information and communication technology

Security in Clouds Environments

Algorithms, Standards, Risks, Procedures and Models

Agnieszka Jakóbik
Jacek Tchórzewski

Kraków 2023

Cracow University of Technology

Security in Clouds Environments

Algorithms, Standards, Risks, Procedures and Models

information and communication technology

Security in Clouds Environments

Algorithms, Standards, Risks, Procedures and Models

Agnieszka Jakóbik
Jacek Tchórzewski

Kraków 2023

This text was published as a part of the project 'Excellence programming – PK XXI 2.0. Cracow University of Technology development program for the years 2018–22'.
Funding from EU: 18,048,774.96 PLN

eISBN 978-83-67188-62-3
Online edition

5 publisher's sheets

Wydawnictwo PK, ul. Skarżyńskiego 1, 31-866 Kraków; 12 628 37 25, fax 12 628 37 60
wydawnictwo@pk.edu.pl
www.wydawnictwo.pk.edu.pl
Correspondence address: ul. Warszawska 24, 31-155 Kraków

For all authors mentioned or cited in this book and for all which were not cited but also have a contribution in improving the security of cyberspace.

For their effort and hard work which allows You, dear Reader, to use modern technologies securely and easily.

CONTENTS

PREFACE

Once upon a time, a bandit called Angulimal threatened to kill Buddha. *Show me your grace Angulimal and fulfil my last wish. Cut the branch from that tree*, says Buddha. Well, Angulimal did it without any problems. Then Buddha said *Now, try to put this branch back to the tree, like it was never cut*. Confused, the bandit said that it is impossible. The cut off branch will never be a part of this tree. Budda concluded *See? You think that you are strong because you can destroy and hurt. It is for kids. Only the great ones know how to create and repair*.

We don't know whether it is a true story or not. However, we agree with the conclusion. In this book, we present modern security solutions dedicated to modern cyberinfrastructures, which is the Cloud. These methods are not easy. They incorporate knowledge from many disciplines, such as mathematics, informatics and even a law. They also assume knowledge about infrastructures, procedures, information flow, etc.

What we tried to do is to present these aspects in a way that non-experts will gain knowledge in this field, and maybe even experts will find something new which can be inspiring for future works.

Agnieszka Jakóbik, Jacek Tchórzewski

ACRONYMS

AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Interface
APT	Advanced Persistent Threat
CA	Certification Authority
CBC	Cipher Block Chaining
CC	Cloud Computing/Computational Cloud
CERT	Computer Emergency Response Team
CFB	Cipher Feedback
CP	Cloud Provider
CSA	Cloud Security Alliance
CTR	Counter Mode
CVSS	Common Vulnerability Scoring System
DAs	Defensible Actions
DDoS	Distributed Denial of Service
DES	Data Encryption Standard
DoS	Denial of Service
DS	Digital Signature
DSS	Digital Signature Standard
EC	Elliptic Curve
ECB	Electronic Codebook
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ENISA	European Union Agency for Cybersecurity
FIPS	Federal Information Processing Standards
IaaS	Infrastructure as a Service
IEC	International Electrotechnical Commission
IoT	Internet of Things
ISO	International Organization for Standardization
MitM	Man in the Middle
NIAC	National Infrastructure Advisory Council
NIST	National Institute of Standards and Technology

OFB	Output Feedback
OS	Operating System
OWASP	Open Web Application Security Project
PaaS	Platform as a Service
PKCS	Public Key Cryptography Standard
PN	Petri Net
RSA	Rivest-Shamir-Adleman
SaaS	Software as a Service
SE	Searchable Encryption
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SLA	Service Level Agreement
SP	Storage Place
VLAN	Virtual Local Area Network
VM	Virtual Machine

NOTATION

In this section, we will present the notation which is common for the whole document. Note that in each chapter or subchapter, local variables and functions necessary for the presentation of algorithms are also defined and are not listed here.

$\wedge$	AND operation.
$\vert$	OR operation.
$not(a)$ or $\neg a$	Negation of a.
$\vert\vert$	Concatenation.
$\forall a$	For all arguments denoted by a.
$\exists a$	There exists element a.
$\times$	Matrix multiplication.
$\oplus$	Bitwise logical exclusive-or on bit strings of the same length.
$\lceil a \rceil$	The ceiling of the argument: the smallest integer that is greater than or equal to a.
$\lfloor a \rfloor$	The floor of the argument; the largest integer that is less than or equal to a.
$\{0, 1\}^n$	Binary string of length n. If $n = *$ length is not limited.
$a.append(b)$	Appends element b to the vector a. If b is also a vector, all elements of b are added to the vector a, and the vector a is containing $len(a) + len(b)$ elements.
$a \pmod n$	The unique remainder when integer a is divided by the positive integer n.
$bitlen(a)$	Number of bits of element a.
$bytearray(a, b)$	Function which converts element a into byte array of size b. Unused bytes are set to $0x00$.
F_p	Field defined over value p.
F_{2^m}	Field defined over binary number 2^m.
$f : a \to b$	Function f assigning b values to all a values.
ϕ	Eulers' totient function.
GF	Galois Fied.
$gcd(a, b)$	Greatest common divisor of the integers a and b.

$H(a)$	Hash produced from message a.
$LSB_s(w)$	s least significant bits of binary word w.
$lcm(a, b)$	The least common multiple of the integers a and b.
$len(a)$	Number of elements in vector a.
$MSB_s(w)$	s most significant bits of binary word w.
$P - a$	EC defined over a bits prime number.
$parseUint32(a)$	Function which is parsing byte array a into unsigned 32 bits integer. Note, that $len(a) = 4$.
$parseUint64(a)$	Function which is parsing byte array a into unsigned 64 bits integer. Note, that $len(a) = 8$.
$RSA - a$	RSA algorithm with key length equal to a.
$RotateLeft(w, n)$	Left binary rotation of binary word w by n positions.
$RotateRight(w, n)$	Right binary rotation of binary word w by n positions.
$SHA - x$	SHA function returning x bits hash.
$ShiftLeft(w, n)$	Left binary shift of binary word w by n positions.
$ShiftRight(w, n)$	Right binary shift of binary word w by n positions.
$seedlen$	The length of the *domain parameter seed* in bits.

1. CLOUD AND SECURITY

1.1. CLOUD COMPUTING

The definition of Cloud was introduced in 2011 by the National Institute of Standards and Technology: *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models* [80].

The essential parameters of Clouds are [80]:

- On-demand self-service – which means that the process of providing computational capabilities is fully automated. Human intervention is no needed at this point [80].
- Broad network access – which means that the Cloud can be used and accessed by any device which supports standard mechanisms promoted by heterogeneous platforms. For example, IoT devices such as mobile phones, as well as standard PCs or super-fast workstations [80].
- Resource pooling – which means that computational ability resources are distributed between many possible customers. The resources can be assigned, reassigned or deleted dynamically; however, the client doesn't physically know where these resources are [80].
- Rapid elasticity – which means that computational capabilities can scale dynamically proportionally to the customers' demands. From the customers' point of view, this scaling should be invisible and unnoticeable [80].
- Measured service – usage of available resources should be constantly optimised and controlled. Appropriate measures have to be introduced to provide clarity for a Cloud Provider and customers [80].

According to [80] and [3], we can distinguish three basic Cloud services models:

- Software as a Service (SaaS) – services are running on Cloud infrastructures and are provided to customers via thin client platforms. These platforms could be a browser, application, e-mail, etc. The important thing is that the

installation of an application is not needed [80]. Responsibility for data security is on the Cloud Provider side. A good example of SaaS is Google Apps.
- Platform as a Service (PaaS) – services can be built and deployed in the Cloud by customers. Customers can use tools supported by the Cloud Provider. Data security responsibility is shared. Infrastructure and underlying layers are controlled by the Cloud Provider. However, applications themselves and their configurations are controlled by the customers. From the developers point of view, usage of these types of services can increase data availability, services scalability and the speed of developing and testing of applications [80].
- Infrastructure as a Service (IaaS) – availability, monitoring and management of whole data centres. The ideology is similar to PaaS. However, customers get more control over basic computational resources, such as firewalls. Any customer application is fully controlled by the customer (including operating systems) and can be arbitrarily deployed. Security responsibility is still shared. The Cloud Provider is responsible only for the underlying Cloud infrastructure. The rest, such as storage or chosen networking components, is the customers' responsibility [80]. As an example of IaaS infrastructure, we can consider AWS (Amazon Web Services).

Deployment – all processes (including testing, running, maintenance) which are needed for software and hardware to exist and work properly.

Deployment model – a systematised and structured way of appropriate deployment. NIST in [80] defines four basic Cloud deployment models:
- Private Cloud – multiple users are allowed. The management is carried out by a single organisation. Infrastructure can be leased, and maintenance can be carried out by a third party [80].
- Community Cloud – multiple users are allowed, and management can be carried out by a single organisation or by multiple organisations. However, consumers (users) create a community and use the Cloud for shared concerns [80].
- Public Cloud – can be managed by any organisation. This type of Cloud is publicly available [80].
- Hybrid Cloud – a combination of two or more of the types mentioned above. The hybrid Cloud consists of separate types which bounded together create one standardised conglomerate [80].

We can point out here that all definitions given above are not the only ones, and not the only proper ones. We based these on the NIST definitions, because NIST is one of the biggest organisation for standardisation of security measures.

To ensure secure Cloud Computing, we have to deeply analyse how Cloud infrastructure looks like. To do this, the authors in [3] divided the Cloud into layers, where each layer represents resources which share similar features:

- Horizontal layer – defines relations between resources. It can be divided into three sublayers [3]:
  - Physical sublayer – defines policies of communication between users and their resources. This sublayer consists of network devices, operating systems and their firmware [3].
  - Virtual sublayer – is designed for user resource management without direct usage of the physical layer. This layer consists of virtual machines and virtual discs which gather physical resources in fully functional logical parts [3].
  - Application sublayer – represents all applications and software which use virtual layer resources or directly use physical resources [3].
- Vertical layer – defines tasks which will be realised by particular resources. This can also be divided into three sublayers [3]:
  - Storage sublayer – which represents a database or set of databases used for information storage [3].
  - Server sublayer – which defines the role of a server working in the Cloud infrastructure [3].
  - Network sublayer – which defines the role of network devices working in the Cloud infrastructure [3].

As can be seen, assuring security in the Cloud is very demanding. All layers with all sublayers have to be considered and secured properly, i.e. effectively and according to the law.

1.2. CRYPTOGRAPHY TECHNIQUES

1.2.1. INTRODUCTION

Assuring security in Computational Clouds is very demanding. In this chapter, we will describe methods which allow one to ensure secure commutation and secure data storage. In chapter 1.2.2, we introduce basic security definitions. In the next chapters, we describe cryptographic solutions to assure an appropriate security level.

1.2.2. BASIC DEFINITIONS

Cryptology – is a branch of science concerning methods of secure information exchange and storage. Cryptology can be divided into cryptography and cryptanalysis.

Cryptography – creating and investigating ciphers, transformations and other securing methods. The idea is to represent a message in a form that only legitimate user can reveal.

Cryptanalysis – part of cryptology oriented towards breaking secure algorithms, transformations and other securing methods. The idea is to find breaches and weaknesses which allow an unauthorised user to reveal (wholly or partially) a secured message.

Steganography – a branch of science oriented towards hiding information. The idea is not to change the message content the but form. An attacker shouldn't know that the message exists. Steganography should be used with cryptography, which means that a hidden message should also be ciphered.

Steganalysis – a branch of science oriented towards detecting messages. Steganalysis algorithms will answer whether a message is present in investigated data and where. Each Cloud user needs to be authenticated and authorised.

Authentication – a method which allows one to confirm the identity of a user. Potential users are physical persons, virtual machines, services, applications, etc.

Authorisation – a method which allows one to verify whether a user can get access to particular resources or not. Authorisation is done after authentication.

Cryptogram – is a presentation of the message in a form where only authorised recipients can reveal the content.

Cipher – algorithm which creates appropriate, secure cryptograms.

Padding – adding to the original message bits to reach the appropriate length of the message. There are several padding methods. One of the simplest is pad10*. At the end of the message, we add '1' bit and then fill the rest with '0' to reach the appropriate length.

Strong Prime – accordingly to [110], this is a prime number p which fulfils the following conditions:

- p is large,
- $p - 1$ has a large prime factor, denoted by r,
- $p + 1$ has a large prime factor,
- $r - 1$ has a large prime factor.

Strong Primes can improve the security of many cryptographic algorithms, though not all of them. For example, it is not necessary to use Strong Primes in RSA cryptosystem [110].

Safe Prime – should be used mainly for cryptosystems based on a discreet logarithm problem (DLP). Safe Primes are numbers which fulfil the condition presented in eq. (1.1) [109]

$$p = 2q + 1 \tag{1.1}$$

where q is also prime.

In this case, q is also called a Sophie Germain Prime [109].

Discrete Logarithm Problem – the authors in [43] defined DLP as follows: *Let G be a cyclic group of order n and g be a generator for G. Given an element $y \in G$, the discrete logarithm problem is to find an integer x such that*:

$$g^x = y \tag{1.2}$$

Secret Sharing – spreading information (secret) to multiple users in a way that only a given number of participants (whole group or a subgroup) can retrieve it.

1.2.3. SYMMETRIC CIPHERS

In the past, two popular and classical cryptography techniques were:

- Substantive Encryption – replacing message signs with a chosen cryptogram (e.g. Cesar Cipher, Playfair Cipher, Hill Cipher).
- Changeover Encryption – appropriate sign mixing.

Both types are insecure against simple attacks. Modern society needed a fast and secure encryption method. That's where symmetric ciphers came in. The main concept is presented in fig. 1.1. The message is passed to the symmetric algorithm, which uses a secret key to create the cryptogram. The cryptogram is passed to the recipient. The recipient uses a reverse algorithm and the same secret key to retrieve a message from the cryptogram [108]. However, there are some basic requirements which need to be fulfilled to reach the appropriate security level [108]:

- Attacker cannot break the algorithm even if some amount of cryptograms were captured (e.g. during communication).
- An attacker cannot restore a secret key even if several messages were captured with corresponding cryptograms.
- There is one key for ciphering and deciphering. Thus, both sides of communication have to keep this key in secret. The key must also be forwarded from one side to the other in a secure way (e.g. via a dedicated secure communication channel).

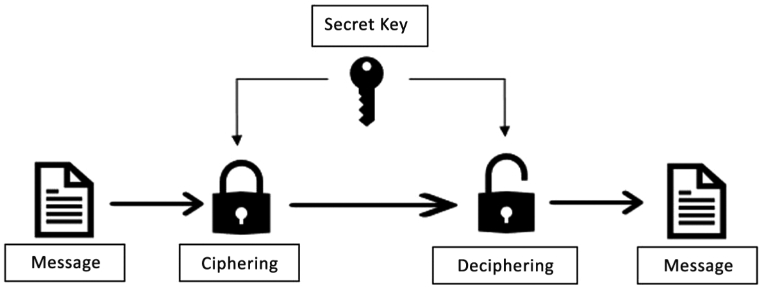


Fig. 1.1. Symmetric Ciphering Scheme [108]

Currently, there are two types with symmetric ciphers:

- Stream Ciphers – the key is at least as long as the message. Ciphering runs bit after bit [72] and depends on adding bits of the message to the bits of the key. The key cannot be reused. Stream Ciphers are faster than Block Ciphers [72]. This is caused by their simplicity. The biggest problems are in generating the pseudorandom secure key and with the key size. One of the most popular stream ciphers is the Vernam Cipher.

- Block Ciphers – the original message is divided into separate blocks (most popular block sizes for symmetric ciphers are 64 bits, 128 bits and 256 bits). Both sides of communication have the same secret key. With the usage of this secret key, blocks of cryptograms are created. Each block has the same size as one block of the message [108]. If the message length is not equal to the multiplication of block size (further denoted by b), an appropriate padding method has to be used. The ciphering scheme is presented in [108] and in fig. 1.2. The Block Ciphering Scheme is the most popular technique used in network communication [108].

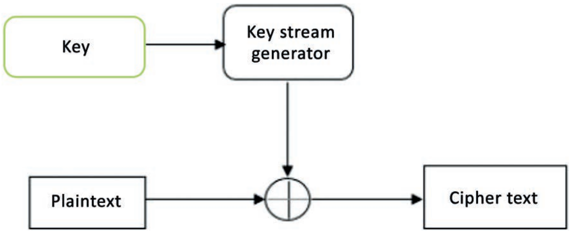


Fig. 1.2. Stream Cipher Encryption scheme presented in [72]

NIST defined five operational modes which can be applied in block ciphering [32]:

– **Electronic Codebook (ECB)** – the ciphering function is used independently and directly on all message blocks [32]. The ciphering process is presented in eq. (1.3) and deciphering in eq. (1.4) [32].

$$C_i = CIPH_k(M_i), \quad i = 1, \dots, n \tag{1.3}$$

$$M_i = CIPH_k^{-1}(C_i), \quad i = 1, \dots, n \tag{1.4}$$

where M_i is one block of the message, C_i one block of the cryptogram, n total number of blocks, $CIPH_k$ ciphering function which uses the secret key k and $CIPH_k^{-1}$ deciphering function which uses the secret key k [32].

