following:

Public key: p = 59; q = 29; α = 3.

Private key: a = 7.

$$y = 3^7\ (mod\ 59) \equiv 4\ mod\ 59$$

h(m) = 26.

k = 10.

$$r\ =\ (3^{10} mod\ 59)\ mod\ 29 \equiv 20\ mod\ 29$$

$$s\ =\ \frac{1}{10} * (26 + 7 * 20) \equiv 5\ mod\ 29$$

Verification:

$$w\ =\ 5^{-1}\ \equiv 6\ mod\ 29$$

$$u1\ =\ 6 *\ 26\ \equiv 11\ mod\ 29 \qquad u2\ =\ 6 * 20\ \equiv 4\ mod\ 29$$

$$v\ =\ (3^{11} *\ 4^4\ mod\ 59)\ mod\ 29 = 20\ mod\ 29$$

$$\boldsymbol{v\ =\ r \rightarrow valid\ signature}$$

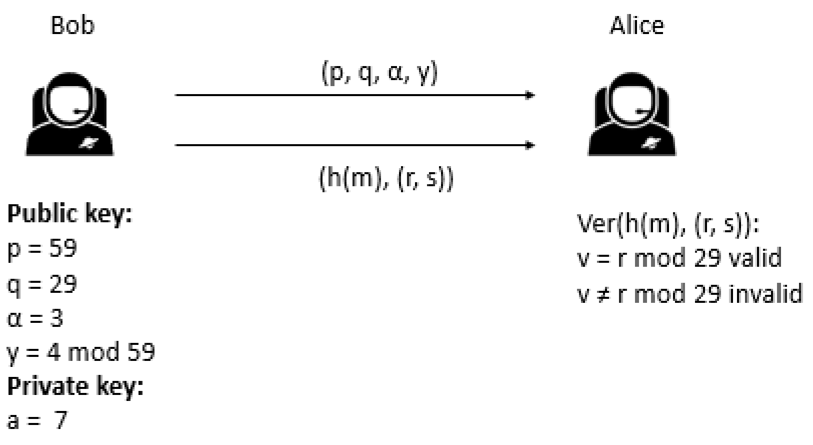


Figure 5 *DSA signature scheme*

3.7 APPLICATIONS

Before discussing possible applications for both algorithms, it is important to understand its advantages. Both RSA and DSA are the same strong algorithms, however they have the following differences in terms of performance:

- DSA is faster in generating keys than RSA
- RSA is faster in encrypting than DSA
- DSA is faster in decrypting than RSA

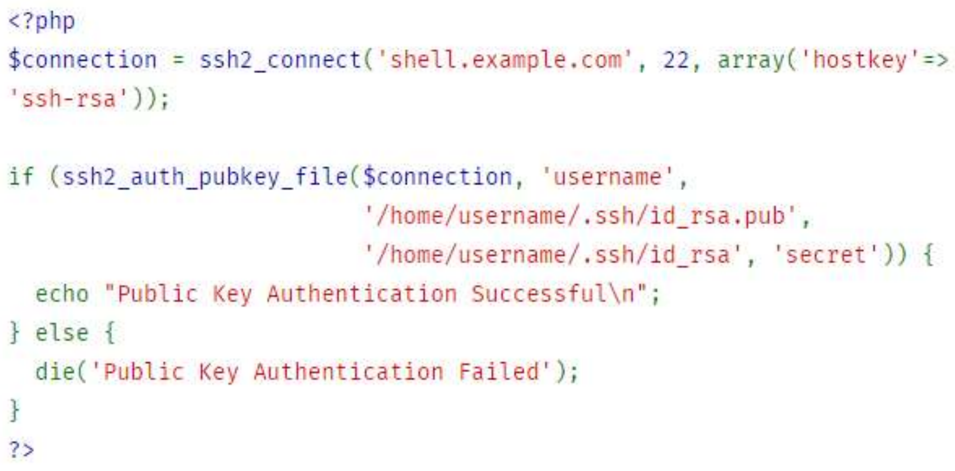
Therefore, DSA is the best choice for digital signing while RSA is better for verification of digital signatures. [5]

Having a better perspective of both algorithms, both signature scheme's application can be discussed.

RSA APPLICATIONS:

RSA covers most of the applications that require public-key encryption and digital signatures. Most famous applications are multinational financial service corporations such as VISA or MasterCard which facilitates electronic fund transfers through RSA signature scheme. Furthermore, inside the e-commerce world, there is also wide use, and platforms as Magento (for SSL) also employ this scheme [6].
Another application for RSA is server connections. PUTTY is a free terminal emulator which provides the user interface to access the files on an encrypted server. It uses RSA to authenticate users without password, by using public key previously generated. Here is referenced the open-source c code. [7]

Finally, an overview of RSA implementation is shown in PHP programming language. By using SSH2 (encrypted channel for logging into another computer over a network) and its PHP library, libssh2, the following code is obtained [8]:



```
<?php
$connection = ssh2_connect('shell.example.com', 22, array('hostkey'=>
'ssh-rsa'));

if (ssh2_auth_pubkey_file($connection, 'username',
                          '/home/username/.ssh/id_rsa.pub',
                          '/home/username/.ssh/id_rsa', 'secret')) {
  echo "Public Key Authentication Successful\n";
} else {
  die('Public Key Authentication Failed');
}
?>
```

Figure 6 *PHP Authentication using a public key [8]*

DSA APPLICATIONS:

As mentioned in section 3.5, DSA is a Federal Information Processing Standard (FIPS) for digital signatures. FIPS are publicly announced standards developed by the NIST for use in computer systems by non-military American government agencies and government contractors. [9]

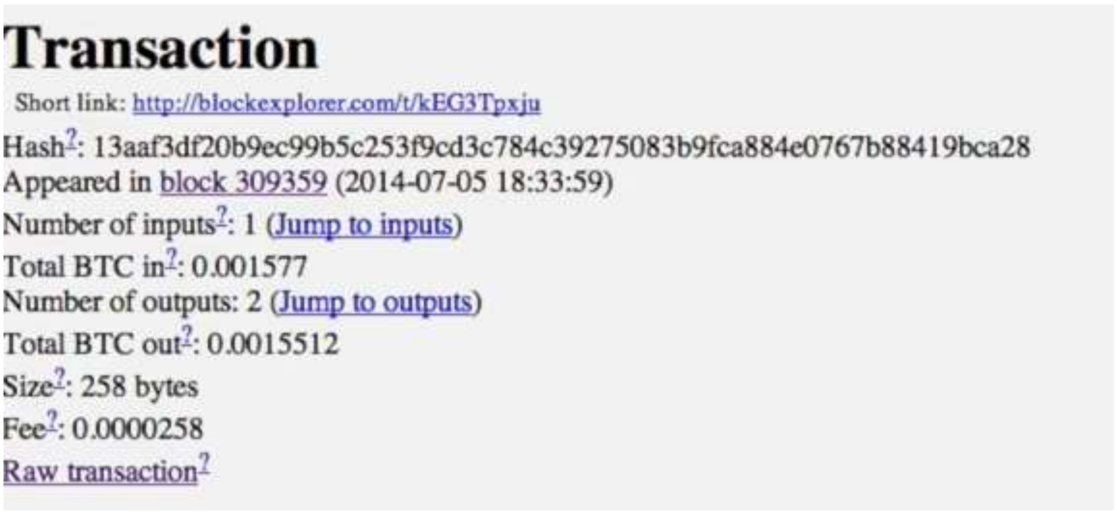
Furthermore, a small variance of DSA named ECDSA is used to authorize Bitcoin transactions. Bitcoin is by far the major cryptographic e-cash system used nowadays. Transactions work using blockchain which basically means that bitcoins are sent in blocks from one user to another one [10]. Every transaction includes:

The user who previously possessed the bitcoins as input.

Bitcoins one user wants to send as output.

The difference between them as bitcoin fee.

A great example of bitcoin transaction is the following [10]:

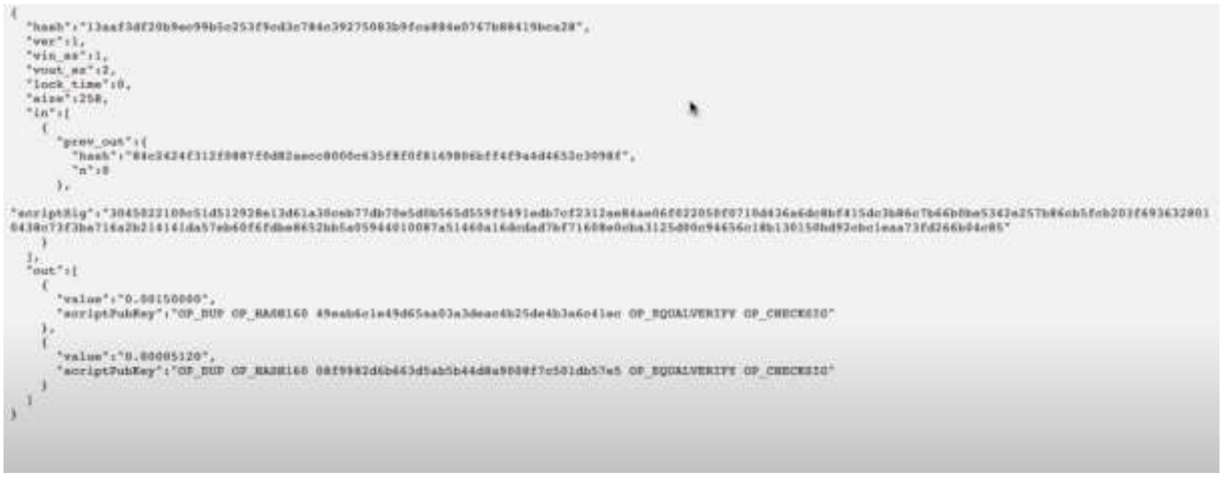


Transaction

Short link: http://blockexplorer.com/t/kEG3Tpxju

Hash?: 13aaf3df20b9ec99b5c253f9cd3c784c39275083b9fca884e0767b88419bca28
Appeared in block 309359 (2014-07-05 18:33:59)
Number of inputs?: 1 (Jump to inputs)
Total BTC in?: 0.001577
Number of outputs: 2 (Jump to outputs)
Total BTC out?: 0.0015512
Size?: 258 bytes
Fee?: 0.0000258
Raw transaction?

Figure 7 *Bitcoin Transaction example [10]*



```
{
  "hash":"13aaf3df20b9ee99b5c253f9ed3e784e39275083b9fea884e0767b88419bea28",
  "ver":1,
  "vin_sz":1,
  "vout_sz":2,
  "lock_time":0,
  "size":258,
  "in":[
    {
      "prev_out":{
        "hash":"84c2424f312f8887f0d82aecc8000e635f8f0f8149806bff4f9a4d4653c3098f",
        "n":0
      },
"scriptSig":"3045022100e51d512928e13d61a38ceb77db78e5d8b565d559f5491edb7cf2312ae84ae06f022058f0718d436a6dc8bf415dc3b86c7b66b0be5342e257b86cb5fcb203f693632801 0438e73f3ba716a2b214141da57eb60f6fdbe8652bb5a0594401008 7a51460a16dedad7bf71608e0eba3125d00e94656c18b130150bd92ebc1eaa73fd266b04c05"
    }
  ],
  "out":[
    {
      "value":"0.00150000",
      "scriptPubKey":"OP_DUP OP_HASH160 49eab6c1e49d65aa03a3deac4b25de4b3a6c41ec OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value":"0.00005120",
      "scriptPubKey":"OP_DUP OP_HASH160 08f9982d6b463d5ab5b44d8a9080f7c501db57e5 OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
}
```