– **Cipher Block Chaining (CBC)** – first block of the message is XORed with a secure initialization vector V before ciphering (see eq. (1.5)) [32]:

$$C_1 = CIPH_k(M_1 \oplus V) \tag{1.5}$$

The size of V is equal to the block size and must be unpredictable (however, it is not necessary to keep it in secret) [32]. The rest of the blocks are then ciphered according to eq.(1.6). Each output block of the cryptogram depends on the key and the previous block [32]:

$$C_i = CIPH_k(M_i \oplus C_{i-1}), \quad i = 2, \dots, n \tag{1.6}$$

The decryption process is a reverse operation (see eq. (1.7) and (1.8)) [32]. However, the first block needs to be deciphered before it is XORed with V:

$$M_1 = CIPH_k^{-1}(C_1) \oplus V \tag{1.7}$$

And the rest of the blocks:

$$M_i = CIPH_k^{-1}(C_i) \oplus C_{i-1}, \quad i = 2, \ldots, n \tag{1.8}$$

- **Cipher Feedback (CFB)** – is slightly different than its predecessors. One parameter is block size b determining the size of blocks on which the cipher will be operating. The second one is called segment size s, and it determines the size of cryptogram blocks and message blocks. s must be greater than 0 and smaller than the block size b. The most common s values are: 1 bit, 8 bits, 64 bits and 128 bits [32]. NIST also introduces input I of size b, which is an input block which will be processed by the ciphering algorithm and output O of size b, which is the output returned by the algorithm. The ciphering starts with passing vector V (the same vector as in CBC) as an input to the algorithm [32]:

  $$I_1 = V \tag{1.9}$$

  considering n segments which have to be ciphered, the next inputs are given as a concatenation of $(b - s)$ Less Significant Bits of previous input I and the previous cryptogram segment C (note that C is the size of s) [32]:

  $$I_i = LSB_{b-s}(I_{i-1}) \,|\, C_{i-1}, \quad i = 2, \ldots, n \tag{1.10}$$

  The outputs of the algorithm are just ciphered inputs. However, these outputs do not create a cryptogram yet [32]:

  $$O_i = CIPH_k(I_i), \quad i = 1, \ldots, n \tag{1.11}$$

  To create the final cryptogram, an XOR operation is performed on the s Most Significant Bits of outputs and corresponding message segments [32]:

  $$C_i = MSB_s(O_i) \oplus M_i, \quad i = 1, \ldots, n \tag{1.12}$$

  Deciphering is the reverse operation. This starts with assigning vector V as in eq. (1.9). The inputs are then calculated as in eq. (1.10) and the outputs as in eq. (1.11) The final deciphering process is based on the fact that the XOR operation is reversible [32]:

  $$M_i = MSB_s(O_i) \oplus C_i, \quad i = 1, \ldots, n \tag{1.13}$$

- **Output Feedback (OFB)** – assuming the notation is the same as in CFB mode, initialisation vector V is assign to the input I_1:

$$I_1 = V \tag{1.14}$$

However, vector V must be unique at each execution of the algorithm, even if the algorithm is using the same secret key [32]. Note that in this example, we are considering block size only; segment size does not exist in the given context. This method does not require padding. It implicates the fact that all cryptogram blocks are the size of b, and the last one can be smaller. The next inputs become outputs (results from symmetric algorithm) from the previous step (see eq. (1.15)) [32].

$$I_i = O_{i-1}, \quad i = 2, \ldots, n \tag{1.15}$$

The outputs are ciphered inputs (see eq. (1.16)) [32]:

$$O_i = CIPH_k(I_i), \quad i = 1, \ldots, n \tag{1.16}$$

The outputs are still not the final cryptogram. All blocks of the cryptogram except the last one are created as presented in eq. (1.17) [32].

$$C_i = O_i \oplus M_i, \quad i = 1, \ldots, n-1 \tag{1.17}$$

Considering that the last block of the message is the size of u ($0 < u < b$), the last block of the cryptogram is created by an XOR operation done on the last block of the message and u Most Significant Bits of the last output (see eq. (1.18)) [32].

$$C_n = MSB_u(O_n) \oplus M_n \tag{1.18}$$

During decryption, we perform the same operations as presented in eq. (1.14), eq. (1.15) and eq. (1.16) [32]. The $n - 1$ blocks of the original message are then decrypted as presented in eq. (1.19) and the last one as presented in eq. (1.20) [32].

$$M_i = O_i \oplus C_i, \quad i = 1, \ldots, n-1 \tag{1.19}$$

$$M_n = MSB_u(O_n) \oplus C_n \tag{1.20}$$

- **Counter Mode (CTR)** – in this mode, all blocks of messages have their own initial vector of size b (block size). These vectors are called counters, and for a single n blocks, the message will be denoted by T_1, T_2, …, T_n [32]. Those counters have to be unique under the given secret key. This means that even if we use one secret key for ciphering many messages, all counters for all messages have to be different [32]. Note that this model also does not demand padding methods. Considering that the rest of the notation remains the same as in the OFB example, outputs are created as presented in eq. (1.21) [32].

$$O_i = CIPH_k(T_i), \quad i = 1, \ldots, n \tag{1.21}$$

All blocks of cryptogram are outputs XORed with message blocks (see eq. (1.22) and eq. (1.23)) [32]. In this case, the last block of the message is also the size of u, where $0 < u < b$, and we XOR this block with u Most Significant Bits of the last output block.

$$C_i = O_i \oplus M_i, \quad i = 1, \ldots, n-1 \tag{1.22}$$

$$C_n = MSB_u(O_n) \oplus M_n \tag{1.23}$$

For decryption purposes, we do the same operation as presented in eq. (1.21), and we once again use the fact that the XOR operation is reversible (see eq. (1.24) and eq. (1.25)) [32].

$$M_i = O_i \oplus C_i, \quad i = 1, \ldots, n-1 \tag{1.24}$$

$$M_n = MSB_u(O_n) \oplus C_n \tag{1.25}$$

Block ciphers are slower than stream ciphers; however, they can offer high security with much smaller and predictable key sizes. One of the most popular symmetric block ciphers is DES, as well as the more modern AES.

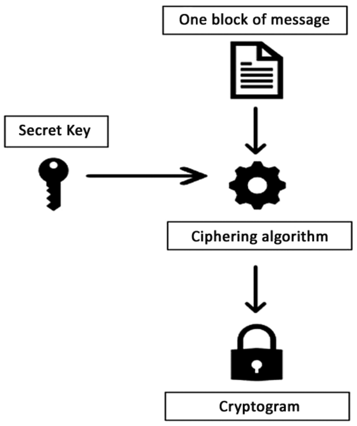


Fig. 1.3. Block Cipher Encryption scheme presented in [108]

1.2.4. ASYMMETRIC CIPHERS

The asymmetric ciphering scheme is presented in fig. 1.4. As we can see, there is no longer only one key for ciphering and deciphering purposes. The first key, which is called the Public Key, is used to transform the message into a cryptogram. This key is not kept in secret, which is why it is called 'public'. The Private Key is used for deciphering purposes, and it is kept in secret [108]. Two important parameters of asymmetric ciphers are [108]:

- It should be hard to retrieve the Private Key if the Public Key and ciphering algorithm are known publicly.

- In some algorithms (such as RSA), both generated keys can be used for ciphering purposes. The second key is then used for deciphering purposes and is kept in secret.

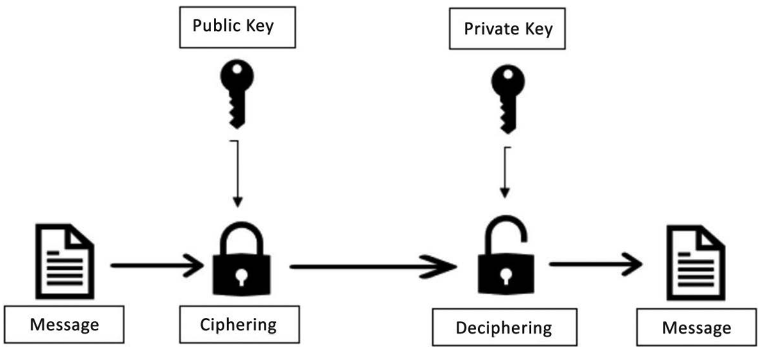


Fig. 1.4. Asymmetric Ciphering Scheme presented in [108]

Asymmetric ciphers are slower than symmetric ciphers [67]. However, they allow one to avoid problems with key distribution, because the Public Key can be widely known, and the Private Key is never shared. Most asymmetric ciphers are also block ciphers. This means that for a given algorithm, there exists a maximal message size, and if a message exceeds this size, it has to be divided into 2 or more blocks. Note that the operational modes for block ciphers mentioned in sec. 1.2.3 are dedicated only for symmetric ciphers.

There are four basic situations where asymmetric ciphers should be used [108]:

- When the sender wants to encrypt the message and the cryptogram should be very strong.
- When the sender wants to sign a message with a digital signature.
- When the sender wants to share a session key. For example, communication will be continued with a symmetric encryption, and a symmetric key is distributed with asymmetric cryptography usage.
- When strong authorisation or confidentiality is necessary.

1.2.5. HASHING FUNCTIONS

Hashing functions belongs to the One Way function family (see eq. (1.26)) [113].

$$\forall x \in X, \quad f : x \to y \wedge \neg(\exists g : y \to x) \tag{1.26}$$

This means that for any input *x*, we can calculate output *y*; however, the reverse operation (calculating value *x* from *y*) is impossible. One way function can be a hashing function if it fulfils the condition presented in eq. (1.27) [113].

$$h:\{0,1\}^* \rightarrow \{0,1\}^*, \quad n \geq 1 \tag{1.27}$$

This equation means that input and output are considered as binary strings. Input can be any length, but the output is always a fixed length. The result of the hashing function is called hash or digest, and it is a sort of fingerprint of a given data set (input binary string). The main structure is presented in fig. 1.5.

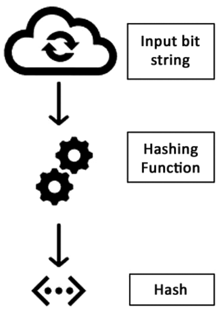


Fig. 1.5. Hashing Function structure described in [113]

Three basic hash function security parameters are [113]:

- Collision Resistance – it is hard to find two different messages which produce the same hash value.
- First Preimage Resistance – for a given hash, it is hard to find a message which produces this hash.
- Second Preimage Resistance – for a given message and its hash, it is hard to find the second message which produces the same hash value. The main difference between Second Preimage Resistance and a Collision Resistance is that the first message is given.

Note that:

- There is no ideal Hashing Function. All hashing functions can be compromised, because there are potentially infinite inputs and a finite number of outputs.
- There are more parameters which have to be fulfilled by the hashing function. The three mentioned above are basic.

The security of hashing functions is measured in bits of security, where n bits of security means that the attacker must perform 2^n operations to break a hash [113]. Detailed information about the security of a standardised hashing function can be found in [33].

There are many hashing functions; however, only four were considered as SHS (Secure Hash Standard) and were accepted by NIST [33], [90]: Secure Hash Algorithm 0 (SHA-0), SHA-1, SHA-2 and SHA-3. SHA-0 and SHA-1 have been compromised and are not considered as secure anymore. SHA-2 and SHA-3 are not hashing functions; they are families of hashing functions. The family of SHA-2 contains the following hashing algorithms [90]: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256, where the number after 'SHA' indicates hash length in bits. SHA-512/256 means that the final hash length is equal to 512 bits; however, it can be securely truncated into 256 bits [90]. The family of SHA-3 contains the following hashing algorithms [90]: SHA3-224, SHA3-256, SHA3-384 and SHA3-512. In fact, some of these algorithms have a constraint on maximal message length which can be hashed. The hashing function can be used: to create digital signatures, to create hashing tables (indexing), for secure password storage in databases and so on.

1.2.6. DIGITAL SIGNATURES

To create a digital signature, both sides of communication need to establish (or just choose) an appropriate hashing function (see sec. 1.2.5). This function must be the same on both sides of the communication. The signer must also establish an appropriate public and private key (see sec. 1.2.4) and deliver this public key to the verifier. The process of creating the digital signature is presented in fig. 1.6 [108].

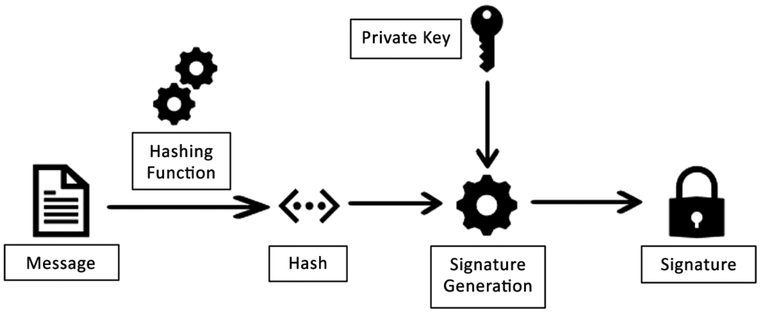


Fig. 1.6. Digital Signature creation presented in [108]

In the first step, the message is hashed. The digital signature is then created with the use of the sender's private key. Note that only the sender can create the signature, because only the sender knows his private key.

The verification process is presented in fig. 1.7 [108]. The verifier gets a message and a signature of this message. Firstly, the hash from the message is calculated by using the same hashing function as the sender. Signature verification is a process which demands a signature, hash from the original message and the sender's public key. The algorithm returns information on whether verification completed successfully or not. Note that everyone who owns the sender's public key can verify the signature. There are four reasons to use digital signatures schemes [67]:

- Authentication – digital signatures allow one to verify a machine's or person's identity [67]. After confirming the identity, authorisation can be done.
- Privacy – only authorised persons or machines can operate on given data [67].
- Integrity – digital signatures themselves do not protect data but allow detect any unauthorised changes [67].
- Non-Repudiation – the sender cannot deny the fact that the message was sent by him [67].

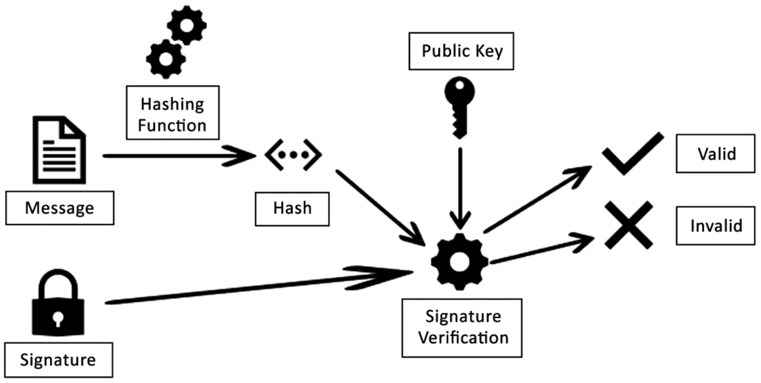


Fig. 1.7. Digital Signature verification presented in [108]

1.3. SECURITY SERVICES FOR CLOUD DEFENCE

Fundamental cryptographic tools that are used in Computational Clouds are symmetric and have public-key encryption, hash functions, message authentication codes and digital signatures (see sec. 1.2). Usually in CC systems, data is stored in different locations than in the units which are processing them (VMs), for example in dedicated virtual disks or databases (Storage Place – SP). Therefore, the results of computations have to be transmitted into the Cloud to be retrieved by the user who uploaded the tasks. Considering data in motion and data at rest, different types of defences have to be used. In this chapter, we will describe what the security solutions are for Cloud environments and how to implement them.

1.3.1. CRYPTOGRAPHIC SOLUTIONS FOR CLOUD ENVIRONMENTS

Several limitations of conventional cryptography have to be considered during the transfer of algorithms and protocols into a Cloud environment [124]:

- Using traditional methods implies the inability of processing of encrypted data. Decrypting data before processing is a very inefficient solution.

- Point to Point Data Access Policies, when the sender knows the intended recipient. Irrelevant when the group of recipients is the target of the data or the recipient is unknown. Also inadequate for shared and dispersed storage.
- Limited trust. All traditional cryptographic protocols remain secure under the assumption that parties can be trusted. Trust in a Cloud environment can be assured only by SLA or by a contract between the customer and provider. Compromise of the private key by an uneducated customer or intentionally returning incomplete results by a provider may collapse the whole chain of trust.
- Requests for encrypted data are made over public channels. The provider does not control customers' endpoints: web browser, mobile phone or tablet.

These are reasons for new cryptographic mechanism development that can reduce the necessary level of trust between a customer and the holder of encrypted data:

- Searching Over Encrypted Data: Conventional ciphertexts are designed not to reveal any information about the original plaintext and are not suitable for searching without decryption. The proposed solutions are Searchable Encryption (SE) schemes. Most algorithms add keywords into encrypted material or allow one to use chosen Boolean formulas to create queries.
- Homomorphic Encryption: Traditional encryption schemes exclude meaningful combinations which can be done on ciphertexts such as lowering the security of the procedures by making them more vulnerable to attacks. Considering some degree of security reduction in favour of computational efficiency, Cloud providers may use homomorphic encryption schemes. This enables chosen computations to be done on encrypted data.
- Aggregating Over Encrypted Data: Certain types and sets of data shared by the same community of users may be aggregated and processed using one of the secret sharing techniques [27, 52, 53]: Shamir secret sharing scheme, Ramp Shamir secret sharing scheme [19], additive secret sharing scheme or replicated additive secret sharing scheme.

Invisible Party or Content Cryptography: Anonymous and blind cryptography is used when one party of the scheme does not want to reveal his identity or the verifying procedure should be done without encrypting the underlying message. Blinded signatures are schemes in which the content of a message is blinded before the message is signed [25]. This may be applied in cryptographic election systems and digital cash schemes. Anonymous cryptography is used to verify a legitimate signer in four different ways:

- authorised entity may identify the signer of a signature,
- authorised entity may link two signatures created by the same signer without identifying the signer,
- both of the authorised entities may identify or link,
- neither of the authorised entities may identify or link.

Authentication of an anonymous party is an example of anonymous digital signatures [23, 118].

1.3.2. IMPLEMENTATION OF CRYPTOGRAPHY IN CLOUDS

Detailed instructions and best practices as far as the implementation of particular algorithms is considered can be found in [28], in NIST SPs and guidelines and in ISO standards [16]. Effective implementation of multiple-precision integers arithmetic, modular arithmetic, greatest common divisor algorithms and exponentiation is crucial for the effectiveness of cryptography algorithms [65]. Furthermore, several design objectives have to be considered [79]:

1. The customer should not need an additional third party to encrypt data on his side.
2. Sending data to the Cloud and reading it from the Cloud should be done with an encryption framework.
3. The customer must be authorised to have access to the data stored in the Cloud.
4. Cryptographic keys must be generated instantly, validated and should never be located in the Cloud storage centre.
5. The provider is obligated to provide customers with free choice of cryptographic algorithms.
6. The provider must develop the efficient mechanism of encryption over the Cloud. It includes scalability supporting, automatisation of processes and 24/7 services availability despite geographic dispersion of customers.

NIST offers the Cryptographic Module Validation Program that verifies cryptographic modules according to the Federal Information Processing Standards (FIPS), such as FIPS 140-1 Security Requirements for Cryptographic Modules. An additional advantage of this procedure is that NIST publishes validated (according to FIPS 140-1 and FIPS 140-2) Cryptographic Modules lists.

The following libraries, among many others, were certified and may be used for high-level cryptographic services providing:

- Security Builder FIPS Java Module: supports optimised Elliptic Curve Cryptography and provides Java application developers with sophisticated cryptographic tools.
- Dell OpenSSL Cryptographic Library: for various Dell Networking products.
- The Luna PCI-e cryptographic module: a multi-chip embedded hardware cryptographic module in the form of a PCI-Express card.
- SUSE Linux Enterprise Server 12 – StrongSwan Cryptographic Module: a complete IPsec implementation for the Linux kernel.
- SUSE Linux Enterprise Server 12 – OpenSSH Client Module, SUSE Linux Enterprise Server 12 – OpenSSH Server Module.
- wolfCrypt module, a comprehensive suite of FIPS approved algorithms.

- The IBM Crypto for C v8.4.0.0 (ICC), cryptographic module implemented in the C programming language. It is packaged as dynamic (shared) libraries usable by applications written in a language that supports C language linking conventions (e.g. C, C++, Java, Assembler, etc.).

Other tools are delivered by Cloud providers themselves for usage in their Clouds. Such examples are:

- AWS Encryption SDK: dedicated to defining a system of keys which customers use to encrypt data. It supports tracking and protecting the data encryption keys and performs low-level cryptographic operations, see fig. 1.8.
- IBM Cloud Data Encryption Services: includes a combination of the AES--256-certified encryption, hashed integrity checking, multifactor secret sharing with keyed information dispersal and internal bulk key management [47].
- Openstack Barbican: symmetric key management system dedicated to SSL/TLS keys, SSH keys or any other type of key material. Additionally, it supports public Certificate Authorities (CAs) [85].
- HP CLOUD: provides advanced data encryption, tokenisation, and key management that protects sensitive data across enterprise applications, data processing IT, Cloud, payment ecosystems, critical transactions, data storage and big data platforms.
- Adobe Cloud provides the Adobe Secure Product Lifecycle: which specifies software development practices, processes and tools.