***Figure 8** Bitcoin Transaction code [10]*

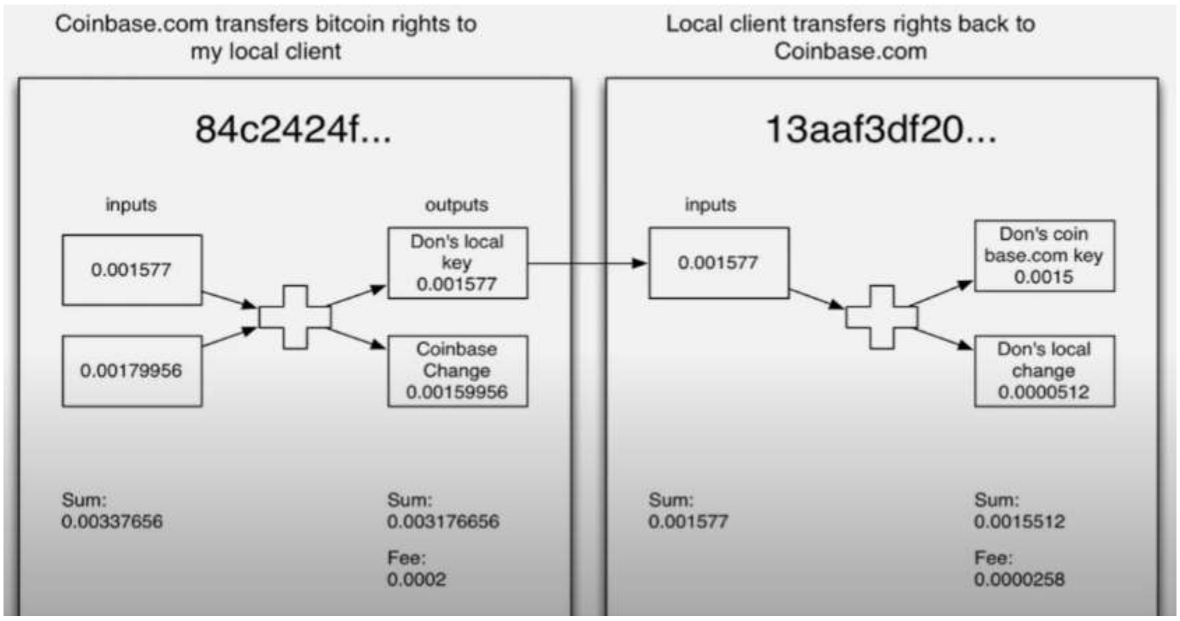

Coinbase.com transfers bitcoin rights to my local client
84c2424f...
inputs
outputs
0.001577
0.00179956
Don's local key 0.001577
Coinbase Change 0.00159956
Sum: 0.00337656
Sum: 0.003176656
Fee: 0.0002
Local client transfers rights back to Coinbase.com
13aaf3df20...
inputs
0.001577
Don's coin base.com key 0.0015
Don's local change 0.0000512
Sum: 0.001577
Sum: 0.0015512
Fee: 0.0000258

***Figure 9** Bitcoin Transaction scheme [10]*

3.8 QUANTUM COMPUTING THREAT

Quantum computing is the use of quantum-mechanical phenomena to perform computation. It is basically a combination of quantum physics and traditional binary computing. Computers that perform quantum computations are known as quantum computers which operate with quantum bits or **qubits**. To understand quantum computing it is necessary to start with its properties: **superposition** and **entanglement** [3].

Superposition: [11] It refers to a combination of two states that would ordinary describe independently such as the superposition of two musical notes. In quantum physics, state’s superposition occurs in particles like electrons which can be in two places at the same time.
This property is easily understandable by the famous experiment **Schrödinger's cat**:
The scenario presents a hypothetical cat box that may be simultaneously both alive and dead. The experiment is the following:
There is a closed box with a cat inside. This box is connected to a device which contains a radioactive substance with 50% probability of decaying. It is also connected to hydrocyanic acid that will poison the cat if the radioactive substance decays. In classical physics, the cat has equal probability to be alive or dead before we open the box.
However, in quantum mechanics, the state’s superposition will allow the cat to be both alive and death before we open it. Then, when someone open the box, the cat can only be in one of those states. [12]
Returning to quantum computing, this property allows qubits to be in both binary states 0 and, but also in a combination of them. This leads to new logic gates and therefore, new algorithms. [11]

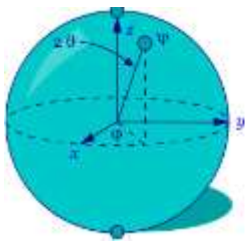


Figure 10 *Qubit representation*

Entanglement: Is a famously quantum phenomenon describing behavior infeasible in classical world. Entangled particles behave together as a system in a way such that the quantum state of each particle of the pair or group cannot be described independently of the state of the others, including when the particles are separated by a large distance. Basically, entangled particles are able to communicate with each other no matter the distance between them. [11]

Thanks to superposition, a qubit can be in multiple places simultaneously. The number of superpositions that a quantum computer can hold at the same time is given by 2^n, where n is the number of qubits. This means that while the information that in a classical computer can contain at one time is n, a quantum computer is able to contain 2^n.

An example will show this better:
Assume a classical computer of 3 bits and a quantum computer of 3 qubits. While the classical computer can contain 3 bits of information such as 011, due to superposition, the quantum computer can contain $2^3 = 8$ bits. The following figure illustrates it:

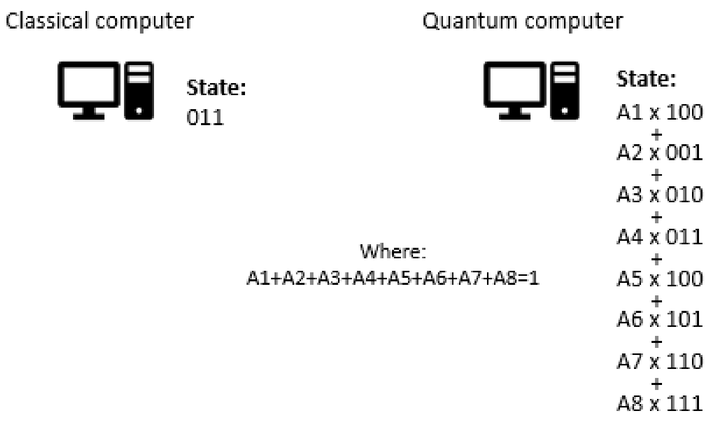


Figure 11 *Superposition in Quantum Computers*

The power of quantum computer arises here. If one increments in 1 the number of bits of a classical computer, it will be able to contain n+1 bits of information. However, in a quantum computer the information will be exponentially incremented to 2^{n+1}.

Having a better outlook of what quantum computing means, it is time to discuss why it is a threat to Digital Signature algorithms, such as RSA

RSA algorithm, as mentioned in section 3.4, is based on integer factorization. Classical computers are not able to solve large integer factorizations because the way a classical computer works obey them to try every single possibility, one by one, to find the desired one. If the primer number are large enough a classical computer will never be able to try enough combinations, thus, find the suitable.
Whereas, in quantum computing the same process becomes exponentially faster due to qubit superposition. A quantum computer is able to analyze 2^n possibility at the same time. There is already a quantum algorithm named Shor algorithm that is supposed to break public-key encryptions. IBM has already launched its IBM Q53, the most powerful quantum computer of 53 qubits. [3]

However, is not all that easy. This technology has yet many issues to solve. Firstly, the more qubits a computer has, the more complicated it is to keep entanglement stable. Another problem of quantum computing is named decoherence. This basically means that the system must be isolated from its environment and possible interactions with the external world. These two problems are delaying the availability of strong quantum computers for commercial use, although the firsts units have already been launched.

Thus, quantum computing is already in development and it will be a threat to current public-key encryption schemes, but to be so, it still needs years of development and investigation.

CHAPTER 4. PROTOTYPE CONSTRUCTION

4.1 PROTOTYPE DESCRIPTION

The goal of this chapter is to elaborate a prototype that implements all the knowledge discussed previously.

This prototype must solve the following task: Build a program to verify digital signatures on EU official electronic journal documents.

4.1.1 OFFICIAL JOURNAL OF THE EUROPEAN UNION

"The Official Journal of the European Union (OJ) is published daily (from Monday to Friday regularly, on Saturdays, Sundays, and public holidays only in urgent cases) in all of the official languages of the EU. Since 1 July 2013 only the electronic edition of the OJ is authentic and produces legal effects". [13]

In this website, user can find available a pdf to the OJ for each day. Along with each pdf, the web provides an XML e-signature to verify the document. Both files can be download from the web. CheckLex is the official website to verify OJ official documents.

4.1.2 CHECKLEX

"CheckLex is an application that enables you to verify the electronic signature and authentic character of the electronic edition of the Official Journal of the European Union (e-OJ). If verification succeeds, you are guaranteed that the electronic edition is authentic and has not been changed since the date of its signature." [14]

This website works as follows:

1. The user downloads from the OJ official website both OJ pdf and e-signature XML document.
2. Save them to his computer or device.
3. In CheckLex website uploads both documents to verify the signature.
4. If the verification fails, the web will show an error message on screen.

4.1.3 PROTOTYPE TASK

The prototype task is to elaborate a command tool in python that performs the same way CheckLex does.

To do that, the program must be able to import a pdf document and an XML e-signature. Then there must be a verification code which reads the signature and verifies the OJ document. The program must show on screen if success, if not, it has to print an error message.

4.1.4 PROGRAMMING DOCUMENTATION

As mentioned previously, prototype's digital signature is provided in XML format. To understand properly the python code, it is required basic knowledge of XML documents. An XML document contains information composed of elements orderly packaged. Elements are the building blocks of XML. These divide the document into a hierarchy of sections. [15]

In digital signatures, there is a standard for XML signature documents. This standard scheme is defined for every XML signature and it has the following components:

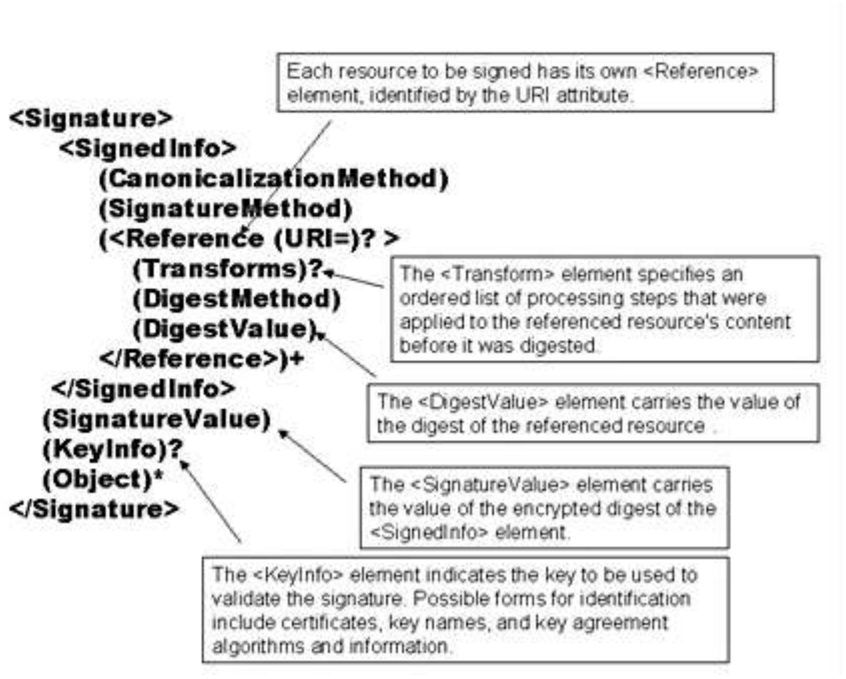

Each resource to be signed has its own <Reference> element, identified by the URI attribute.
<Signature>
<SignedInfo>
(CanonicalizationMethod)
(SignatureMethod)
(<Reference (URI=)? >
(Transforms)?
(DigestMethod)
(DigestValue)
</Reference>)+
</SignedInfo>
(SignatureValue)
(KeyInfo)?
(Object)*
</Signature>
The <Transform> element specifies an ordered list of processing steps that were applied to the referenced resource's content before it was digested.
The <DigestValue> element carries the value of the digest of the referenced resource.
The <SignatureValue> element carries the value of the encrypted digest of the <SignedInfo> element.
The <KeyInfo> element indicates the key to be used to validate the signature. Possible forms for identification include certificates, key names, and key agreement algorithms and information.