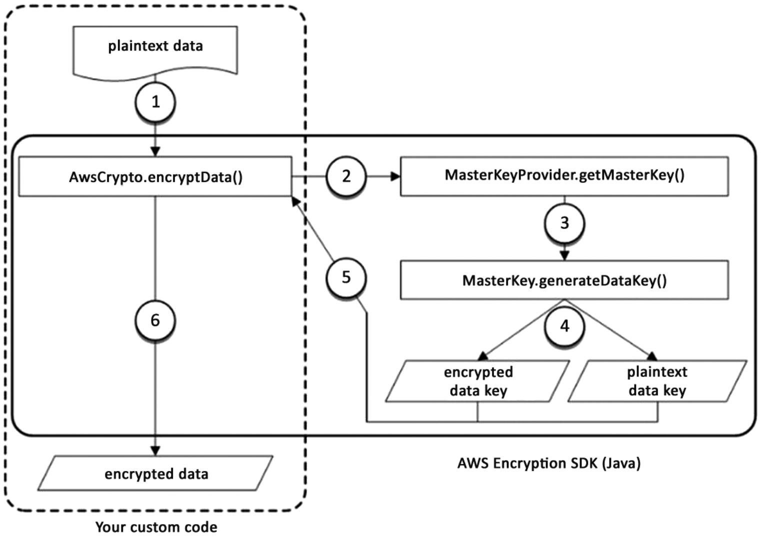


Fig. 1.8. AWS Encryption SDK encryption scheme presented in [11]

2. PRACTICAL EXAMPLES OF SECURITY TECHNIQUES

In this chapter, we present security algorithms and solutions which are currently considered as secure. All functions, algorithms and computations assume Big--Endian byte order, unsigned integer usage (size depends on the algorithm) and standard wrapping behaviour of unsigned integer numbers. All arrays and tables are indexed from 0.

2.1. ADVANCED ENCRYPTION STANDARD

The Advanced Encryption Standard (AES) is a symmetric block cipher announced by the National Institute of Standards and Technology in 2001 [34]. It can be used as a *CIPHER* algorithm presented in the symmetric block cipher modes in sec. 1.2.3. Operations are made on 128-bit blocks, and possible key sizes are 128, 192 or 256 bits [34]. AES is considered as a secure standard which can be used for network communication. Before presenting the algorithm itself, we introduce appropriate notations and information in sec. 2.1.2.

2.1.1. AES IN CLOUDS

AES is one of the most efficient symmetric algorithms [73]. Thus, the authors in [98] claimed that the Advanced Encryption Standard is very suitable for Cloud environments. The authors point out that [98]:

- AES can perform efficiently on software platforms, as well as on hardware platforms (including 8-bit and 64-bit platforms).
- Using AES results in very high performance in software usage, because of ease of parallelisation of computations and parallelisation of instructions.
- AES is suitable for restricted-space environments due to less memory allocation than other ciphering schemes.
- No proven cryptoanalysis attacks on AES cryptosystem have been made. What's more, AES potentially supports any key and block size greater than 128 bits.

The authors in [73] performed tests on 128-bit AES used under a simulated Cloud environment. Their conclusion was: *The performance evaluation shows that AES cryptography can be used for data security*.

The authors in [13] present complex studies on how AES can be used for secure data storage in Cloud environments. They investigated security, speed and the delay which appears when AES is used. They conclude that the use of AES allows one to increase confidentiality, authenticity and access control. The only drawback was that encrypting larger files increased the delay.

AES is also used in commercial Clouds:

- Google: *Data stored in Google Cloud Platform is encrypted at the storage level using either AES256 or AES128* [42].
- Amazon Web Services: *The AWS Key Management Service uses the Advanced Encryption Standard (AES) algorithm in Galois/Counter Mode (GCM) with 256-bit secret keys* [12].
- Microsoft OneDrive: *While BitLocker encrypts all data on a disk, per-file encryption goes even further by including a unique encryption key for each file. Furthermore, every update to every file is encrypted using its own encryption key. Before they're stored, the keys to the encrypted content are stored in a physically separate location from the content. Every step of this encryption uses Advanced Encryption Standard (AES) with 256-bit keys...* [82].

2.1.2. AES NOTATION AND BASIC INFORMATION

- M – is a message which will be ciphered, and $len(M) \pmod{16} = 0$. If the message length does not fulfil this equation, appropriate padding methods have to be used.
- State – 4×4 matrix (see tab. 2.1), which is used for ciphering operations [34]. Firstly, it contains a message than ciphering operations can be done. $s_{i,j}$ denotes bytes, and thus $4 * 4B = 16B = 16 * 8b = 128b$. As we mentioned earlier, in the first step, the message is put into a State array. This is done by taking byte after byte of the message and putting them into columns.
  Example: let's consider a message given in a hex form:
  $M = 0xa1b1c1d1e1f1a2b2c2d2e2f2a3b3c3d3$,

Table 2.1

AES State array

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

The State will then look as presented in tab. 2.2. Note that, technically, each row is a 32-bit word (it is also a classic integer size). It implicates the fact that State rows can be interpreted as integer numbers which can speed up computations. In such a case, State contains 4 elements, and in our example, $S(0) = 0xa1e1c2a3$, etc.

Table 2.2

AES State array example

0xa1	0xe1	0xc2	0xa3
0xb1	0xf1	0xd2	0xb3
0xc1	0xa2	0xe2	0xc3
0xd1	0xb2	0xf2	0xd3

- *Nb* – is equal to 4, because it is the number of 32-bits word stored in a State array [34].
- *Nk* – as we mentioned earlier, possible key sizes are 128, 192 or 256 bits. However, the key is also stored as 32-bit words. Thus, appropriate *Nk* values are: 4, 6 or 8 [34].
- *K* – ciphering and deciphering key. Contains 32-bit words.
- *Nr* – represents the number of rounds. When *Nk* = 4, *Nr* = 10. When *Nk* = 6, *Nr* = 12, and when *Nk* = 8, *Nr* = 14 [34].

For ciphering and deciphering purposes, we also have to define 5 tables containing constant values: Sbox (see tab. 2.3), InvSbox (see tab. 2.7), MixingVars (see tab. 2.5) and InvMixingVars (see tab. 2.6) [34].

Table 2.3

AES Sbox Constants

0x63	0x7C	0x77	0x7B	0xF2	0x6B	0x6F	0xC5	0x30	0x01	0x67	0x2B	0xFE	0xD7	0xAB	0x76
0xCA	0x82	0xC9	0x7D	0xFA	0x59	0x47	0xF0	0xAD	0xD4	0xA2	0xAF	0x9C	0xA4	0x72	0xC0
0xB7	0xFD	0x93	0x26	0x36	0x3F	0xF7	0xCC	0x34	0xA5	0xE5	0xF1	0x71	0xD8	0x31	0x15
0x04	0xC7	0x23	0xC3	0x18	0x96	0x05	0x9A	0x07	0x12	0x80	0xE2	0xEB	0x27	0xB2	0x75
0x09	0x83	0x2C	0x1A	0x1B	0x6E	0x5A	0xA0	0x52	0x3B	0xD6	0xB3	0x29	0xE3	0x2F	0x84
0x53	0xD1	0x00	0xED	0x20	0xFC	0xB1	0x5B	0x6A	0xCB	0xBE	0x39	0x4A	0x4C	0x58	0xCF
0xD0	0xEF	0xAA	0xFB	0x43	0x4D	0x33	0x85	0x45	0xF9	0x02	0x7F	0x50	0x3C	0x9F	0xA8
0x51	0xA3	0x40	0x8F	0x92	0x9D	0x38	0xF5	0xBC	0xB6	0xDA	0x21	0x10	0xFF	0xF3	0xD2
0xCD	0x0C	0x13	0xEC	0x5F	0x97	0x44	0x17	0xC4	0xA7	0x7E	0x3D	0x64	0x5D	0x19	0x73
0x60	0x81	0x4F	0xDC	0x22	0x2A	0x90	0x88	0x46	0xEE	0xB8	0x14	0xDE	0x5E	0x0B	0xDB
0xE0	0x32	0x3A	0x0A	0x49	0x06	0x24	0x5C	0xC2	0xD3	0xAC	0x62	0x91	0x95	0xE4	0x79
0xE7	0xC8	0x37	0x6D	0x8D	0xD5	0x4E	0xA9	0x6C	0x56	0xF4	0xEA	0x65	0x7A	0xAE	0x08
0xBA	0x78	0x25	0x2E	0x1C	0xA6	0xB4	0xC6	0xE8	0xDD	0x74	0x1F	0x4B	0xBD	0x8B	0x8A
0x70	0x3E	0xB5	0x66	0x48	0x03	0xF6	0x0E	0x61	0x35	0x57	0xB9	0x86	0xC1	0x1D	0x9E
0xE1	0xF8	0x98	0x11	0x69	0xD9	0x8E	0x94	0x9B	0x1E	0x87	0xE9	0xCE	0x55	0x28	0xDF
0x8C	0xA1	0x89	0x0D	0xBF	0xE6	0x42	0x68	0x41	0x99	0x2D	0x0F	0xB0	0x54	0xBB	0x16

Table 2.4

AES Rcon Constants

0x01	0x00	0x00	0x00
0x02	0x00	0x00	0x00
0x04	0x00	0x00	0x00
0x08	0x00	0x00	0x00
0x10	0x00	0x00	0x00
0x20	0x00	0x00	0x00
0x40	0x00	0x00	0x00
0x80	0x00	0x00	0x00
0x1b	0x00	0x00	0x00
0x36	0x00	0x00	0x00

Table 2.5

AES MixingVars Constants

0x02	0x03	0x01	0x01
0x01	0x02	0x03	0x01
0x01	0x01	0x02	0x03
0x03	0x01	0x01	0x02

Table 2.6

AES InvMixingVars Constants

0x0e	0x0b	0x0d	0x09
0x09	0x0e	0x0b	0x0d
0x0d	0x09	0x0e	0x0b
0x0b	0x0d	0x09	0x0e

Table 2.7

AES InvSBox Constants

0x52	0x09	0x6A	0xD5	0x30	0x36	0xA5	0x38	0xBF	0x40	0xA3	0x9E	0x81	0xF3	0xD7	0xFB
0x7C	0xE3	0x39	0x82	0x9B	0x2F	0xFF	0x87	0x34	0x8E	0x43	0x44	0xC4	0xDE	0xE9	0xCB
0x54	0x7B	0x94	0x32	0xA6	0xC2	0x23	0x3D	0xEE	0x4C	0x95	0x0B	0x42	0xFA	0xC3	0x4E
0x08	0x2E	0xA1	0x66	0x28	0xD9	0x24	0xB2	0x76	0x5B	0xA2	0x49	0x6D	0x8B	0xD1	0x25
0x72	0xF8	0xF6	0x64	0x86	0x68	0x98	0x16	0xD4	0xA4	0x5C	0xCC	0x5D	0x65	0xB6	0x92
0x6C	0x70	0x48	0x50	0xFD	0xED	0xB9	0xDA	0x5E	0x15	0x46	0x57	0xA7	0x8D	0x9D	0x84
0x90	0xD8	0xAB	0x00	0x8C	0xBC	0xD3	0x0A	0xF7	0xE4	0x58	0x05	0xB8	0xB3	0x45	0x06
0xD0	0x2C	0x1E	0x8F	0xCA	0x3F	0x0F	0x02	0xC1	0xAF	0xBD	0x03	0x01	0x13	0x8A	0x6B
0x3A	0x91	0x11	0x41	0x4F	0x67	0xDC	0xEA	0x97	0xF2	0xCF	0xCE	0xF0	0xB4	0xE6	0x73
0x96	0xAC	0x74	0x22	0xE7	0xAD	0x35	0x85	0xE2	0xF9	0x37	0xE8	0x1C	0x75	0xDF	0x6E
0x47	0xF1	0x1A	0x71	0x1D	0x29	0xC5	0x89	0x6F	0xB7	0x62	0x0E	0xAA	0x18	0xBE	0x1B
0xFC	0x56	0x3E	0x4B	0xC6	0xD2	0x79	0x20	0x9A	0xDB	0xC0	0xFE	0x78	0xCD	0x5A	0xF4
0x1F	0xDD	0xA8	0x33	0x88	0x07	0xC7	0x31	0xB1	0x12	0x10	0x59	0x27	0x80	0xEC	0x5F
0x60	0x51	0x7F	0xA9	0x19	0xB5	0x4A	0x0D	0x2D	0xE5	0x7A	0x9F	0x93	0xC9	0x9C	0xEF
0xA0	0xE0	0x3B	0x4D	0xAE	0x2A	0xF5	0xB0	0xC8	0xEB	0xBB	0x3C	0x83	0x53	0x99	0x61
0x17	0x2B	0x04	0x7E	0xBA	0x77	0xD6	0x26	0xE1	0x69	0x14	0x63	0x55	0x21	0x0C	0x7D

2.1.3. AES CIPHERING ALGORITHM

The ciphering algorithm presented by NIST looks as follows [34]:

```
Step 0  $K = ExpandKey(K)$
Step 1  $State = M$
Step 2  $State = AddRoundKey(State, K[0, \ldots, Nb - 1])$
Step 3  $i = 1, \ldots, Nr - 1$ do Steps 4, 5, 6, 7
Step 4  $State = SubWords(State)$
Step 5  $State = ShiftRows(State)$
Step 6  $State = MixColumns(State)$
Step 7  $State = AddRoundKey(State, K[\mathrm{Nr} * Nb, \ldots, (i + 1) * Nb - 1])$
Step 8  do Steps 4, 5
Step 9  $State = AddRoundKey(State, K[Nr * Nb, \ldots, (Nr + 1) * Nb - 1])$
```

Note that putting the message to the *State* presented in Step 1 is described in sec. 2.1.2. After all these operations, *State* contains the cryptogram.

Deciphering algorithm presented by NIST [34]:

```
Step 0  $K = ExpandKey(K)$
Step 1  $State = AddRoundKey(State, K[Nr * Nb, \ldots, (Nr + 1) * Nb - 1])$
Step 2  $i = Nr - 1, \ldots, 1$ do Steps 3, 4, 5, 6
Step 3  $State = InvSubWords(State)$
Step 4  $State = InvShiftRows(State)$
Step 5  $State = AddRoundKey(State, K[i * Nb, \ldots, (i + 1) * Nb - 1])$
Step 6  $State = InvMixColumns(State)$
Step 7  do Steps 3, 4
Step 8  $State = AddRoundKey(State, K[0, \ldots, Nb - 1])$
```

After all these operations, *State* contains the original, deciphered message. All ciphering and deciphering functions are described in the next sections.

2.1.4. SubWords(State) AND InvSubWords(State)

This function uses *SBox* (see tab. 2.3) for changing byte values stored in *State* during ciphering, and *InvSBox* (see tab. 2.7) during deciphering.

Let's assume that $State(0, 0) = 0xa1$. $0xa1$ indicates which value from *Sbox* (during ciphering) will replace the value currently stored in $State(0, 0)$. In particular, the 4 most significant bits indicate the row in *SBox*, and the 4 least significant bits indicate the column number of *SBox* [34]. Thus, in our example: $State(0, 0)' = SBox(0xa, 0x1) = SBox(10, 1) = 0x32$, where $State(0, 0)'$ is the new value of $State(0, 0)$.

Function *SubWords* performs this substitution with the use of *Sbox* on all *State* values during ciphering (see eq. (2.1)), and function *InvSubWords* performs this substitution with the use of *InvSBox* on all *State* values during deciphering [34].

$$State'(i,j) = SBox\left(\left\lfloor \frac{State(i,j)}{16} \right\rfloor, State(i,j) - \left\lfloor \frac{State(i,j)}{16} \right\rfloor * 16\right), \quad i,j = 0,\ldots,3 \qquad (2.1)$$

Note that eq. (2.1) is also equivalent to eq. (2.2)

$$State'(i,j) = SBox(ShiftRight(State(i,j),1), State(i,j) \wedge 0xF), \quad i,j = 0,\ldots,3 \qquad (2.2)$$

2.1.5. ExpandKey(K)

As we mentioned earlier, the AES key can contain 4, 6 or 8 32-bit words, and the key size determines the number of rounds (Nr). In fact, in each round, we are operating on 4 32-bit words of the key. Practically, the input key words are used as a seed for generation of another $Nr * Nb$ different 32-bit words of K. The final size of K is equal to $(Nr + 1) * Nb$, which produces $Nr + 1$ vectors, each containing 4 32-bit words. These vectors can also be presented as 4×4 matrices containing bytes values (if we divide each 32-bit word into 4 bytes).

To expand the values of K, the following algorithm should be used [34]:

```
Step 1  $i = Nk, \ldots, Nk * Nb$ do Steps 2–7
Step 2  $tmp = K(i-1)$
Step 3  if $i \pmod{Nk} = 0$ do Step 4
Step 4  $tmp = RotateLeft(SubWord(tmp), 1) \oplus Rcon\left(\left\lfloor \frac{i}{Nk} \right\rfloor - 1\right)$
Step 5  if $Nk > 6 \wedge i \pmod{Nk} = 4$ do Step 6
Step 6  $tmp = SubWord(tmp)$
Step 7  $K.append(K(i - \mathrm{Nk}) \oplus tmp)$
```

Note that this algorithm assumes that the initial Nk 32-bit words of key K were generated, and $SubWords(tmp)$ is the same function as described in sec. 2.1.4, performed on one vector containing 4 bytes. $Rcon(i)$ denotes a 32-bit word stored in i-th row of the $Rcon$ constants table (see tab. 2.4).

2.1.6. ShiftRows(State) AND InvShiftRows(State)

The *ShiftRows* operation can be formally represented as in eq. (2.3), and *InvShiftRows* as in eq. (2.4) [34].

$$State'(i,j) = State(i, (i+j) \pmod 4), \quad i,j = 0,\ldots,3 \qquad (2.3)$$

$$State'(i,j) = State(i,(4-i+j) \pmod 4), \quad i,j = 0,\ldots,3 \tag{2.4}$$

In fact, *ShiftRows* performs left binary rotation by a row number, and *InvShiftRows* performs right binary rotation by a row number. If we represent *State* as 4 32-bit words, *ShiftRows* can be represented as in eq. (2.5), and *InvShiftRows* as in eq. (2.6), equivalently [34].

$$State'(i) = RotateLeft(State(i), i), \quad i = 0,\ldots,3 \tag{2.5}$$

$$State'(i) = RotateRight(State(i), i), \quad i = 0,\ldots,3 \tag{2.6}$$

After using *ShiftRows* or *InvShiftRows*, *State'* becomes a new *State*.

2.1.7. MixingColumns(State) AND InvMixingColumns(State)

The *MixingColumns* operation is based on a matrix multiplication and is presented in eq. (2.7). *InvMixingColumns* is also a matrix multiplication; however, the constant values are different, see eq. (2.8).

$$State' = MixingVars \times State \tag{2.7}$$

$$State' = InvMixingVars \times State \tag{2.8}$$

MixingVars are presented in tab. 2.5, and InvMixingVars in tab. 2.6. The important thing is that MixingVars and *InvMixingVars* represent the polynomial coefficients in $GF(2^8)$, and all calculations are done under this field [34]. Thus, multiplication of byte $a = [b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0]$ by 0*x*02 can be done as presented in eq. (2.9) [108].

$$a * 0x02 = \begin{cases} \text{if} \quad b_7 = 1 \ ShiftLeft(a,1) \oplus 0x1b \\ \text{else} \quad b_7 = 0 \ ShiftLeft(a,1) \end{cases} \tag{2.9}$$

The plus operator is represented by XOR: $a + b = a \oplus b$. All multiplications must be done with the use of appropriate combinations of adding and multiplications by 0*x*02. For example: if $a = 0xaa$ and $b = 0x09$ then $a * b = 0xaa * 0x09 = 0xaa * (0x02 * 0x02 * 0x02 + 0x01) = 0xaa * 0x02 * 0x02 * 0x02 + 0xaa = 0x4f * 0x02 * 0x02 + 0xaa = 0x9e * 0x02 + 0xaa = 0x27 + 0xaa = 0x8d$.

2.1.8. AddRoundKey(State, RoundKey)

This function performs the XOR operation on *RoundKey* words and corresponding State columns (see eq. (2.10)) [34].

$$State'(j,i) = State(j,i) \oplus RoundKey(i,j), \quad i,j = 0,\ldots,3 \tag{2.10}$$

State(j, i) denotes i-th byte in j-th 32-bit word of State, and *RoundKey*(i, j) denotes j-th byte in i-th 32-bit word of *RoundKey*. Note that:

- RoundKey is created by four 32-bit words taken from key *K* after key extension. The ciphering and deciphering algorithms presented in sec. 2.1.3 indicate which words should be taken for a given round.
- Key binary words are XORed with *State* columns. In other words, the first row of *State* will be XORed with a binary word created by the 4 most significant bytes of the *RoundKey* array.

2.2. SECURE HASH ALGORITHM 2

In this chapter, we will describe two classic hashing algorithms which are current Secure Hash Standards (SHS): SHA-256 and SHA-512. Before we describe the algorithms themselves, we present additional functions which will be useful [90]:

- *S* function [90]:

$$S(x,a,b,c) = RotateRight(x,a) \oplus RotateRight(x,b) \oplus RotateRight(x,c) \quad (2.11)$$

- *s* function [90]:

$$s(x,a,b,c) = RotateRight(x,a) \oplus RotateRight(x,b) \oplus ShiftRight(x,c) \quad (2.12)$$

- *Maj* function [90]:

$$Maj(a,b,c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c) \quad (2.13)$$

- *Ch* function [90]:

$$Ch(a,b,c) = (a \wedge b) \oplus (not(a) \wedge c) \quad (2.14)$$

For both of the hashing algorithms mentioned earlier, the messages have to have an appropriate byte length. Let's define the padding algorithm as well [90]:

1. *m* – byte array representing message
2. *n* – block size

```
Step 0  $pad(m, n)$:
Step 1  $tmp = len(m) * 8$
Step 2  $m.append(0x80)$
Step 3  $i = 0$
Step 4  while $i \pmod n \neq (n - 8)$ do Step 5, 6
Step 5  $m.append(0x00)$
Step 6  $i = i + 1$
Step 7  $m.append(bytearray(tmp, 8))$
Step 8  $return\ m$
```

Note that:

- *tmp* variable stores the number of bits in the original message.
- binary representation of $0x80$ is $(10\ 000\ 000)_b$. This means that this padding method assumes adding bit '1' to the message and later an appropriate number of '0' bits.

This method is called pad10*.

2.2.1. SHA-2 IN CLOUDS

Hashing functions are one of the most important tools in Cloud security aspects. The authors in [104] present a data integrity checking method with the use of hashing functions. The method presented by the authors calculates the hash value from data to be sent on the customer side. Their methodology allows one to avoid third-party auditors. Hashed data is stored in secured local repositories. When a client wants to download data from a Cloud, a hash can be calculated again and compared with the stored one. This assumption allows one to verify Cloud Provider credibility and compliance with the Service Level Agreement (SLA).

In [4], the authors presented a hybrid hashing security algorithm for data storage in Cloud. The authors claims that a hash function is the best solution to reach integrity in Cloud environments. However, combining a hashing function with secure cryptography algorithms (such as AES or RSA) can ensure confidentiality of data, integrity and non-repudiation. The authors' algorithm is named hybrid-SHA256.

The hashing function can be used without any additional security algorithms (such as storing passwords in databases or the method presented in [104]) and can be combined with different security techniques (such in digital signatures or as presented in [4]). As an another example, we can give an algorithm for secure data storage in Cloud Computing through Blowfish, RSA and Hash Function, as presented in [66].

In this chapter, we present two hashing function from the SHA-2 family. Note that in most of the algorithms, only the certificate hash function shall be used. Thus, using SHA-3 is also a secure solution (if the chosen security algorithm allows one to do so).

2.2.2. SHA-256

In this section, we present an algorithm of the calculation SHA-256 digest [90]. Note that the hash is calculated from byte array M, H are initial hash constants, and K are fixed constants given by NIST [90]. K and H values are 32-bit unsigned integers.

Step 1 H = (0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a, 0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19)

Step 2 K = (0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5, 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,

```
        0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174, 0xe49b69c1, 0xefbe4786,
        0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
        0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147,
        0x06ca6351, 0x14292967, 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
        0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85, 0xa2bfe8a1, 0xa81a664b,
        0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
        0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a,
        0x5b9cca4f, 0x682e6ff3, 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
        0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2)
Step 3  $M = pad(M, 64)$
Step 4  $i = 0$
Step 5  $tmp = (0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)$
Step 6  while $i < len(M)$ do Steps 7-23
Step 7  $W = ()$
Step 8  for $j = 0, \ldots, 15$ do Step 9
Step 9  $W.append\left(parseUint32\left(M\left(\left\lfloor \frac{i}{64} \right\rfloor * 64 + j * 4, \ldots, \left\lfloor \frac{i}{64} \right\rfloor * 64 + j * 4 + 3\right)\right)\right)$
Step 10 for $j = 16, \ldots, 63$ do Step 11
Step 11 $W.append(W(j - 16) + W(j - 7) + s(W(j - 15), 7, 18, 3) + s(W(j - 2), 17, 19, 10))$
Step 12 for $j = 0, \ldots, 7$ do Step 13
Step 13 $tmp(j) = H(j)$
Step 14 for $j = 0, \ldots, 63$ do Step 15-20
Step 15 $t1 = K(j) + W(j) + S(tmp(4), 6, 11, 25) + Ch(tmp(4), tmp(5), tmp(6)) + tmp(7)$
Step 16 $t2 = Maj(tmp(0), tmp(1), tmp(2)) + S(tmp(0), 2, 13, 22)$
Step 17 for $k = 7, \ldots, 1$ do Step 18
Step 18 $tmp(k) = tmp(k - 1)$
Step 19 $tmp(0) = t1 + t2$
Step 20 $tmp(4) = tmp(4) + t1$
Step 21 for $j = 0, \ldots, 7$ do Step 22
Step 22 $H(j) = H(j) + tmp(j)$
Step 23 $i = i + 64$
Step 24 return $H(0)||H(1)||H(2)||H(3)||H(4)||H(5)||H(6)||H(7)$
```

2.2.3. SHA-512

In this section, we present an algorithm of the calculation SHA-512 digest [90]. Note that the hash is calculated from byte array *M*, *H* are initial hash constants, and *K* are fixed constants given by NIST [90]. *K* and *H* values are 64-bit unsigned integers.

```
Step 1  $H$=(0x6a09e667f3bcc908, 0xbb67ae8584caa73b, 0x3c6ef372fe94f82b, 0xa54ff53a5f1d36f1,
        0x510e527fade682d1, 0x9b05688c2b3e6c1f, 0x1f83d9abfb41bd6b, 0x5be0cd19137e2179)
```

```
Step 2   $K$ = (0x428a2f98d728ae22, 0x7137449123ef65cd, 0xb5c0fbcfec4d3b2f, 0xe9b5dba58189dbbc,
         0x3956c25bf348b538, 0x59f111f1b605d019, 0x923f82a4af194f9b, 0xab1c5ed5da6d8118,
         0xd807aa98a3030242, 0x12835b0145706fbe, 0x243185be4ee4b28c, 0x550c7dc3d5ffb4e2,
         0x72be5d74f27b896f, 0x80deb1fe3b1696b1, 0x9bdc06a725c71235, 0xc19bf174cf692694,
         0xe49b69c19ef14ad2, 0xefbe4786384f25e3, 0x0fc19dc68b8cd5b5, 0x240ca1cc77ac9c65,
         0x2de92c6f592b0275, 0x4a7484aa6ea6e483, 0x5cb0a9dcbd41fbd4, 0x76f988da831153b5,
         0x983e5152ee66dfab, 0xa831c66d2db43210, 0xb00327c898fb213f, 0xbf597fc7beef0ee4,
         0xc6e00bf33da88fc2, 0xd5a79147930aa725, 0x06ca6351e003826f, 0x142929670a0e6e70,
         0x27b70a8546d22ffc, 0x2e1b21385c26c926, 0x4d2c6dfc5ac42aed, 0x53380d139d95b3df,
         0x650a73548baf63de, 0x766a0abb3c77b2a8, 0x81c2c92e47edaee6, 0x92722c851482353b,
         0xa2bfe8a14cf10364, 0xa81a664bbc423001, 0xc24b8b70d0f89791, 0xc76c51a30654be30,
         0xd192e819d6ef5218, 0xd69906245565a910, 0xf40e35855771202a, 0x106aa07032bbd1b8,
         0x19a4c116b8d2d0c8, 0x1e376c085141ab53, 0x2748774cdf8eeb99, 0x34b0bcb5e19b48a8,
         0x391c0cb3c5c95a63, 0x4ed8aa4ae3418acb, 0x5b9cca4f7763e373, 0x682e6ff3d6b2b8a3,
         0x748f82ee5defb2fc, 0x78a5636f43172f60, 0x84c87814a1f0ab72, 0x8cc702081a6439ec,
         0x90befffa23631e28, 0xa4506cebde82bde9, 0xbef9a3f7b2c67915, 0xc67178f2e372532b,
         0xca273eceea26619c, 0xd186b8c721c0c207, 0xeada7dd6cde0eb1e, 0xf57d4f7fee6ed178,
         0x06f067aa72176fba, 0x0a637dc5a2c898a6, 0x113f9804bef90dae, 0x1b710b35131c471b,
         0x28db77f523047d84, 0x32caab7b40c72493, 0x3c9ebe0a15c9bebc, 0x431d67c49c100d4c,
         0x4cc5d4becb3e42b6, 0x597f299cfc657e2a, 0x5fcb6fab3ad6faec, 0x6c44198c4a475817)
Step 3   $M = pad(M, 128)$
Step 4   $i = 0$
Step 5   $tmp$ = (0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)
Step 6   while $i < len(M)$ do Steps 7-23
Step 7   $W = ()$
Step 8   for $j = 0, \ldots, 15$ do Step 9
Step 9   $W.append\left(parseUint64\left(M\left(\left\lfloor \frac{i}{128} \right\rfloor * 128 + j*8, \ldots, \left\lfloor \frac{i}{128} \right\rfloor * 128 + j*8+7\right)\right)\right)$
Step 10  for $j = 16, \ldots, 79$ do Step 11
Step 11  $W.append(W(j-16) + W(j-7) + s(W(j-15), 1, 8, 7) + s(W(j-2), 19, 61, 6))$
Step 12  for $j = 0, \ldots, 7$ do Step 13
Step 13  $tmp(j) = H(j)$
Step 14  for j = 0, …, 79 do Step 15-20
Step 15  $t1 = K(j) + W(j) + S(tmp(4), 14, 18, 41) + Ch(tmp(4), tmp(5), tmp(6)) + tmp(7)$
Step 16  $t2 = Maj(tmp(0), tmp(1), tmp(2)) + S(tmp(0), 28, 34, 39)$
Step 17  for $k = 7, \ldots, 1$ do Step 18
Step 18  $tmp(k) = tmp(k-1)$
Step 19  $tmp(0) = t1 + t2$
Step 20  $tmp(4) = tmp(4) + t1$
Step 21  for $j = 0, \ldots, 7$ do Step 22
Step 22  $H(j) = H(j) + tmp(j)$
Step 23  $i = i + 128$
Step 24  return $H(0)||H(1)||H(2)||H(3)||H(4)||H(5)||H(6)||H(7)$ In this case,
         the digest is equal to 512 bits (8 * 64-bit word).
```

2.3. RIVEST-SHAMIR-ADLEMAN

Rivest-Shamir-Adleman (RSA) is an asymmetric block cipher. This algorithm is the most popular asymmetric scheme and overwhelms its competitors [108]. Published by the authors in 1978, RSA became a global standard very quickly. Accepted by NIST as Public Key Cryptography Standard 1 (PKCS1), RSA can also be used for digital signatures generation. The algorithm is based on the Discrete Logarithm Problem. The authors in [108] distinguish 4 main attacks which can be done on RSA:

- Brute force attack: checking all possible combinations to break the cipher [108].
- Mathematical attacks: factorisation of RSA parameters [108].
- Time attacks: which use the realisation time of particular RSA instructions during deciphering [108].
- Attacks with a chosen cryptogram: which use mathematical parameters of RSA [108].

To avoid all these possible problems, an appropriate key should be chosen. All details are described in the next sections.

2.3.1. RSA IN CLOUD

The authors in [93] proposed the use of RSA in Cloud environments. They proved that the use of RSA protects against unauthorised access to data. The performance of algorithms varies according to the size of the data. To compensate for these drawbacks, the authors in [39] proposed an algorithm called ERSA (Enhanced RSA), which is based on RSA. ERSA enable one to perform fast complex calculations and increase the speed of encryption and decryption. The authors state that ERSA can be used to ensure data security in the Cloud. In [119], the authors proposed a method for providing security in the Cloud via the classic RSA cryptosystem. They performed comprehensive studies and finally state that: *RSA provides high security in high potential data encryption methodology* [119].

RSA can also be used for digital signature creation. The authors in [105] proposed and tested a secure RSA digital signature scheme, which can be used in Cloud environments to ensure authentication, data integrity, privacy and non-repudiation (all main features which digital signature should have). We presented their algorithm in sec. 2.3.3.

RSA is also used by commercial Cloud providers, e.g.:

- Google Cloud KMS in asymmetric encryption schemes [41]. Currently, they also use SHA-256.
- Amazon: *The keys that Amazon EC2 use are 2048-bit SSH-2 RSA keys. You can have up to five thousand key pairs per Region* [9].

2.3.2. RSA CIPHERING SCHEME

Firstly, key pairs have to be generated according to the following algorithm [108]:

1. p, q – two distinct positive prime numbers with reasonable size.

Step 1 $n = p * q$
Step 2 $\phi(p, q) = (p - 1) * (q - 1)$
Step 3 find e such that: $1 < e < \phi(p, q) \wedge \gcd(e, \phi(p, q)) = 1$
Step 4 $d \equiv e^{-1} \pmod{\phi(p, q)}$

A public key PUB is created by pair (e, n) and a private key PRIV by pair (d, n). Note that:

- $\phi(p, q)$ is an totient function (also called Euler Function). If p and q are prime numbers, this totient function can be calculated as presented in Step 2.
- Step 4 presents a modular inverse of $e \pmod{\phi(p, q)}$. Practically, it means that: $d \equiv e^{-1} \pmod{\phi(p, q)}$ can be presented as $e * d \equiv 1 \pmod{\phi(p, q)}$. d can be found, e.g., via the Extended Euler Algorithm.
- The RSA key length is equal to the bit length of n. Thus, for example, RSA-1024 notation means that the bit length of n is equal to 1024.
- If number a is represented on a_d bits and number b on b_d bits, product $c = a * b$ will be represented on $a_d + b_d$ bits or $a_d + b_d - 1$ bits.

However, RSA parameters have some restrictions, which are presented in [89]:

- p, q and d shall be kept in secret. e and n can be shared [89].
- The bit length of n shall be equal to: 1024, 2048, 3072 bits or more. The bigger the n, the harder the factorisation problem becomes, and the time of computations is longer [89].
- Certification Authorities (CAs) shall use n equal to or longer than their subscribers [89].
- p and q shall be generated with an approved pseudorandom number generator [89].
- Seeds of pseudorandom prime generators shall be kept in secret (or destroyed) [89].

The ciphering scheme is presented in eq. (2.15), and the deciphering scheme in eq. (2.16) [108].

$$c = m^e \pmod{n} \tag{2.15}$$

$$m = c^d \pmod{n} \tag{2.16}$$

Note that:

- m is a message which will be ciphered. m must be converted into a number such that: $m < n$. If this condition is not fulfilled, m has to be divided into blocks such that each block of the message creates a number smaller than n ($m_{block} < n$). Each block is then ciphered separately.
- $m = m^{ed} \pmod{n}$.
- Everyone with the sender's public key can cipher the message; however, only sender the can decipher it with a private key.

2.3.3. RSA DIGITAL SIGNATURE

RSA can also be used for signing messages with digital signatures. Before presentation of the algorithm itself, let's introduce some additional notations:

- (e_s, n_s), (d_s, n_s) – signer public and private key, respectively.
- (e_v, n_v), (d_v, n_v) – verifier public and private key, respectively.
- H – hashing function. The same for signer and verifier.
- m – message which will be signed.

Basic signature creation is presented in eq. (2.17) [63].

$$s = m^{d_s} \pmod{n_s} \tag{2.17}$$

A signature s and message m are sent to the verifier, and value v is then calculated as presented in eq. (2.18). If $m = v$, verification is completed with success.

$$v = m^{e_s} \pmod{n_s} \tag{2.18}$$

However, this scheme was improved by the authors in [105]. Their scheme assumes the use of RSA signatures in Cloud environments. The improved scheme assumes that the signature s looks as presented in eq. (2.19):

$$s = (m^{e_v} \pmod{n_v}),\ H(m)^{d_s} \pmod{n_s}) \tag{2.19}$$

s is sent to the verifier, which calculates v as presented in eq. (2.20).

$$v = (H(s(0)^{d_v} \pmod{n_v})),\ s(1)^{e_s} \pmod{n_s}) \tag{2.20}$$

If $v(0) = v(1)$, verification is completed with success. Note that everyone can verify the digital signature with the sender's public key, but only the sender can create it with a private key.

For signature purposes, additional security requirements are presented by NIST [89]:

- The key pair used for the digital signature scheme can be used only once and shall not be used for any other purposes [89].
- Only approved hash functions shall be used for digital signature schemes. The security of the chosen hashing function cannot be lower than the security of modulus n [89].
- *The length in bits of the hash function output block shall meet or exceed the security strength associated with the bit length of the modulus n* [89].
- The security strength of the hash function and modulus n shall exceed the security required by the digital signing process [89].

2.3.4. BLIND RSA

Considering Cloud Computing, another version of RSA, called Blind RSA, may be used. Each result of the computation is firstly blinded by a VM that computed the task. The blinded results are sent to the SP, and there they are decrypted using the RSA procedure. Using such an extension of the basic algorithm, the SP is not aware of the exact data that will be stored (as the black box coding engine). In the case of transmitting data to the user, the results of the computation are retrieved from the SP and decoded by VM. Only the authorised VM may decode the results.