Figure 12 *The Components of an XML Signature [15]*

Figure 12 shows XML signature standard, where elements are organized in an XML tree structure. User can access all subelements by class inheritance (parent-child relationship) through any programming language. [15]

The programming language used for this prototype is python 3.0. To verify documents in python language several libraries must be imported. These libraries are*: lxml, signxml* and *haslib.*

Lxml: From this library it is imported etree. Etree allows to parse an XML document. The goal of parse is to transform XML into a readable code. It checks for proper format of the document.

It also allows to access the different packaged elements inside an XML document. [16]

Signxml: This library is an implementation of XML Signature standard in python. It allows to sign and verify XML signatures. It needs *lxml* to work with XML data. [17]

Hashlib: With hashlib module user can implement many different secure hash and digest algorithms such as sha512. [18]

4.2 DESIGN

The prototype design is divided in the following steps:

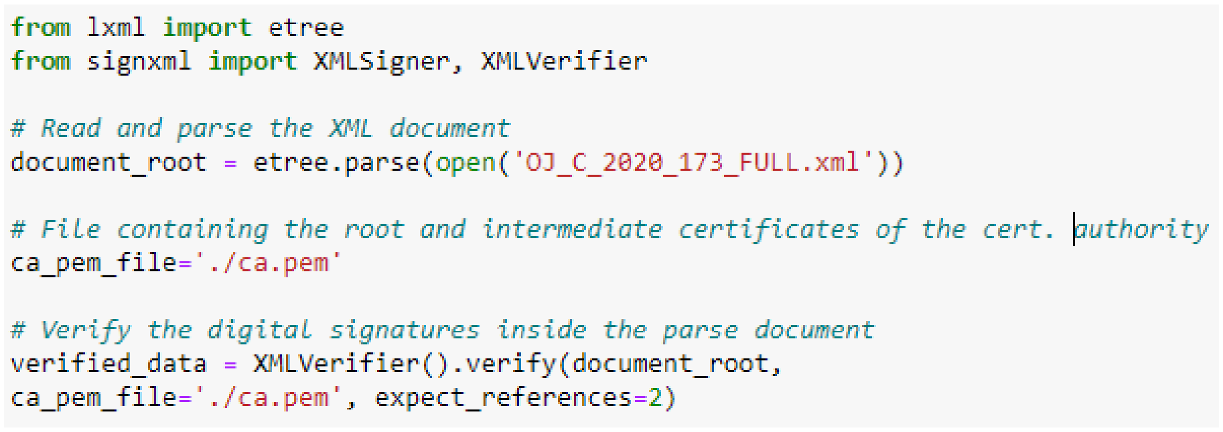
1. Verify XML signature document from OJ
2. PDF Document verification

1. Verify XML signature document from OJ:

The first step is to verify the XML document. Firstly, the document is imported and parsed so it can be read by python. Then a file containing the root and certificates is also imported. This file is named ca.pem.

Using XMLVerifier from *signxml* one can verify the digital signature inside the document. From the OJ XML document it is expected to verify two references.

The code is shown below:



```python
from lxml import etree
from signxml import XMLSigner, XMLVerifier

# Read and parse the XML document
document_root = etree.parse(open('OJ_C_2020_173_FULL.xml'))

# File containing the root and intermediate certificates of the cert. authority
ca_pem_file='./ca.pem'

# Verify the digital signatures inside the parse document
verified_data = XMLVerifier().verify(document_root,
ca_pem_file='./ca.pem', expect_references=2)
```

Figure 13 *Verification of XML signature document from OJ*

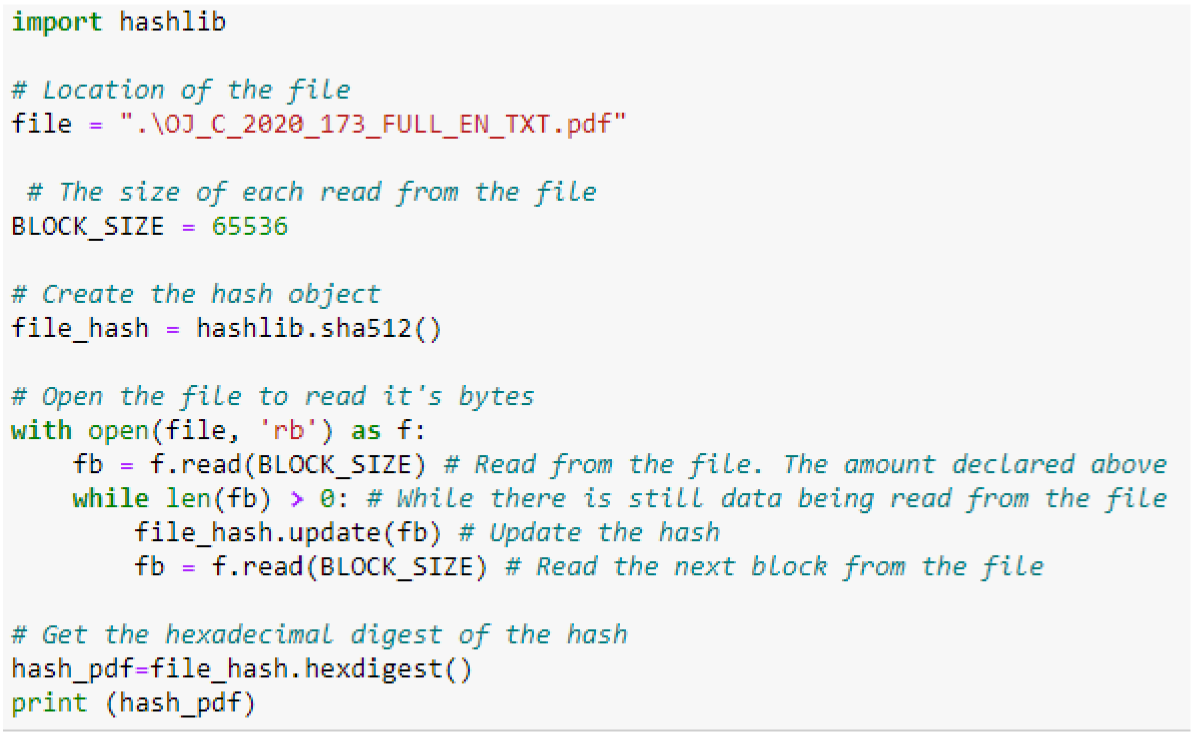
2. PDF Document verification:

Once the XML document has been verified, is time to verify the PDF. To do so, the following process has been developed:

- Calculate the hash of the given document
- Fetch the hash value from the verifed_data
- Compare the two values

- Calculate the hash of the given document:

Hashlib is the library that carries out the hash value calculation. This prototype works with sha512 hash objects. The following code imports the PDF file and hashes it in blocks of 65536 bits. Then it transforms the hash value into hexadecimal:



```python
import hashlib

# Location of the file
file = ".\OJ_C_2020_173_FULL_EN_TXT.pdf"

 # The size of each read from the file
BLOCK_SIZE = 65536

# Create the hash object
file_hash = hashlib.sha512()

# Open the file to read it's bytes
with open(file, 'rb') as f:
    fb = f.read(BLOCK_SIZE) # Read from the file. The amount declared above
    while len(fb) > 0: # While there is still data being read from the file
        file_hash.update(fb) # Update the hash
        fb = f.read(BLOCK_SIZE) # Read the next block from the file

# Get the hexadecimal digest of the hash
hash_pdf=file_hash.hexdigest()
print (hash_pdf)
```

Figure 14 *Calculation of the hash of the PDF document*

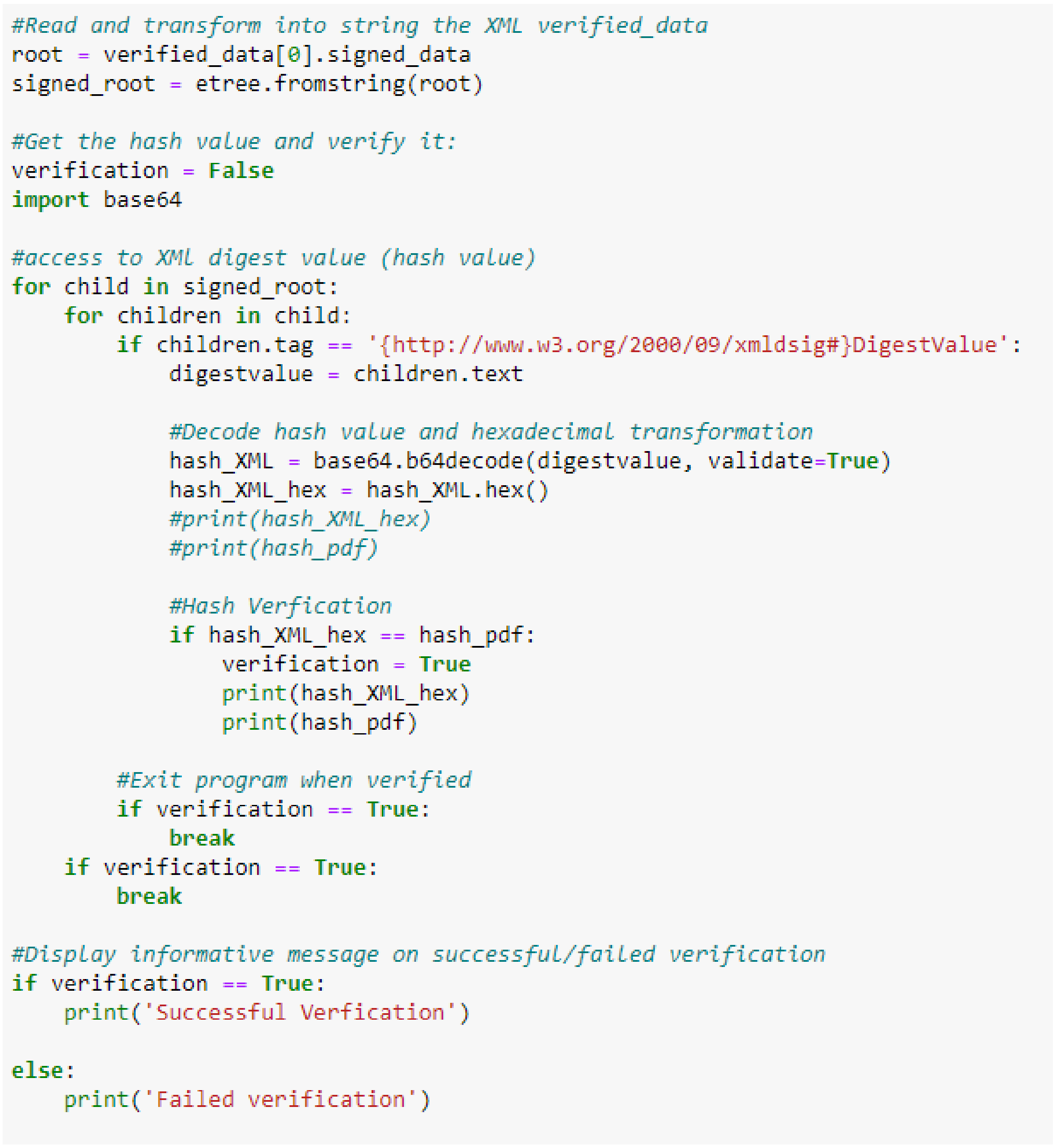
- Fetch the hash value from the verifed_data:

To fetch the hash value from the XML document, is it required to implement *etree* from *lxml*. From verified_data one can access any information by parent-child relationship. In this case, the hash value is allocated at <DigestValue>. Then, program decodes the digest value and turns it into hexadecimal so it can be compared with the hash value obtained from the PDF.

It is important to mention that in the XML document there are hash values for all published OJ documents (one document for each European Union language). Thus, the program obtains all hash values using *for* loops.

- Compare the two values:

Finally, once both XML and PDF hash values have been fetched, they are compared one by one, if at some point the two values matches the program exits both *for* loops and display an informative message ‘Successful verification’. In contrary if the program never matches values it displays a ‘failed verification’ message.



```python
#Read and transform into string the XML verified_data
root = verified_data[0].signed_data
signed_root = etree.fromstring(root)

#Get the hash value and verify it:
verification = False
import base64

#access to XML digest value (hash value)
for child in signed_root:
    for children in child:
        if children.tag == '{http://www.w3.org/2000/09/xmldsig#}DigestValue':
            digestvalue = children.text

            #Decode hash value and hexadecimal transformation
            hash_XML = base64.b64decode(digestvalue, validate=True)
            hash_XML_hex = hash_XML.hex()
            #print(hash_XML_hex)
            #print(hash_pdf)

            #Hash Verfication
            if hash_XML_hex == hash_pdf:
                verification = True
                print(hash_XML_hex)
                print(hash_pdf)

        #Exit program when verified
        if verification == True:
            break
    if verification == True:
        break

#Display informative message on successful/failed verification
if verification == True:
    print('Successful Verfication')

else:
    print('Failed verification')
```

Figure 15 *XML Hash value and verification*

CHAPTER 5. TESTS AND RESULTS

This chapter shows how the prototype performs following in the steps described in previous chapter. Attached is the display window for each code section.

In [1]:

```
from lxml import etree
from signxml import XMLSigner, XMLVerifier

# Read and parse the XML document
document_root = etree.parse(open('OJ_C_2020_173_FULL.xml'))

# File containing the root and intermediate certificates of the cert. authority
ca_pem_file='./ca.pem'

# Verify the digital signatures inside the parse document
verified_data = XMLVerifier().verify(document_root,
ca_pem_file='./ca.pem', expect_references=2)

# Verified XML data is here
print(verified_data[0])
```

```
VerifyResult(signed_data=b'<ds:Manifest xmlns:ds="http://www.w3.org/2000/0 9/xmldsig#"
Id="manifest">\n <ds:Reference URI="c_17320200520bg.pdf">\n
<ds:DigestMethod              Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"></ds:
DigestMethod>\n                        <ds:DigestValue>ky+t2TnYdM4qEAezp6ojcxjA+Me0WwPe62Y
6cT3gFgI0fxEDGmEroxuB9vJ2cr7X2kKVi36dm+rVLfHNt7CCkQ==</ds:DigestValue>\n
</ds:Reference>\n    <ds:Reference URI="c_17320200520cs.pdf">\n        <d
s:DigestMethod              Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"></ds:Di
gestMethod>\n        <ds:DigestValue>wrEZEFZiea8G0EoWx4iQ/FSYStqcC1xrOJZv/
iUeE78mi9/BWlQRSD/KQXE07jh7A9fRV2RRGL8VH4uAM1J4UA==</ds:DigestValue>\n
</ds:Reference>\n    <ds:Reference URI="c_17320200520da.pdf">\n        <d
s:DigestMethod              Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"></ds:Di
gestMethod>\n        <ds:DigestValue>HGdIp+gZ6djyy0cNtD+qLwT+Cor5Gsk1FWpyn
XnOH8/36uY8/9pZtOlIxTUGt68yeqNBqb6D+8gUYAfCStss0g==</ds:DigestValue>\n
</ds:Reference>\n    <ds:Reference URI="c_17320200520de.pdf">\n        <d
s:DigestMethod              Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"></ds:Di
gestMethod>\n        <ds:DigestValue>WUi2QMEPQl4mRgX8CA5nMvIU5IvNc5hJHNyvW
I1FNFSPx00duPCT6yHvqwyU1Az075laaV0Yy9SnpF5NECyzlQ==</ds:DigestValue>\n
```

```
import hashlib

# Location of the file
file = ".\OJ_C_2020_173_FULL_EN_TXT.pdf"

  # The size of each read from the file
BLOCK_SIZE = 65536

# Create the hash object
file_hash = hashlib.sha512()

# Open the file to read it's bytes
with open(file, 'rb') as f:
        fb = f.read(BLOCK_SIZE) # Read from the file. The amount declared above
        while len(fb) > 0: # While there is still data being read from the file
                file_hash.update(fb) # Update the hash
                fb = f.read(BLOCK_SIZE) # Read the next block from the file

# Get the hexadecimal digest of the hash
hash_pdf=file_hash.hexdigest()
print(hash_pdf)
```

```
891caba6bfe8ac00b56bfc6c5e1636427d7df83e82a5b8ab1e3a3e8d254179afc5121e9522
615b2335517ea17a502669dc0354acfbbc35d32ad6a627d6c72c1c
```

```
#Read and transform into string the XML verified_data
root = verified_data[0].signed_data
signed_root = etree.fromstring(root)

#Get the hash value and verify it:
verification = False import base64

#access to XML digest value (hash value)
for child in signed_root:
    for children in child:
        if children.tag == '{http://www.w3.org/2000/09/xmldsig#}DigestValue':
            digestvalue = children.text

            #Decode hash value and hexadecimal transformation
            hash_XML = base64.b64decode(digestvalue, validate=True)
            hash_XML_hex = hash_XML.hex()
            #print(hash_XML_hex) #print(hash_pdf)

            #Hash Verfication
            if hash_XML_hex == hash_pdf: verification = True
                print(hash_XML_hex) print(hash_pdf)

        #Exit program when verified
        if verification == True: break
    if verification == True: break

#Display informative message on successful/failed verification
if verification == True:
    print('Successful Verification')

else:
    print('Failed verification')
```

```
891caba6bfe8ac00b56bfc6c5e1636427d7df83e82a5b8ab1e3a3e8d254179afc5121e9522
615b2335517ea17a502669dc0354acfbbc35d32ad6a627d6c72c1c
891caba6bfe8ac00b56bfc6c5e1636427d7df83e82a5b8ab1e3a3e8d254179afc5121e9522
615b2335517ea17a502669dc0354acfbbc35d32ad6a627d6c72c1c
Successful Verification
```

CHAPTER 6. CONCLUSIONS

In the XXI century world, humanity is completely surrounded by day-by-day improving technology that makes our lives easier. This technological revolution is growing up exponentially letting us take enormous steps in a short period of time. But along with any advance, it is of urgent necessity to ensure its security and viability. Thus, nowadays cybersecurity has become one of the most important fields in IT. Within this field there is a large number of subfields among which digital signatures stands out.