Let's assume the same notation as presented in sec. 2.3.2. Public and private keys are generated in the same way; however, the ciphering scheme is a little bit different. If the VM wants the SP to cipher the message $m \in \{0, 1, \ldots, n\}$ blindly, m is multiplied by $k^e \bmod n$, where k is a randomly chosen number called the blinding factor. The VM then sends the blinded message $m * k^d \bmod n$ to the SP. Next, the SP ciphers the blinded message, which results in $c = (m * k^d)^e \bmod n$, and stores it. Finally, the VM unblinds the message by multiplying c by $k^{-1} \bmod n$, which results in:

$$c * k^{-1} \bmod n = (m * k^d)^e * k^{-1} \bmod n =$$

$$m^e * k^{de} * k^{-1} (\bmod n) =$$

$$m^e * kk^{-1} (\bmod n)$$

and deciphers if:

$$(m^e)^d (\bmod n) = m$$

2.4. ELLIPTIC CURVE CRYPTOGRAPHY (ECC)

Elliptic Curves (ECs) allows one to implement some cryptographic algorithms in a different way. In this section, we present the Elliptic Curve Digital Signature Algorithm, ElGamal cryptosystem implemented with ECs usage and all formulas which are necessary for understating these algorithms. The main reason why ECs are gaining popularity is the key size. Algorithms based on ECs use much smaller keys for the encryption and decryption process, which results in much faster (even many times faster) computations in comparison to classical asymmetric algorithms (such as RSA). Less time of computations results in less memory usage and less power consumption. What is more, the smaller keys in ECC do not cause a decrease in overall security. However, one of the biggest drawbacks of ECC is the complexity of algorithms and complexity of implementation. Cryptograms can also be bigger in comparison to RSA.

There are three main types of ECs [24]:

- ECs over real numbers.
- ECs over binary fields (also called EC over F_{2^m}).
- ECs over finite fields (also called EC over F_p).

In ECC, curves over F_{2^m} and over F_p are used. We will focus on the third type (ECs over F_p). All necessary formulas and notation are presented in sec. 2.4.2.

2.4.1. ECC IN CLOUDS

In [7], the authors claim that ECC is currently becoming more and more popular in Cloud environments due to its smaller cryptography key sizes (which leads to the less memory consumption and reduction of computational time) and less power consumption. These two factors are very important, e.g., in mobile applications. They also proposed an encryption scheme which is basing on the ElGamal cryptosystem with ECC usage. Their conclusion is: *ECC had provided a robust and secure model for the development and deployment of a secure application in the Cloud. This work would promote confidence in both large and small scale organisations in Cloud investment* [7].

In [48], the authors demonstrated an encryption scheme, decryption scheme and digital signature scheme with ECC usage. The authors also pointed out that currently there are only a few known attacks on Elliptic Curves which won't work if EC parameters are chosen according to the security requirements.

In [100], the authors described how to effectively and securely use ECDSA in Cloud systems. They also compared the effectiveness of the RSA signature scheme and ECDSA. They pointed out that using ECC also ensures faster access to the memory in comparison to other existing Digital Signature Algorithms.

The examples mentioned above are only a drop in the ocean. It is possible to give many more examples of ECC usage in Cloud computing environments. If we take into consideration that ECC is also used by private Cloud providers, such as:

- Google Cloud: in asymmetric signing algorithms. Curves P-256 and P-384 [40].

- Amazon Web Services: in CloudFront for supporting ECDSA Certificates for HTTPS Connections to Origins [8].
- Microsoft Azure: in the cryptographic library for creating ECDSA [83].

We can assume that it is an important technique which is worthy to be described in the Cloud security context.

2.4.2. EC DOMAIN PARAMETERS

Elliptic Curve E over F_p is a curve given by formula (2.21) [24], [116]:

$$E : y^2 \pmod p = x^3 + ax + b \pmod p \tag{2.21}$$

Where [116]:

- $p > 3$, and p is a prime number.
- $a, b \in [0, p-1]$ and $(4a^3 + 27b^2) \pmod p \neq 0 \pmod p$

Each curve used for cryptographic purposes must define the following parameters (see tab. 2.8):

Table 2.8

EC basic parameters and notation

Symbol	Name	Description
O	Point at infinity	This is EC additive identity [24]. Every EC has point O.
$\#E$	EC order	Total number of points (including point O) generated by the curve E (see eq. (2.21)) [24], [116]. E should be a prime number.
$G(g_x, g_y)$	EC generator	Randomly chosen point which belongs to E (see eq. (2.21)) [116]. Note that: $g_x, g_y \in [0, p-1]$ and $g_y^2 \pmod p = g_x^3 + ag_x + b \pmod p$.
n	EC generator order	In such case, the order of the generator means that for any number x from range $[1, n-1]$, xG will be a point on a curve; however $nG = O$. Practically n is indicating how many times G can be multiplied. n must be a prime number and: $n > 2^{160} \wedge n > 4\sqrt{p}$ [116].

The idea behind ECC is similar to the modulus arithmetic. EC, which fulfils all the conditions above, creates a finite number of distinct points. Adding a point to another point (under some conditions mentioned later) will produce a point at the curve. Multiplication works in the same way. When the curve is big enough, some points create subgroups within the group generated by a curve. Multiplication of a point from this subgroup will produce another point from this subgroup. Formally, it can be said that EC can create cyclic abelian groups [108], such as the generator mentioned in tab. 2.8.

Let's assume that an appropriate EC was generated (see eq. (2.21)) and $A = (a_x, a_y)$ and $B = (b_x, b_y)$ are two points which belong to the curve. $-A$ is equal to $(a_x, -a_y \pmod p)$ [24]. To perform an adding operation, the following algorithm should be used [24]:

1. $A(a_x, a_y) + B(b_x, b_y) = C(c_x, c_y)$
2. A and B are distinct points
3. $A \neq -B$

Step 1 $\quad s = \dfrac{a_y - b_y}{a_x - b_x} \pmod p$

Step 2 $\quad c_x = s^2 - a_x - b_x \pmod p$

Step 3 $\quad c_y = -a_y + s * (a_x - c_x) \pmod p$

To double point A, the following algorithm shall be used [24]:

1. $2B(b_x, b_y) = C(c_x, c_y)$
2. $b_y \neq 0$

Step 1 $\quad s = \dfrac{3b_x^2 + a}{2b_y} \pmod p$

Step 2 $\quad c_x = s^2 - 2b_x \pmod p$

Step 3 $\quad c_y = -b_y + s * (b_x - c_x) \pmod p$

To multiply a point, doubling and adding operations must be combined. For example: $9B = 8B + B = 2(4B) + B = 2(2(2B)) + B$. This can be done, for example, via the binary powering algorithm. Note that points are usually multiplied by a huge number, and thus naive powering method will be useless.

Finding the appropriate curve and calculation of all necessary parameters is generally complicated. Thus, NIST in [89] presented curves which are considered as secure. We present three of them in tab. 2.9, tab. 2.10 and tab. 2.11.

Table 2.9

Curve P-192 [89]

a	3
b	0x64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1
p	6277101735386680763835789423207666416083908700390324961279
n	6277101735386680763835789423176059013767194773182842284081
g_x	0x188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012
g_y	0x07192b95ffc8da78631011ed6b24cdd573f977a11e794811

Table 2.10

Curve P-256 [89]

a	3
b	0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b
p	115792089210356248762697446949407573530086143415290314195533631308 867097853951
n	115792089210356248762697446949407573529996955224135760342422259061 068512044369
g_x	0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0 f4a13945d898c296
g_y	0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ece cbb6406837bf51f5

Table 2.11

Curve P-512 [89]

a	3
b	0x051953eb9618e1c9a1f929a21a0b68540eea2da725b99b315f3b8b489918ef10 9e156193951ec7e937b1652c0bd3bb1bf073573df883d2c34f1ef451fd46b503f00
p	686479766013060971498190079908139321726943530014330540939446345918 554318339765605212255964066145455497729631139148085803712198799971 6643812574028291115057151
n	686479766013060971498190079908139321726943530014330540939446345918 554318339765539424505774633321719753296399637136332111386476861244 0380340372808892707005449
g_x	0xc6858e06b70404e9cd9e3ecb662395b4429c648139053fb521f828af606b4d3d baa14b5e77efe75928fe1dc127a2ffa8de3348b3c1856a429bf97e7e31c2e5bd66
g_y	0x11839296a789a3bc0045c8a5fb42c7d1bd998f54449579b446817afbd17273e6 62c97ee72995ef42640c550b9013fad0761353c7086a272c24088be94769fd16650

Note that:

- a, p and n are given in decimal format, while b and the coordinates of G in hexadecimal format.
- Users can calculate their own G with different n (G and n must fulfil all conditions mentioned in this chapter). The rest of the parameters can remain unchanged.
- Notation $P - X$ indicate that the curve is defined over X bit's prime number.
- According to the NIST Special Publication 800-57 [15], curves P-160 to P-223 offer the same security as RSA-1024, curves P-224 to P-255 offer the same security as RSA-2048, and curves P-256 to P-383 offer the same security as RSA-3072. RSA needs a 15360-bit key to reach the same security level as curve P-512 (and higher) [15].

2.4.3. ELGAMAL CRYPTOSYSTEM

ElGamal is an asymmetric cipher which can be implemented with Elliptic Curve (EC) usage. A description of particular variables is presented in sec. 2.4.2.

To represent how the algorithm works, let's assume communications between *A* and *B*. Both sides of communications must choose the same [21]:

- *a* and *b* EC parameters.
- Finite field F_p.
- Generator *G* of EC.
- The same algorithm of mapping message *M* into EC.

To generate cryptographic keys, *A* and *B* have to [21]:

Step 1 choose natural number *k* such that: $k \in [1, n - 1]$
Step 2 calculate point $Q = k * G$

$Q_a = (q_x^a, q_y^a)$ is a public key of *A*, and k_a is a private key of *A*. Note that Q_a is a point, and k_a is a number. *B* generates keys analogously. After the keys were generated, both sides need to validate them. An appropriately generated public and private key fulfils the following conditions [21]:

- $Q \neq O$ and $nQ = O$ [21].
- $(q_x, q_y) \in F_p$ and $Q \in E$ [21].

Before ciphering, message *M* has to be encoded as a point on the chosen curve *E*. This is quite challenging and may reduce the speed of the overall ciphering process. The authors in [84] described a Koblitz method which allows one to overcome this problem.

One of the biggest downsides is that the hardness of encoding message *M* into curve *E* depends on the private key *k*, and this is not always possible [84].

If we assume that message *M* was successfully encoded as a point on a curve (let's denote this point by $MP = (mp_x, mp_y))$ and that *A* wants to send this message to *B*, the ciphering scheme presented in [21] looks as follows:

Step 1 Choose natural number *r* such that: $r \in [1, n - 1]$
Step 2 Compute $c_1 = r * G$
Step 3 Compute $c_2 = r * Q_b + MP$
Step 4 Send cryptogram $C = (c_1, c_2)$ to the *B*

To decipher cryptogram $C = (c_1, c_2)$, *B* must perform the following operations [21]:
Step 1 $V = k_b * c_1$
Step 2 $MP = c_2 - V$

After deciphering, *B* must also map point *MP* back to message *M*.

To avoid the whole message mapping process, the bauthors in [26] proposed a slightly different ciphering scheme based on ElGamal, called Hash ElGamal. Let's assume the same notation and scenario as in the ElGamal encryption; however, A and B have access to the same hash function denoted by H, and we are considering M as a binary string representing the message. Then, if A wants to encrypt a message, the following algorithm should be used [26]:

Step 1 $c_1 = r * G$
Step 2 $c_2 = H(Q_b * r) \oplus M$
Step 3 Send cryptogram $C = (c_1, c_2)$ to the B

To decrypt a message, B must perform the following instructions [26]:

Step 1 $M = H(k_b * c_1) \oplus c_2$

Note that in the Hash ElGamal encryption scheme:

- Binary length of message M cannot be longer than the digest length produced by hashing function H. Otherwise, the message must be divided into blocks with appropriate bit length.
- Ciphering and deciphering processes assume calculating a hash from a point of curve E.
- Hash ElGamal is partially homomorphic in the field Z_p [26].

2.4.4. ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM

The Elliptic Curve Digital Signature Algorithm (ECDSA) is an American National Standard (ANSI X9.62), created in 1998 [116]. A description of particular variables is presented in sec. 2.4.2. Additionally, the Signer and Verifier have access to the same hashing function H. The Signer private key is k_a, and public key is $Q_a = (q_x^a, q_y^b)$. To create a signature from message M, the following algorithm has to be used by the Signer [116]:

Step 1 Choose a such that: $a \in [1, n - 1]$
Step 2 Compute $P(p_x, p_y) = a * G$
Step 3 $r = p_x \pmod{n}$
Step 4 if $r = 0$ go to Step 1
Step 5 if $bitlen(H(M)) > bitlen(n)$ compute $e = ShiftRight(H(M), bitlen(H(M)) - bitlen(n))$, else compute $e = H(M)$
Step 6 Compute $s = a^{-1} * (e + r * k_a) \pmod{n}$. If $s = 0$ go to Step 1.
Step 7 Send (r, s) to the Verifier.

Verifier get signature (r, s) and message M. To validate signature, Verifier must perform the following steps [116]:

Step 1 If r or s are not in interval $[1, n - 1]$ signature is invalid.
Step 2 Compute $c = s^{-1} \pmod{n}$
Step 3 if $bitlen(H(M)) > bitlen(n)$ compute $e = ShiftRight(H(M), bitlen(H(M)) - bitlen(n))$, else compute $e = H(M)$
Step 4 Compute $u_1 = e * c \pmod{n}$
Step 5 Compute $u_2 = r * c \pmod{n}$
Step 6 Compute $P(p_x, p_y) = u_1 * G + u_2 * Q_a$. If $P = O$ signature is invalid.
Step 7 Compute $v = p_x \pmod{n}$
Step 8 If $v = r$ signature is valid.

Note that:
- Signature (r, s) is made of numbers, not points.
- Step 5 in the signature creation algorithm (and corresponding Step 3 in verification algorithm) assuming taking $bitlen(n)$ leftmost bits of hash value as an e parameter (if hash is longer than $bitlen(n)$). e can be greater than n; however, it cannot be longer. The original ANSI X9.62 assumes the use of the SHA-1 function, which produces 160-bit hashes (bit length of n is always greater than 160 bits). In such a case, whole hash values can be taken as e; however, NIST in 2015 announced that SHA-1 should not be used anymore. It is recommended to use SHA-2 or SHA-3, and in this case, there exists a possibility that the bit length of n will be smaller than a hash length. Truncation is then needed.

2.5. SHAMIR SECRET SHARING

Tasks from the same batch are calculated and stored in different locations. Nevertheless, to process the batch further or to send it back to the user, CC has to assure that all elements of the batch were gathered, and no single part of the computational chain has lost its share. The proposed scheme may be an alternative for traditional methods for information integrity checking that use hash functions like SHA-2 or SHA-3.

Secret sharing procedure enables one to distribute certain knowledge among the group of participants so that only a predesignated collection of them are able to recreate the knowledge by collecting and combining their shares, [101]. The secret is split among t VMs that are chosen out of n to run the batch of tasks. The completeness of the results may be checked by recreating the secret from them. Such a procedure can guarantee that less than t participants are not able to recreate the secret. Let's assume that the first participant of the secret is the unit sending a batch to workers (dealer), and we have n participants in the system, and that the batch was split among $t - 1$ VMs so that each VM has at least one task from that batch to run. The Shamir Scheme [101] uses polynomial interpolation over finite field $GF(q)$, where $q >= n + 1$.

1. *Secret splitting*. The dealer choose n distinct nonzero elements from $GF(q)$:

$$x_1, x_2, \ldots, x_n$$

   and allocates them among participants. In the next step dealer fixes element $K \in GF(q)$ as the secret. The shares of the secret are created by the following scheme:

   - Dealer sets elements

$$a_1, a_2, \ldots, a_{t-1} \in GF(q)$$

     randomly, uniformly and independently.
   - if the $a(x) = K + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$ is the polynomial of degree $t - 1$ then shares are defined as $y_i = a(x_i)$ for $i = 1, 2, \ldots t$.

2. *Retrieving the secret*. If all t participants gather their shares together (dealer and chosen VMs), they formulate the set of t points (x_i, y_i) of the polynomial a. Using the Lagrangian Interpolation, there is a possibility to find the unique polynomial of degree $t - 1$ passing throughout these points. The secret is found by taking the value of that polynomial at point 0. Shares may be also computed from the system of linear equations:

$$y_1 = K + a_1 x_1 + a_2 x_1^2 + \ldots + a_{t-1} x_1^{t-1}$$

$$y_2 = K + a_1 x_2 + a_2 x_2^2 + \ldots + a_{t-1} x_2^{t-1}$$

$$y_t = K + a_1 x_t + a_2 x_t^2 + \ldots + a_{t-1} x_t^{t-1}$$

   where y_i for $i = 1, 2, \ldots t$ are known shares,

$$K, a_1, a_2, \ldots, a_{t-1} \in GF(q)$$

   are unknown. The solution of this system, including secret K, may be found (e.g. via Gaussian Elimination Method) only if exactly t participants gave their shares.

2.6. COUNTER EXAMPLE

One of the best symmetric stream ciphers is the Vernam Cipher. The algorithm is presented in [102].

Ciphering process:

$$C = M \oplus K \tag{2.22}$$

Deciphering process:

$$M = C \oplus K \tag{2.23}$$

where: Vernam uses the XOR operation to produce the ciphertext [102].

Table 2.12

Vernam Cipher Symbols

C	Cryptogram
M	Message
K	Key

The resulting bit is equal to "1" if the input bits are different and "0" if they are the same [102]. If Key is generated appropriately, the Vernam Cipher becomes an ideal cipher. Theoretically, it is extremely hard to break within a reasonable time, even with all the computational power from all over the world.

The only problem is the key generation. There are several technical solutions for generation of random bit strings. Some research points out that even images (chosen properly) or different files can be used as a key [5]. As an example of a pseudorandom bit generator, we can also point out Blum Blum Shub (BBS) [30].

However, the Vernam Cipher, despite its strength, has many drawbacks, which are presented in [61]:

- Key has to be equal or longer than the message.
- Key cannot be reused. Practically, this means that half of the communication process will be spent on key exchange.
- Security channel for key exchange is necessary.

These drawbacks cause a reduction of usage of the Vernam Cipher in Cloud environments. Most commercial Cloud Providers do not support stream encryption, because it is replaced by faster symmetric block encryption. Despite the fact that there exist articles which propose the use of variations of stream ciphers in Cloud environments (such as [91]), they will probably not be useful in practice.

Most deadly errors arise from obsolete assumptions
Frank Herbert, Children of Dune

3. SECURITY ASPECTS IN CLOUD SYSTEMS

3.1. INTRODUCTION

The Cloud Computing (CC) model incorporates multiple stakeholders for providing scalable on-demand access to a shared pool of configurable computing resources. The liability of service quality in the CC model is on the Provider side [3]. Security is the next objective after assuring proper infrastructure and software for elastic, complex and heterogeneous Cloud Services. The basic security services that have to be implemented in the Computational Clouds are the same as for traditional systems:

- the preservation of **confidentiality** – information/data/services should be accessible only for authorised users;
- **integrity** – information/data/services should be accurate and complete; and
- **availability** – ensuring access on demand.