As discussed throughout this thesis, digital signatures are widely used in almost every digital asset. Documents, transactions, or connections to servers are daily activities for any person. Hence, anyone who works with a computer in an office or at home, or anyone who is interested in IT should know at least what digital signatures are, and how they assure all operations that one carries out.

On the other hand, for those who work in IT such as software engineers, data scientists, developers, or obviously anyone who works in cybersecurity, it is strongly recommended to understand not only what digital signatures are, but also how these algorithms are deployed and their performance. It can be helpful for many purposes aside from cybersecurity, such as for software developers that usually need to connect to servers using RSA algorithms.

Furthermore, this thesis shows that building a prototype nowadays is just a matter of researching. Thanks to all the steps programming languages have already taken it is not necessary today to manually code all the algorithms. Finding suitable standards and libraries allows one user to build and implement a digital signature and its verification.

CHAPTER 7. LITERATURE

[1] C. P. a. J. Pelzl, Understanding Cryptography, Berlin, 2010.

[2] A. M. P. v. O. a. S. Vanstone, Handbook of Applied Cryptography, 1996.

[3] IBM, «IBM Quantum,» 19 June 2020. [En línea]. Available: https://www.ibm.com/quantum-computing/learn/what-is-quantum-computing/.

[4] C. Friedrich Gauss, Disquisitiones Arithmeticae, Yale University Press, 1965.

[5] David, «Difference Between,» 9 January 2018. [En línea]. Available: http://www.differencebetween.net/technology/protocols-formats/difference-between-rsa-and-dsa/.

[6] S. Bhowmick, «what is the Role of RSA in e-commerce,» *APPSeCONNECT*, 2017.

[7] Ezhimde, «Github,» 11 April 2016. [En línea]. Available: https://github.com/KAsperDeng/putty/blob/master/putty-src/sshrsa.c.

[8] «PHP documentation,» 2016. [En línea]. Available: https://www.php.net/manual/en/function.ssh2-auth-pubkey-file.php.

[9] «FIPS General information,» 09 09 2013. [En línea]. Available: https://www.nist.gov/itl/fips-general-information.

[10] B. M. a. D. J. Patterson, From Barter to Bitcoin: Society, Technology and the Future of Money.

[11] E. G. a. M. Horowitz, Quantum Computing: Progress and Prospects, Washington DC,

2018.

[12] E. Schródinger, Die gegenwärtige Situation in der Quantenmechanik, 1935.

[13] «Official Journal of the European Union,» 20 May 2020. [En línea]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:C:2020:173:TOC.

[14] «Check Lex,» 1 July 2013. [En línea]. Available: https://checklex.publications.europa.eu/?lang=en.

[15] P. M. a. C. A. Ed Simon, An introduction to XML Digitl Signatures, 2001.

[16] S. Behnel, «Lxml Tutorial,» [En línea]. Available: https://lxml.de/tutorial.html.

[17] A. Kislyuk, «Signxml,» 21 June 2020. [En línea]. Available: https://pypi.org/project/signxml/.

[18] A. Sokolovskiy, «Hashlib,» 15 June 2020. [En línea]. Available: https://docs.python.org/3/library/hashlib.html.

CHAPTER 8. SUMMARY

In a digital world, digital signatures have become one of the most important tools in cybersecurity. They are widely used to sign documents, transactions or to set server connections. Thus, the ability to develop digital signature allows the acquisition of several capabilities from the technology that is surrounding us every day.

The aim of this project is to, firstly research and understand concepts, properties, and applications of digital signatures, highlighting RSA and DSA algorithms. Furthermore, this thesis discusses how new technologies such as Quantum computing are becoming a serious threat to current digital signatures' algorithms.

Secondly, with all the knowledge acquired throughout previous investigation, a prototype for verifying digital signatures is build and discussed step-by-step. This prototype is designed for verifying the Official Journal of the European Union by employing python programming language and its different standards and libraries.

Key words:
Cryptography
Cybersecurity
Digital signature
Public-key
Private-key
Quantum Computing
Hash function
RSA
DSA

U digitalnom svijetu digitalni potpisi postali su jedno od najvažnijih oruđa za cyber-sigurnost. Oni se široko koriste za potpisivanje dokumenata, transakcija ili za postavljanje veza s poslužiteljem. Tako sposobnost razvijanja digitalnog potpisa omogućava stjecanje nekoliko mogućnosti iz tehnologije koja nas svakodnevno okružuje.

Cilj ovog projekta je najprije istražiti i razumjeti koncepte, svojstva i primjene digitalnog potpisa, ističući RSA i DSA algoritme. Nadalje, ova teza govori o tome kako nove tehnologije poput kvantnog računanja postaju ozbiljna prijetnja algoritmima digitalnih potpisa.

Drugo, uz sve znanje stečeno tijekom prethodne istrage, prototip za provjeru digitalnog potpisa gradi se i raspravlja se korak po korak. Ovaj je prototip dizajniran za provjeru Službenog lista Europske unije primjenom programskog jezika python i njegovih različitih standarda i knjižnica.

Ključne riječi:

Kriptografija

Cybersecurity

Digitalni potpis

Javni ključ

Privatni ključ

Kvantno računanje

Hash funkcija

RSA

DSA

ANNEX. SUSTAINABLE DEVELOPMENT GOALS

This thesis is also committed with Sustainable Development Goals. The world is evolving towards digitalization where increasingly, instead of using tons of paper, documents, contracts, and all data are digitally stored in servers or clouds. These changes make an enormous impact on the environment and, therefore, play an important role in the SDG.

This digital growth can only be suitably developed by ensuring the same security as analog-world. Inside this field we find cybersecurity. Cybersecurity has several subfields among which Digital Signatures is part.

Hence, Digital Signatures are indirectly a part of SDG, which allow the continuous growth of digitalization towards a world where paper no longer will be used.