The necessity for additional security services comes from the fact that physical resources are shared by a lot of customers. Secret sharing, auctioning and voting protocols are main examples of specialised algorithms used for protection resources with many participants. The next aspect of any Computational Cloud is the large number of end-users. This results in higher vulnerability to inside threats or security ignorance. Moreover, the complex and heterogeneous architecture and massive scale of offered services make the system more vulnerable to unintentional security threats like users errors. All security components, including cryptography algorithms, protocols and schemes, need to be established, implemented, monitored, reviewed and improved constantly. This is an obligation under international law regarding the right to privacy and many international standards and regulations, e.g. ISO/IEC 27002:2013 Norm or NIST Special Publication 800-53. This chapter presents different aspects of security in Computational Clouds.

3.2. IDENTIFICATION OF SECURITY PROCEDURES IN CLOUD INFRASTRUCTURE

When it comes to cryptography elements, computational Cloud architecture can be divided into several parts, see fig. 3.1 [44]:

- Cloud consumer's part: responsible for cryptography solutions for consumption of the services,
- Cloud provider's part: supporting secure orchestration, secure deployment, security of services, secure recourse abstraction and security of the physical layer, as well as secure resource management,
- Cloud broker's part: providing secure service aggregation and secure wrapping services [44].

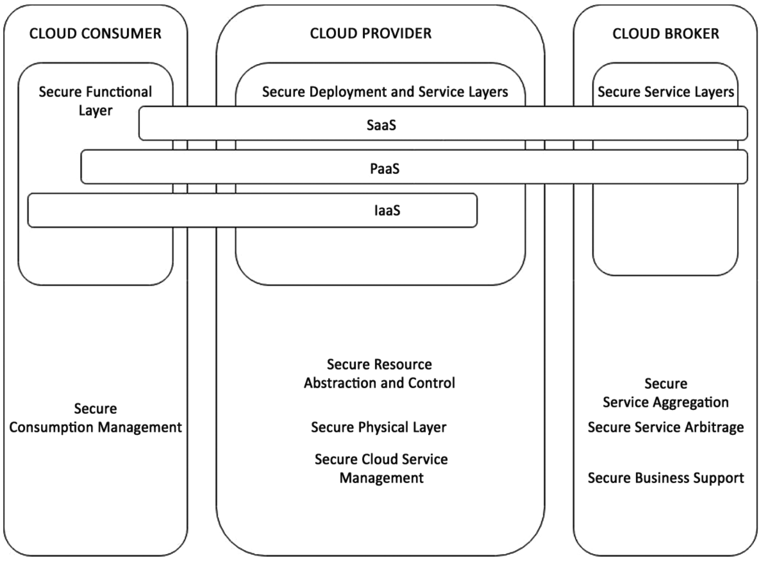


Fig. 3.1. Chosen security preserving methods in a Computational Cloud ecosystem presented in [44]

Considering CC layers, the following cryptography element placement is necessary [88]:

- Software as a Service method supporting secure use of applications running on a Cloud infrastructure. The broad access of applications from various client devices, including thin clients, web browsers and tablet interfaces, has to be considered.
- Platform as a Service dedicated methods. Cryptography tools assuring security of programming languages, libraries, data basis.

- Infrastructure as a Service (IaaS). Cryptography enabling processing, storage, networking.

As far as a deployment model is considered, different cryptography services and trust levels have to be considered for:

- Private Clouds governing very sensitive data under strong authentication and authorisation of strictly defined customers.
- Public Cloud cryptography assuring services are open for public use. These systems need to handle many lean or security-unaware clients on a massive scale.
- Hybrid Cloud cryptography tools serving as security for the composition of private, community or public Clouds. The main challenge is the unification of cryptography security levels among different providers [99].

Regarding infrastructure layers, the following components may be listed:

- Physical layer cryptography Methods dedicated to physical resources and their interactions. Examples include servers, storage disks, modems and switches.
- Virtual layer methods support virtual resources. Examples of such resources may be VMs, virtual networks and virtual storage disks.
- Application layer schemes secure applications hosted using virtual resources.

The International Organization for Standardization Standard ISO/IEC 19790:2012(E), entitled Information Technology Security techniques, Security requirements for cryptographic modules, specifies four levels for cryptography modules as apart of general security requirements:

1. Security Level 1: at least one approved algorithm or approved security function shall be used;
2. Security Level 2: role-based authentication in which a cryptographic module authenticates the authorisation;
3. Security Level 3: identity-based authentication mechanisms implemented. Moreover, the input or output has to be processed inside a secure module with a trusted path from other interfaces. Plain-text may be passed through the cryptographic module in encrypted form;
4. Security Level 4: the highest level of security defined in this standard. Penetration of the cryptographic module is closed from any direction. A security threat has a very high probability of being detected, resulting in secure backup of the system running [50].

In such a complex system, load balancing is necessary. The proper mapping of data and tasks in Cloud systems are realised by schedulers. This implies that the initial requirements and assumptions as far as the strength of the cryptographic shield over tasks and data is considered, and they have to be properly mapped into units serving as workers, see fig. 3.2 [3]. Moreover, the developer is obligated to provide a sufficient amount of VMs meeting these needs. Load balancers and schedulers have to be informed about these cryptography services divided into categories, measurable and verifiable. Additionally, they have to optimise not only the parameters of schedules concerning time and efficiency but additional criteria considering fulfilment of the security requirements. This results in the necessity of multistage optimisation problem solving [55, 56, 69].

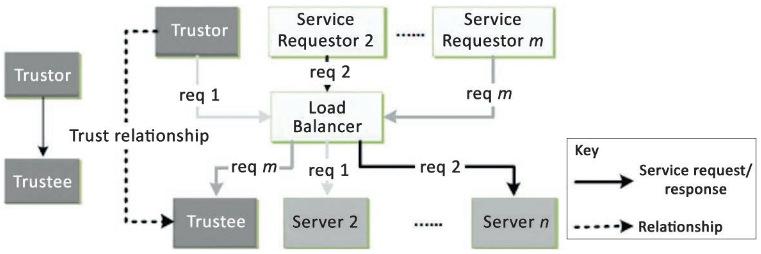


Fig. 3.2. Mapping security requirements by schedulers presented in [3]

Horizontal and Vertical Scaling service usage also has deep consequences. They are available on-demand for CC users. The scaling of one component results in cascaded scaling at other dependent resources. This may be done at the same level of infrastructure in the form of horizontal scaling or at the previous (next) level of infrastructure, namely as vertical scaling. This enforces the frequent integrity checking and authentication mechanisms used for each consecutive part of the process, see fig. 3.3.

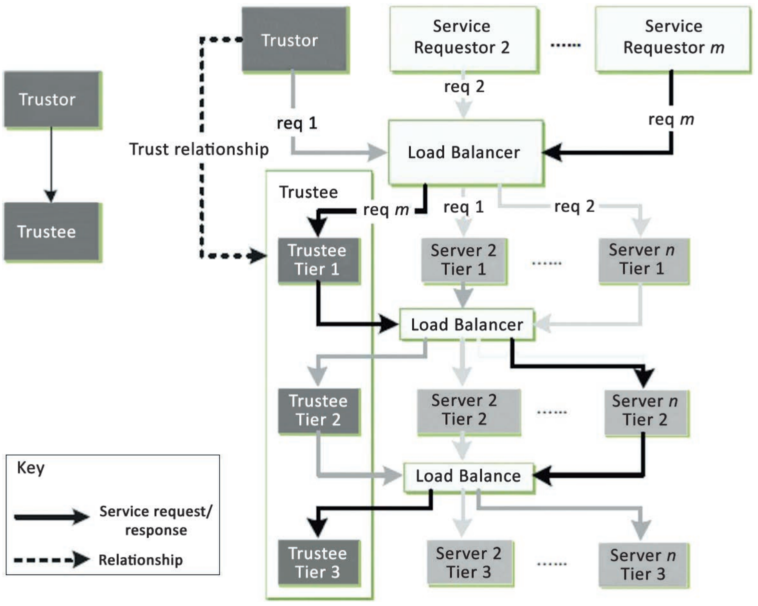


Fig. 3.3. Horizontal matching of security requirements presented in [3]

The Cloud model assumes the provision of services regardless of problems with the physical equipment or virtual resource failures. Therefore, backup, trust connections with other servers and transparent migrations to new physical units or virtual resources are necessary to be established and to be ready for use at any time. The backup scenarios have to take into consideration the secure backup of cryptography keys, privileges, rights, restrictions and roles of the customer, see fig. 3.4.

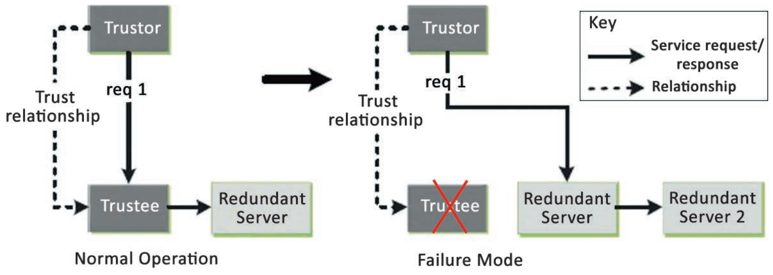


Fig. 3.4. Redundancy of trust connection presented in [3]

Virtualisation enables a single hardware system to concurrently run multiple isolated virtual machines. The IaaS layer is based on many virtualised components: virtual machines, hypervisors and virtual networks [22]. Solutions dedicated for such a resource may be:

- guest OS Isolation [107],
- isolation of VM systems with mandatory access control mechanisms [54],
- encryption and hashing of the state of VMs before saving,
- data in transit between VM protection by securing the network using VLANs [46].

Clouds offer services on a massive scale. Therefore, security establishment requires automated solutions. Verifying trust chain elements has to be done with minimum human intervention. Moreover, performance Analysis of Data Encryption Algorithms or key generation systems should be done to avoid using inefficient solutions. Cryptographic protocols dedicated to Big Data Systems are often the best choice for CC systems. Among them are Multi-Party Computation [18], or Functional Encryption [6].

Computational Clouds process a large variety of data types. Data at rest, being stored on psychical units in digital form, has to be treated differently than data in motion over an infrastructure. Data in use (active data) stored in non-persistent units, typically in computer random access memory, CPU caches or CPU registers, also needs dedicated solutions. Data in transit may be moved using several paths: Firstly, from the customer endpoint (computer, laptop, tablet) using the Internet and a web service provided by the CC. The data or metadata is then transferred between

Services and Virtual Machines inside the Cloud. Lastly, the data has to be transited physically, for example, to store it onto hard drives. Examples of a dedicated solution for encryption of data at rest may be found in [96]. Techniques for encryption of data in motion are presented, for example, in [94]. And finally, the encryption of data in use is described in [103].

Security of data storage inside Clouds may be supported via:

- a sequential revocation game where Cloud users have to choose the best strategy from a set of three strategies to revoke and eliminate the malicious user while minimising their costs [58],
- proactively Secure Cloud-Enabled Storage [37],
- auditing and monitoring mechanisms to detect and prove violations of security properties [31],
- a secure cloud storage system based on discrete logarithm problems [121],
- strong Key-Exposure Resilient Auditing for Secure Cloud Storage [120],
- valet security model [20].

Sensitive data is defined in a very broad sense. It is treated as any information which compromised can adversely affect the interest of the data owner or the privacy rights of individuals. The use of such data is regulated by law. For example, the Data Protection Act of 1998 in the United Kingdom [49], specifies personal data stored on computers or in an organised paper filing system as a sensitive data when it contains: the racial or ethnic origin, political opinions, religious beliefs or other beliefs of a similar nature, physical or mental health or condition, sexual life. User passwords, credit card information and security policy numbers are further examples of data being under special control.

The processing of such data is defined as obtaining, recording, holding, or carrying out any operation or set of operations on it, such as:

- organisation, adaptation or alteration of the information or data,
- retrieval, consultation or use of the information or data,
- disclosure of the information or data by transmission, dissemination or otherwise making available, or
- alignment, combination, blocking, erasure or destruction of the information or data.

In Europe, sensitive data is protected by Directive 95/46/EC. This regulation aims at the protection of individuals with regard to the processing of personal data and to the free movement of such data [1]. There is an analogous law which specifies sensitive data processing in the USA and many other countries. An example of dedicated algorithms for encryption of sensitive data may be found in [57], [115] or [45].

3.3. TRANSPARENCY CONTRARY TO SECURITY BALANCE

Some layers of the Cloud have to be transparent or cannot be accessible to customers and others. In such layers, the following aspects have to be taken into special consideration: cryptographic algorithms involved, the length of the keys and libraries used for implementation. The transparency strategy has to be updated according to the newest stage of cryptology achievements, known threats and possible attacks on the Cloud infrastructures.

Despite these kinds of layers, a customer must have the possibility to control the security of the data, processes and infrastructure which is rented. This is why Service Level Agreements (SLAs) are used. SLAs are standardised contracts where cryptography services are formally defined. Negotiating the contracts and monitoring of their fulfilment in real-time is possible. Assuring SLA conditions without revealing details about schemes and procedures is possible by external independent audits and systems of certificates and norms. The chosen international standards which can be used for appropriate SLA formulation are presented in tab. 3.1 and in tab. 3.2.

SLA formulation, enforcement and compliance is necessary. Example aspects which SLA shall consider are: performance metric in the case of multiple class customers, trustworthiness of resources, response time of service (see [62]) or automated processing (see [106]).

Table 3.1

Chosen Cloud relevant ISO/IEC and NIST standards

Tile	Content
NIST SP 800-145	Definition of Cloud Computing
ISO/IEC 17788:2014	Cloud Computing terms and definitions
ISO/IEC 17789:2014	Cloud computing architecture
ISO/IEC 27000:2014	Information security management systems, vocabulary
ISO/IEC 27001:2013	Information security management systems
ISO/IEC 27002:2013	Code of practice for information security controls
ISO/IEC FDIS 27017 (2015)	Information security controls for Cloud services
ISO/IEC 27018:2014	Protection of PII in public Clouds acting as PII processors
ISO/IEC 29100	Privacy principles for information storage and processing
ISO/IEC 27040:2015	Storage security
ISO/IEC 17203:2011	Virtualisation Format
NIST SP 800-146	Cloud Computing Synopsis and Recommendations
NIST SP 500-292	NIST Cloud Computing Reference Architecture

Table 3.2

Chosen certificates for Cloud systems

Standard	Content
ISO 27001	Internationally accepted independent security standard
SSAE16 / ISAE 3402	Assurance standard for information security
SAS 70 audit	Internal control of a service organisation
ISO 27017	Cloud Security standard
ISO 27018	Cloud Privacy standard
HIPAA	Use of Protected Health Information
PCI DSS	Payment Card Industry Data Security Standard

3.4. SECURITY CONTRARY TO COMPUTING EFFICIENCY

The Computational Cloud is based on the pay-as-you-go method. The provider charges the customers based on the resources they used. To do this effectively, the computational costs of all cryptography services have to be measured, monitored and properly assigned to individual customers. Cost reduction should not be possible from the users' side by using an inappropriate low security level. On the other hand, setting exaggerated security levels is also negative. This will slow the processing of data and will lead to unnecessary exploitation of resources. Customers should be informed about the increase in charges due to the specified cryptography services used.

If the system is well protected from inside threads, lightweight cryptography can be used [51], for example:

- PHOTON: lightweight hash-function computing hash-codes of length 80, 128, 160, 224 and 256 bits,
- SPONGENT: lightweight hash-function computing hash-codes of length 88, 128, 160, 224 and 256 bits,
- Lesamnta-LW: a lightweight hash-function computing a hash-code of length 256 bits,
- PRESENT: a lightweight block cipher with a block size of 64 bits and a key size of 80 or 128 bits,
- CLEFIA: a lightweight block cipher with a block size of 128 bits and a key size of 128, 192 or 256 bits.

3.5. SUMMARY

The presented methods and techniques are designed for reliable building of CC systems, but implementing them is not voluntary. CC systems serve broad public communities, academic communities and private companies. The security assurance in such systems is guided by international norms, standards and even by international law.

For example, the ISO/IEC 27000 norm for Information Security Management Systems provides requirements for many systems, including Clouds. This standard contains 11 security controls containing a total of 39 main security categories. The 11 clauses include, for example, Security Policy setting, Organising Information Security, Access Control procedure customisation and Information Security Incident Management.

Another list was prepared by Cloud Security Alliance (CSA) in the form of Security Guidance for Cloud Computing. CSA proposed the Cloud Security Reference Model and a list of security controls. They described 14 Governance Domains and Guidance for dealing with them.

This complex list of security controls was proposed in the form of the Cloud Security Alliance Controls Matrix. The matrix is the mapping of 35 international norms and standards (including ISO 27002/27017/27018) into one list. The matrix specifies 16 main security domains, all with subdomains. This results in 132 Control Domains in total. For each domain, architectural relevance is marked. The considered architectural parts of the Cloud systems are: Physical Layer, Network layer, Compute, Storage, Application and Data Layers. Additionally, domains are assigned into 3 Cloud Service Delivery Models: SaaS, PaaS or IaaS.

All lists given above, as well as guidance and norms, do not specify any strategy stating which of the security controls will be beneficial for implementation in a specific Cloud system.

The surest way to keep a secret is to make someone think they already know the answer
Frank Herbert, Heretics of Dune

4. CYBERATTACK MODELLING TECHNIQUES OVERVIEW

4.1. INTRODUCTION

To secure a system properly, all dangers should be taken into consideration. Experiments and tests should be done in a controlled environment. However, to perform such tests, dangers have to be appropriately modelled. The negative impact of potential threats should also be measured. This will be the main topic of this chapter.

NIST Glossary of Key Information Security Terms [68] defines the three basic concepts that will be used during the modelling of cyberattacks:

- A threat is a circumstance or event that has an adverse impact on the considered system operations (system functions, image or even reputation), assets and users.
- A vulnerability is a weakness in the system itself, in the security procedures, security internal controls or implementation that may be exploited or triggered by a threat source.
- Risk is the level of impact resulting from a threat with a potential detrimental impact.

4.2. MODELLING THE DYNAMIC OF THE ATTACK

4.2.1. ATTACK GRAPHS

Liu Xuezhong [76] ordered a vulnerability set in the form of exploited paths. The model assumes the dependencies between some vulnerabilities. If a chosen threat can't be applied easily, or there is no vulnerability-exploiting path to the certain vulnerability, the threat resulting from that vulnerability will be weakened; otherwise, the effect of the threat will be increased. These vulnerability-exploiting paths were used to construct a security threat model based on the attacking-path graph.

A set of security vulnerabilities is denoted by:

$$\{U_1, U_2, \ldots, U_n\} \tag{4.1}$$

and the vector of system authority threats is denoted by:

$$\{T_1, T_2, \ldots, T_n\} \tag{4.2}$$

The feasibility matrix is then introduced. The element in the j-th row and j-th column $P_{(j,j)}$ denotes the feasibility that an attacker executes during the attack with the use of vulnerability j. It is scaled to the interval [0, 1). The element in the i-th row and j-th column, $P_{(i,j)}$ equals the feasibility that the attacker will perform during exploitation of vulnerability j, after the exploitation of vulnerability i. If there is no exploitation path from U_i to U_j, then $P_{(i,j)} = 0$. An acyclic digraph, see fig. 4.1, presents the dependencies between threats and vulnerabilities.

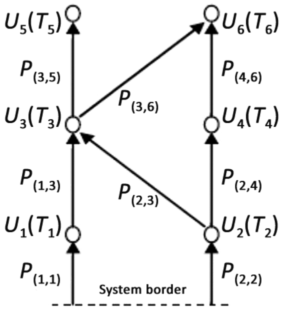


Fig. 4.1. Legend of attack presented in [76]

Pontus Johnson et al. [60] provided an attacker-centric threat modelling method for threat identification and quantification based on network modelling. The model matches the assets of a network with attack steps. This procedure allows one to find how these assets can be compromised and what the consequences for the rest of the assets are. The generated network is used for generating an attack graph. In the proposed model, the attack graph is an edge-weighted directed graph $G = (V, I, E, w)$, where:

- V is a set of nodes, which are the attack steps;
- I is a subset of V that denotes the starting point(s) of an attack;
- E is a set of directed edges $E \in VV$ and is equal to the possible progression of the attacker during a successful attempt of attack steps;
- The weight function $w : (A, B) \in E \rightarrow P(TTC_A)$ defines the probability distribution over time that an attacker will successfully perform an attack step (i.e. TTC).

Calculation of the TTC value is a two-steps process. Firstly, each edge of the graph is formulated by drawing a sample from its TTC probability distribution. The sampled value becomes the weight of the edge. It represents the TTC of the edge target attack step, under the assumption that the attacker has successfully attempted the edges' source attack step. Secondly, Dijkstra's shortest path algorithm is used to calculate the smallest TTC value for each attack-step, depending on its ancestry.

4.2.2. ATTACK SURFACE

Pratyusa K. Manadhata, in [78], proposed lowering the security risk by measuring and reducing attack surfaces. The attacked system is modelled as in the I/O automata model [77]. For a particular system S, the attack surface is triple, consisting of the set of entry points and exit points, the set of channels and the set of untrusted data items. The authors proposed estimating the damage caused by the attack in terms of the attributes of the resource. The main metric was the Damage Potential-Effort Ratio, which estimates the contribution of resources to the attack surface.

4.2.3. KILL CHAIN

A cyber kill chain is another tool for modelling the stages of cyberattacks. The kill chain is also used as a network defence. The model assumes that every threat must follow several stages, see fig. 4.2. For stopping an attack in progress, Defensible

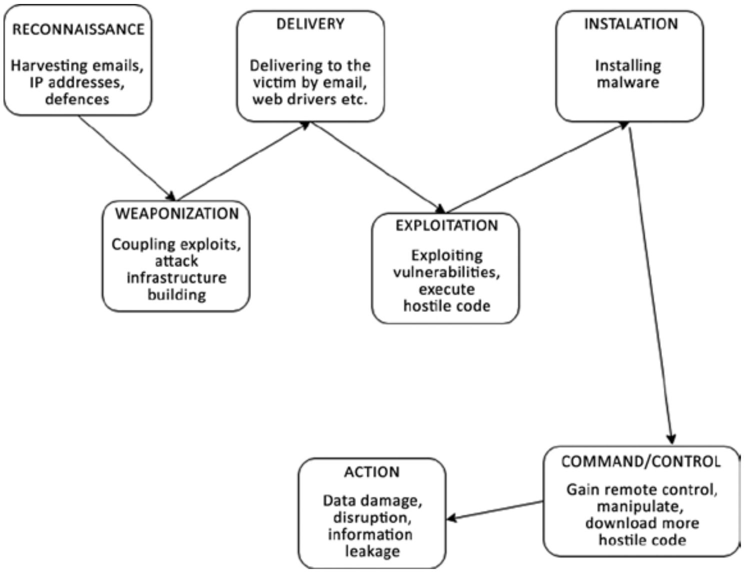

RECONNAISSANCE
Harvesting emails, IP addresses, defences
DELIVERY
Delivering to the victim by email, web drivers etc.
INSTALATION
Installing malware
WEAPONIZATION
Coupling exploits, attack infrastructure building
EXPLOITATION
Exploiting vulnerabilities, execute hostile code
ACTION
Data damage, disruption, information leakage
COMMAND/CONTROL
Gain remote control, manipulate, download more hostile code

Fig. 4.2. Kill chain analysis of an advanced thread presented in [125]

Actions (DAs) have to be taken. DAs are also presented in the form of a chain of actions.

- Detect: determine whether an attacker is poking around.
- Deny: prevent information disclosure and unauthorised access.
- Disrupt: stop or change outbound traffic (to the attacker).
- Degrade: counter-attack command and control.
- Deceive: interfere with command and control.
- Contain: network segmentation changes.

Using this methodology, Mujahid Mohsin and Zahid Anwar [86] proposed protecting Internet of Things (IoT) systems against Advanced Persistent Threats (APTs), see fig. 4.3.

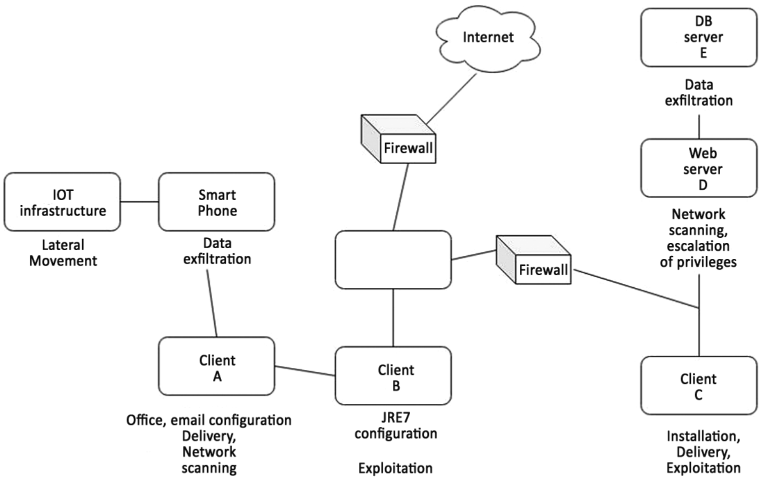


Fig. 4.3. Kill chain example in simple IoT network presented in [86]

4.2.4. ATTACK TREES MODELS

Xiaoli Lin et al. [75] presented attack trees for modelling Cross-Site Request Forgery (CSRF) attack against Web application users. In the attack tree, the attacker's goal is the root node. The possible actions are represented as leaf nodes, see fig. 4.4. The purpose of such modelling is to clarify conditions on with the adversaries may reach their targets and help to find week points of the system.

Rajesh Kumar et al. [70] proposed Attack-Fault Trees (AFTs) for the decomposition of the security attack into smaller sub-goals, until decomposition is possible. The leaves of proposed trees are the Basic Component Failures (BCFs), the Basic Attack Steps (BASs) or on-demand Instant Failures (IFAILs). The tree structure is represented by a directed acyclic graph. The logical gates are used for modelling dependencies between leaves.

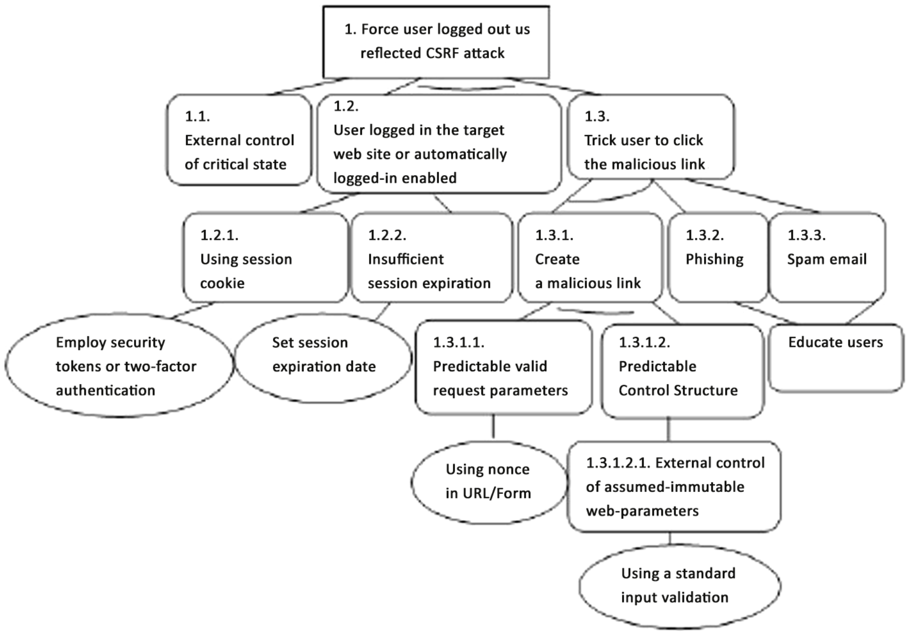

1. Force user logged out us reflected CSRF attack
1.1. External control of critical state
1.2. User logged in the target web site or automatically logged-in enabled
1.3. Trick user to click the malicious link
1.2.1. Using session cookie
1.2.2. Insufficient session expiration
1.3.1. Create a malicious link
1.3.2. Phishing
1.3.3. Spam email
Employ security tokens or two-factor authentication
Set session expiration date
1.3.1.1. Predictable valid request parameters
1.3.1.2. Predictable Control Structure
Educate users
Using nonce in URL/Form
1.3.1.2.1. External control of assumed-immutable web-parameters
Using a standard input validation

Fig. 4.4. Attack Trees Modelling example presented in [75]

4.2.5. PETRI NETS

A Petri Net (PN) is a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, represented by bars) and places (i.e. conditions, represented by circles). The directed arcs describe which places are pre- and/or postconditions for each of the transitions (denoted by arrows) [97].

M. Szpyrka [112] proposed a model of risk propagation among connected parts of the modelled system. A static structure composed of n components (nodes, elements, subsystems) is the representation of the system. The function f maps the component set into a chosen set V. This function may be a probability function, then $V = [0, 1]$. In general, the set V may be any numerical set or enumerated set, e.g. $V = \{\text{high, medium, low}\}$. Values of f depend on values of the function f used on other components. The internal dependencies are modelled in the form of modified coloured Petri Nets [59]. The Propagation Nets are built to map trigger places representing the possible attack points into sequences of transitions. The authors introduce the transitions that represent internal dependencies and transitions representing changes of the function f values caused by the system environment. The result of the modelling is to obtain all possible paths of risk propagation.

W. C. Moody et al. [87] proposed the use of Stochastic Petri Net (SPN) for security threat defence and deceptive defence tactics.

4.2.6. MARKOV PROCESSES

In [64], D. A. Karras used Markov models for modelling states of each system component inside interconnected infrastructures with respect to possible attacks. The model introduces 7 states of the system. State 0 represents the case when there are no attack attempts. The first attack changes the state into state 1. The system is in this state when it is under undetected attack but has not been penetrated (ATTACK IS NOT SUCCESSFUL). From state 1, the transition is made back to state 0 if an attack is detected or into state 2 if an attack is successful. The system remains in state 2 as long as confidential information is processed. The system is in state 3 if it is modified by the attacker. The system is in state 4 if the access of authorised users to programs, hardware and data are tied by the attacker. When an attacker is detected, the system is in state 5. The transition from state 0 to state 6 occurs if a false alarm is reported.

Markov Chain models the transition rate from state i to state j, the transition probability from state i to state j and the probability of the system or network or infrastructure to be in state i [81], for $i, j \in \{0,1, \ldots, 6\}$. Equilibrium (steady-state) equations for the Cloud model are established.

4.3. ATTACK CONSEQUENCES MEASUREMENT MODELS

4.3.1. MEAN FAILURE COSTS MODEL

Rabai et al. [95] proposed quantitative security metrics in the form of the vector of mean failure costs.

A system S is modelled using H_1, H_2, H_3, H_k, stakeholders, i.e. parties that have a stake in the system. They may be, for example, H_1-Cloud provider, H_2, H_3, H_k Cloud end users. The authors proposed R_1, R_2, R_3, R_n security requirements that should be incorporated into the system. $ST_{i,j}$, for $1 <= i <= k$, and $1 <= j <= n$ denotes the stake that stakeholder H_i has in ensuring requirement R_j expressed as a real cost of operation unit (for example in dollars per hour). For example, if R_1 is the integrity of critical data, $ST_{1,1}$ = 100 dollars per hour, $ST_{2,1}$ = 0.01 dollars per hour.

If we denote by PR_j, for $1 <= j <= n$, the probability that the security requirement R_j is not introduced into the system, then MFC (Mean Failure Cost), for $1 <= i <= k$, is the random variable describing stakeholder' H_i cost which may result from a security failure:

$$MFC_i = \sum_{j=1}^{n} ST_{i,j} PR_j \tag{4.3}$$

This random variable value is also expressed in dollars/hour. The Stakes Matrix ST is formulated from elements $ST_{i,j}$, for $1 <= i <= k$, and $1 <= j <= n$.

For C_1, C_2, C_3, C_h components of system S, the authors define a set of complementary events:

- E_l, $1 <= l <= h$ means that the work of component C_l is affected by a security threat,
- E_{h+1} means that none of the components are affected.

F_j denotes that the system fails to fulfil requirement R_j. Later, the probability of failure with respect to R_j may be represented by conditional probabilities as:

$$PR_j = P(F_j) = \sum_{l=1}^{h+1} P(F_j \| E_l) P(E_l) \qquad (4.4)$$

According to the law of total probability, the equation above is true when events E_l are disjointed, their probabilities are greater than zero, and their union formulates the probability space. This condition has to be fulfilled for all conditional probability equations formulated in this paragraph.

DP (Dependency) Matrix is formulated as $DP_j^l = P(F_j \| E_l)$ matrix, which has n rows and $h + 1$ columns, and where the entry at row j and column l is the probability that the component l failed to fulfil requirement j. PE is the vector concatenated from $P(E_l)$.

T_1, T_2, T_3, T_p, denotes threats which have occurred, and T_{p+1} is the event that no threat has occurred.

Additionally, PT_q, for $1 <= q <= p$, is the probability that threat T_q was active during a unitary period of time (for example 1 hour). PT_{p+1} is the probability that there was no threat in that time. Next, we may write the probability:

$$P(E_l) = \sum_{q=1}^{p+1} P(E_l \| T_q) PT_q \qquad (4.5)$$

IM (Impact) Matrix has $h + 1$ rows and $p + 1$ columns. Entry at row k and column q is the probability that the component C_k fails because threat q has occurred. When $q = p + 1$, no threats have occurred.

$$IM_{l,q} = P(E_l \| T_q) \qquad (4.6)$$

Vector PT of length $p + 1$ is formulated from elements PT_q.

Given the stakes matrix ST, the dependency matrix DP, the impact matrix IM and the threat vector PT, the vector of mean failure costs (one entry per stakeholder) is then given by the following formula (4.7):

$$MFC = ST * DP * IM * PT \qquad (4.7)$$

Where matrix *ST* is formulated collectively by the stakeholders, matrix *DP* is obtained by the systems architect, matrix *IM* is taken from security analysts, and vector *PT* is gathered from security analysis of perpetrator models.

4.3.2. SECURITY THREAT MEASUREMENT MODEL (STMM)

Lai et al. proposed in [71] the Security Threat Measurement Indicator (STMI), representing a three-layered security model. For each of the layers, the relevant measure was proposed, represented by the value from the interval [0, 1]. The first layer of the Threats Measurement Model is SLSM: System Level Security Measurement. It is divided into three categories:

- UAMP: User Authorization Management Procedure with W_1^1: weight of UAMP,
- LMM: Logging and Monitoring Mechanism with W_2^1: weight of LMM,
- SMSC: Security Management System Certification with W_3^1: weight of SSLC.

The aggregated measure for this layer is:

$$\mathrm{SLSM} = \mathrm{UAMP} * W_1^1 + \mathrm{LMM} * W_2^1 + \mathrm{SMSC} * W_3^1 \tag{4.8}$$

with the additional condition $W_1^1 + W_2^1 + W_3^1 = 1,\ W_1^1, W_2^1, W_3^1 \in [0,1]$.

The second layer is MLSM: Management Level Security Measurement. It is also divided into three categories:

- UQIP: User Qualification Inspection Procedure with W_1^2: weight of UQIP,
- RCCM: Regulatory Compliance Contracts Management with W_2^2: weight of RCCM,
- DLMP: Data Location Management Procedure with W_3^2: weight of DLMP.

The proposed measure for this layer is:

$$\mathrm{MLSM} = \mathrm{UQIP} * W_1^2 + \mathrm{RCCM} * W_2^2 + \mathrm{DLMP} * W_3^2 \tag{4.9}$$

under the condition $W_1^2 + W_2^2 + W_3^2 = 1,\ W_1^2, W_2^2, W_3^2 \in [0,1]$.

Finally, TLSM: Technique Level Security Measurement formulates the third layer:

- RSIA: Routine Security Inspection Activities with W_1^3: weight of RSIA,
- EDRP: Events Detection and Recovery Procedure with W_2^3: weight of SEDP,
- STEA: Security Technique Enhance Ability with W_3^3: weight of STLE.

The proposed measure for the third layer is:

$$\mathrm{TLSM} = \mathrm{RSIA} * W_1^3 + \mathrm{SEDP} * W_2^3 + \mathrm{STEA} * W_3^3 \tag{4.10}$$

under the condition $W_1^3 + W_2^3 + W_3^3 = 1,\ W_1^3, W_2^3, W_3^3 \in [0,1]$.

Finally, the authors combined SLSM, MLSM and TLSM in the form of a Security Threat Measurement Indicator (STMI), with $W1$: Weight of SLSM, $W2$: Weight of MLSM, $W3$: Weight of TLSM:

$$\text{STMI} = \text{SLSM} * W1 + \text{MLSM} * W2 + \text{TLSM} * W3 \tag{4.11}$$

where $W1 + W2 + W3 = 1$, and $W1$, $W2$, $W3 \in [0, 1]$.

The authors proposed the following use of these metrics in CC systems:

- Step 1. Collecting and normalising security factors.
- Step 2. Discovering Cloud security threats.
- Step 3. Identifying unacceptable threats.
- Step 4. Enforcing improvements.
- Step 5. Repeating steps 1-4 regularly.

4.3.3. EMPIRICAL RISK ASSESSMENT EQUATION

Noha E. El-Attar proposed in [36] the risk assessment function, based on NIST likelihood and impact factors [111], but they introduced numerical values instead of descriptive ones: Risk factor = (likelihood rate ∗ impact rate) ∗ [1 – (percentage of current control/100) + (percentage of uncertainty degree/100)], where:

- Likelihood is High – The threat is highly motivated and sufficiently capable. Controls to prevent the vulnerability from being exercised are ineffective.
- Likelihood is Medium – The threat-source is motivated and capable, but controls are in place that may impede the successful exercise of the vulnerability.
- Likelihood is Low – The threat-source lacks motivation or capability, or controls are in place to prevent, or at least significantly impede, the vulnerability from being exercised.

Impact rate is defined as:

- Impact is High when the exercise of the vulnerability may result in the highly costly loss of major tangible assets or resources; may significantly violate, harm or impede an organisation's mission, reputation or interest; may result in human death or serious injury.
- Impact is Medium when the exercise of the vulnerability may result in the costly loss of tangible assets or resources; may violate, harm, or impede an organisation's mission, reputation or interest; may result in human injury.
- Impact is Low when the exercise of the vulnerability may result in the loss of some tangible assets or resources; may noticeably affect an organisation's mission, reputation or interest.

In the proposed system, likelihood rate and impact factors are measured as quantitative values:

- *likelihoodrate* = 1 for Likelihood Very High
- *likelihoodrate* 0.7 to 0.9 for Likelihood High
- *likelihoodrate* 0.4 to 0.6 for Likelihood Medium
- *likelihoodrate* 0.2 to 0.3 for Likelihood Low
- *likelihoodrate* = 0.1 for Likelihood Very Low

The *impact rate* for each threat is in [0.1, 1] interval. The Current Control parameter measures the degree (in %) of control of the vulnerability, while the uncertainty percentage is represented by describer unexpected errors in %:

- *percentage of current control* ∈ [0, 100]
- *percentage of uncertainty degree* ∈ [0, 100]

Therefore, *Risk factor* takes values from 0 to 2.

4.3.4. COMMON VULNERABILITY SCORING SYSTEM METRICS

The Common Vulnerability Scoring System (CVSS) is an open industry standard proposed by the National Infrastructure Advisory Council (NIAC) [2]. The standard introduces the following set of metrics:

1. Base Metrics: used for vulnerabilities that may cause a successful attack:
   - Attack Vector (AV): representing vulnerabilities which can be exploited:
     - Network (N): a vulnerability exploitable with network access. The vulnerable component is bound to the network stack, and the attacker's path is through OSI layer 3 (the network layer). Such a vulnerability is often termed *remotely exploitable*. An attack can be performed one or more network hops away. An example of a network attack is when an attacker causes a Denial of Service (DoS) by sending specially crafted TCP packets across the public Internet (e.g. CVE-2004-0230).
     - Adjacent (A): a vulnerability exploitable with adjacent network access. The vulnerable component is bound to the network stack: however, the attack is limited to the same shared physical (e.g. Bluetooth, IEEE 802.11) or logical (e.g. local IP subnet) network and cannot be performed across OSI layer 3 boundaries (e.g. via a router). An example of an Adjacent attack would be an ARP spoof (IPv4) or neighbour discovery (IPv6).
     - Local (L): a vulnerability exploitable with local access. The vulnerable component is not bound to the network stack, and the attacker's path is via read/write/execute capabilities. In some cases, the attacker may be logged in locally in order to exploit the vulnerability: otherwise, he may rely on User Interaction to execute a malicious file.
     - Physical (P): a vulnerability exploitable with physical access. The attacker physically touches or manipulates the vulnerable component. Physical interaction may be brief (e.g. evil maid attack1) or persistent. An example of such an attack is a cold boot attack, which allows an attacker access to disk encryption keys after gaining physical access to the system, or peripheral attacks such as Firewire/USB Direct Memory Access attacks.
   - Attack Complexity (AC): describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability:

   - AC=Low (L): specialised access conditions or extenuating circumstances do not exist. An attacker can expect repeatable success against the vulnerable component.
   - AC=High (H): a successful attack depends on conditions beyond the attacker's control. That is, a successful attack cannot be accomplished ad hock but requires the attacker to invest in some measurable amount of effort in preparation or execution against the vulnerable component. For example, a successful attack may depend on overcoming any of the following conditions: the attacker must conduct target-specific reconnaissance (e.g. target configuration settings, sequence numbers, shared secrets, etc); the attacker must prepare the target environment to improve exploit reliability (e.g. overcoming advanced exploit mitigation techniques); the attacker must inject himself into the logical network path between the target and the resource requested by the victim in order to read and/or modify network communications (e.g. man in the middle attack).
- Privileges Required (PR): describes the level of privileges which an attacker must possess before successfully exploiting the vulnerability:
   - PR=None (N): the attacker is unauthorised before the attack and therefore does not require any access to settings or files to carry out an attack.
   - PR=Low (L): the attacker is authorised with privileges that provide basic user capabilities that could normally affect only settings and files owned by a user. Alternatively, an attacker with low privileges may have the ability to cause an impact only on non-sensitive resources.
   - PR=High (H): the attacker is authorised with privileges that provide significant (e.g. administrative) control over the vulnerable component that could affect component-wide settings and files.
- User Interaction (UI): the requirement for a user, other than the attacker, to participate in the successful compromise of the vulnerable component. This metric determines whether the vulnerability can be exploited solely at the will of the attacker, or whether a separate user (or user-initiated process) must participate in some manner:
   - UI=None (N): the vulnerable system can be exploited without interaction from any user.
   - UI=Required (R): successful exploitation of this vulnerability requires a user to take some action before the vulnerability can be exploited. For example, a successful exploit may only be possible during the installation of an application by a system administrator.
- Scope (S): a vulnerability in one software component which can influence resources or privileges beyond it:

    - S=Unchanged (U): an exploited vulnerability can only affect resources managed by the same authority. In this case, the vulnerable component and the impacted component are the same.
    - S=Changed (C): an exploited vulnerability can affect resources beyond the vulnerable component. In this case, the vulnerable component and the impacted component are different.
2. Impact Metrics: the properties of the impacted component:
   - Confidentiality Impact (C):
     - C=High (H): there is a total loss of confidentiality, resulting in all resources within the impacted component being divulged to the attacker. Alternatively, access to the restricted information is obtained. The disclosed information presents a direct, serious impact. For example, an attacker steals the administrator's password or private encryption keys of a web server.
     - C=Low (L): there is some loss of confidentiality. Access to some restricted information is obtained, but the attacker does not have control over what information is obtained, or the amount or kind of loss is constrained. The information disclosure does not cause a direct, serious loss to the impacted component.
     - C=None (N): there is no loss of confidentiality within the impacted component.
   - Integrity Impact (I): measures the impact of a successfully exploited vulnerability to the component's integrity:
     - I=High (H): there is a total loss of integrity or a complete loss of protection. For example, the attacker is able to modify any/all files protected by the impacted component. Alternatively, only some files can be modified, but malicious modification would present a direct, serious consequence to the impacted component.
     - I=Low (L): modification of data is possible, but the attacker does not have control over the consequences of a modification, or the amount of modification is constrained. The data modification does not have a direct, serious impact on the attacked component.
     - I=None (N): there is no loss of integrity within the impacted component.
   - Availability Impact (A): measures the influence of a vulnerability to the availability of the impacted component:
     - A=High (H): there is a total loss of availability, resulting in fully denied access to resources in the impacted component; this loss is either sustained (while the attacker continues the attack) or persistent (the condition persists even after the attack has been completed). Alternatively, the attacker has the ability to deny some availability, but the loss of availability presents a direct, serious consequence to the impacted

component (e.g. the attacker cannot disrupt existing connections but can prevent new connections, or the attacker can repeatedly exploit the vulnerability. In each instance of a successful attack, only a small amount of memory leaks, but repeated exploitation, causes a service to become completely unavailable).

- A=Low (L): there is a reduced performance of components or interruptions in resource availability. Even if repeated exploitation of the vulnerability is possible, the attacker does not have the ability to completely deny service to legitimate users. Resources in the impacted component are either partially available at any time or not fully available all the time. Overall, there are not direct, serious consequences to the impacted component.
- A=None (N): impacted component is fully available.

3. Temporal Metrics: measures the current state of exploitation techniques, existence of any patches or workarounds and the confidence that known vulnerabilities have been documented:
   - Exploit Code Maturity (E). This measures the likelihood of the usage of a particular vulnerability in the code and is typically based on the current state of exploit techniques, exploit code availability or active, *in-the-wild* exploitation. The public availability of easy-to-use exploits in code increases the number of potential attackers (including those who are unskilled). Initially, real-world exploitation may be only theoretical. Publication of proof-of-concept code, functional exploitable code or sufficient technical details necessary to exploit the vulnerability may change theory into practice. Furthermore, the available exploitable code may progress from a proof-of-concept demonstration to exploitable code that allows one to use vulnerabilities consistently:
     - E=Not Defined (X): assigning this value to the metric will not influence the score. It is a signal to a scoring equation to skip this metric.
     - E=High (H): functional autonomous code exists, no exploit is required (manual trigger), and details are widely available. Exploitable code works in every situation or is actively being delivered via an autonomous agent (such as a worm or virus). Network-connected systems are likely to discover exploitation attempts. Exploit development has reached the CVSS v3.0 Specification (v1.7) 13 / 21 level of reliable, widely-available and easy-to-use automated tools.
     - E=Functional (F): functional exploitable code is available. The code works in most situations where the vulnerability exists. The code or technique is not functional in all situations and may require substantial modification by a skilled attacker.

  - E=Unproven (U): no exploitable code is available, or an exploit is strictly theoretical.
- Remediation Level (RL): The Remediation Level of a vulnerability is an important factor for prioritisation. The typical vulnerability is unpatched when it's initially published. Workarounds or hotfixes may offer interim remediation until an official patch or upgrade is introduced. The less official and permanent the fixes, the higher the vulnerability score.
  - RL=Not Defined (X): assigning this value to the metric will not influence the score. It is a signal to a scoring equation to skip this metric.
  - RL=Unavailable (U): there is either no solution available, or it is impossible to apply.
  - RL=Workaround (W): there is an unofficial, non-vendor solution available. In some cases, users of the affected technology will create a patch by their own or provide steps to work around or mitigate the vulnerability.
  - RL=Temporary Fix (T): there is an official but temporary fix available. This includes instances where the vendor issues a temporary hotfix, tool or workaround.
  - RL=Official Fix (O): a complete vendor solution is available. Either the vendor has issued an official patch, or an upgrade is available.
- Report Confidence (RC): This measures the degree of confidence of the vulnerability existence and the credibility of the known technical details:
  - RC=Not Defined (X): assigning this value to the metric will not influence the score. It is a signal to a scoring equation to skip this metric.
  - RC=Confirmed (C): detailed reports exist or functional reproduction is possible (functional exploits may provide this). The source code allows one to independently verify the assertions of the research or the author or vendor of the affected code has confirmed the presence of the vulnerability.
  - RC=Reasonable (R): significant details are published, but researchers either do not have full confidence in the root cause or do not have access to the source code to fully confirm all of the interactions that may lead to the result. Reasonable confidence exists, however, and the bug is reproducible and at least one impact can be verified (proof-of-concept exploits may provide this). An example is a detailed write-up of research into a vulnerability with an explanation (possibly obfuscated or *left as an exercise to the reader*). That provides assurances on how to reproduce the results.
  - RC=Unknown (U): there are reports claiming that a vulnerability is present. Reports indicate that the cause of the vulnerability is unknown, and they may differ in the cause or impacts of the vulnerability. There is

little confidence in the validity of the reports, and there is uncertainty in the static base score of the vulnerability. An example is a bug report which claims that an intermittent but non-reproducible crash occurs, with evidence of memory corruption suggesting a Denial of Service attack, or possibly more serious impacts.

4. Environmental Metrics: these metrics enable the analyst to customise the CVSS score depending on the importance of the affected IT asset of a user's organisation. They are measured in security controls complementary or alternative to basic ones (confidentiality, integrity and availability). These metrics are the modified equivalent of base metrics and have assigned values based on the component placement in an organisation's infrastructure.
   - Security Requirements (CR, IR, AR) are used to customise the CVSS score depending on the importance of the affected IT asset of a user's organisation, measured in terms of Confidentiality, Integrity and Availability.
     - Not Defined (X), assigning this value to the metric will not influence the score. It is a signal to the equation to skip this metric.
     - High (H), loss of confidentiality, integrity or availability is likely to have a catastrophic effect on the organisation or individuals associated with the organisation (e.g. employees, customers).
     - Medium (M), loss of confidentiality, integrity or availability is likely to have a serious effect on the organisation or individuals associated with the organisation (e.g. employees, customers).
     - Low (L), loss of confidentiality, integrity or availability is likely to have only a limited effect on the organisation or individuals associated with the organisation (e.g. employees, customers).
   - Modified Base Metrics enable the analyst to adjust the base metrics accordingly to modifications that exist within the analyst's environment:
     - Modified Attack Vector (MAV) contains the same values as the corresponding base metric, plus: Not Defined (the default); Modified Attack Complexity (MAC); Modified Privileges Required (MPR); Modified User Interaction (MUI); Modified Scope (MS); Modified Confidentiality (MC); Modified Integrity (MI); Modified Availability (MA).

Qualitative Severity Rating Scale maps values of the above ratings into a qualitative severity scale: None 0.0., Low 0.1–3.9, Medium 4.0–6.9, High 7.0–8.9, Critical 9.0–10.0.

The base metrics are assigned by an analyst, and the base equation computes a score ranging from 0.0 to 10.0. Firstly, the Impact Subscore (ISC_{Base}) has to be defined as:

$$ISC_{Base} = 1 - [(1 - C) * (1 - I) * (1 - A)] \tag{4.12}$$

The Exploitability Subscore is then given as follows:

$$Exploitability = 8.22 * AV * AC * PR * UI \tag{4.13}$$

4.4. COMMUNITY, INDUSTRY AND EXPERT SURVEYS

'Top 10' Threat Lists publish by organisations like NIST, CSA, CERT or ENISA are very valuable.

They are formulated based on the data from real systems and cover a lot of incidents. In the next subsections, only Cloud-related lists will be presented.

4.4.1. CLOUD SECURITY ALLIANCE Top Threats

This list is based on an industry expert survey and compiles opinions about the most dangerous security issues within Cloud Computing [29]. The following 12 critical issues (ranked in order of severity) were found:

1. Data Breaches: This is a situation when an unauthorised user get access to critical data, such as usernames and passwords or any information which shouldn't be given to the public. Data breaches can be caused by many factors, such as poor network security, application vulnerabilities or simply human error. The risk related to information disclosure is currently one of the top concerns of Cloud users and providers [29].
2. Weak Identity, Credential and Access Management: In this point, CSA pointed out that lack of multifactor authentication, lack of appropriate cryptographic key rotation, lack of appropriate certificates and password policies and unscalable identity management systems can lead to: spoofing identity attacks, tampering with data, repudiation, Denial of Service, information disclosure and elevation of privileges [29].
3. Insecure APIs: Security of Cloud services depend on basic API security. The following aspects should be taken into consideration: information management, data security, interoperability, portability, incidents response, applications security, encryption key management, encryption algorithms and access management [29]. Possible threats are: tampering with data, information disclosure, repudiation and elevation of privileges [29].
4. System and Application Vulnerabilities: System vulnerabilities are bugs which can be used by an attacker to steal data or take control over a machine. The most serious danger can be caused by bugs in critical parts of the infrastructure, such as the kernel [29]. Threats caused by these vulnerabilities are the same as mentioned in point 2 [29]. To prevent the system, the following security aspects must be taken into consideration: CC Architectural Framework,

business continuity possibility, traditional security, operations performed in data centres, appropriate risk management, disaster recovery, application security and virtualisation [29].

5. Account Hijacking: This threat involves taking control over a user's account by an attacker. In the CC context, this attacks can lead to eavesdropping of transitions, manipulating of data inside the Cloud, returning falsified information and redirecting the user to illegible sites [29]. Possible consequences are the same as mentioned in points 4 and 2. Prevention methods consider risk, information, identity and access management, data security, disaster recovery, business continuity, incident response, encryption and key management [29].
6. Malicious Insiders: A person (e.g. a current or former employee) which can use knowledge about the organisation and its structures to perform an attack. An insider is: *...responsible for most Cloud computing security woes* [29]. Possible threats related to malicious insiders are: spoofing identity, tampering with data and information disclosure [29]. To minimise these threats, CC providers should consider risk, information, encryption algorithms, encryption keys, identity and access management and data security aspects.
7. Advanced Persistent Threats (APTs): These kinds of threats are not oriented towards fast system paralysis. The main idea is that the malicious software will remain undetected as long as possible and will lead to information disclosure or elevation of privileges over time [29]. To avoid these kinds of threats, CC providers should consider: CC Architectural Framework, risk management, data recovery, business continuity, traditional security, operations performed in Data Centres, application security and virtualisation [29].
8. Data Loss: This can be caused by malicious software, as well as by human error or physical disaster. If the loss is permanent, the consequences for a business organisation may be devastating (repudiation or denial of service) [29]. To avoid data loss problems, the following aspects should be taken into consideration: risk, identity and access management, data security (disaster recovery, back-up policies), information management, application security and virtualisation [29].
9. Insufficient Due Diligence: This considers Cloud technologies and CC providers. The problem is quite general and touches upon commercial, technical, financial as well as legal issues. Diligence in every CC layer is a basis during developing new CC technologies or evaluating Cloud Providers. Lack of diligence can lead to all problems mentioned in this chapter [29]. To avoid these problems, all aspects mentioned in points 1–8 have to be taken into consideration, plus: legal aspects (contracts and electronic discovery), audit management and security as service framework usage [29].

10. Abuse and Nefarious Use of Cloud Services: The attacker may use delivered CC power as leverage for attacking purposes [29]. Sometimes it is simplified thanks to free trials of Cloud services and poorly secured CC systems. Possible threats are Distributed Denial of Service (DDoS) attacks, e-mail spam or phishing attacks [29]. To avoid these problems, the following aspects should be taken into consideration: traditional security, business continuity, disaster recovery, incident response and legal aspects [29].
11. Denial of Service: In this point, CSA focused on a DoS attack (a simplified version of DDoS mentioned in point 10). A classic DoS attack is oriented towards flooding a service with data and making it unstable or useless. In CC systems, an asymmetric application-level DoS attack may be even more dangerous than DDoS, because in some circumstances, it allows an attacker to take out an application with an extremely small payload (sometimes less than 100B) [29]. To protected the CC infrastructure, the following aspects should be taken into consideration: operations performed in Data Centres, incident response, application security, virtualisation and security as a service framework usage [29].
12. Shared Technology Issues: Underlying components of Cloud infrastructure (e.g. GPUs) may not provide enough isolation levels. This leads to a set of vulnerabilities, such as information disclosure or elevation of privileges [29]. A single vulnerability can compromise the whole Cloud system [29]. To prevent CC infrastructure, the following aspects should be taken into consideration: CC Architectural Framework, information, encryption, key, identity and access management, data security (defence in depth) and virtualisation.

4.4.2. OWASP CLOUD Top Ten

In this section, we will describe the Open Web Application Security Project (OWASP) top 10 risks related to Cloud users and providers. This list is formulated based on community reports, security experts and security incidences submitted by Cloud Providers [92]:

- Risk 1: Accountability and Data Risk: data stored locally is fully managed by an organisation. Responsibility for data security is also fully on the organisation's side (which has pluses and minuses). A decision about transferring data into the Cloud put this responsibility fully or partially on the Cloud Provider (backups, data recovery, etc.). This is a critical risk which should be carefully considered by the organisation or a private user [92].
- Risk 2: User Identity Federation: There must be full control over users' identities in the Cloud and between Cloud Providers if a particular user wants to transfer data. There should be a unified identification system which works across platforms and allows one to identify users uniquely [92].

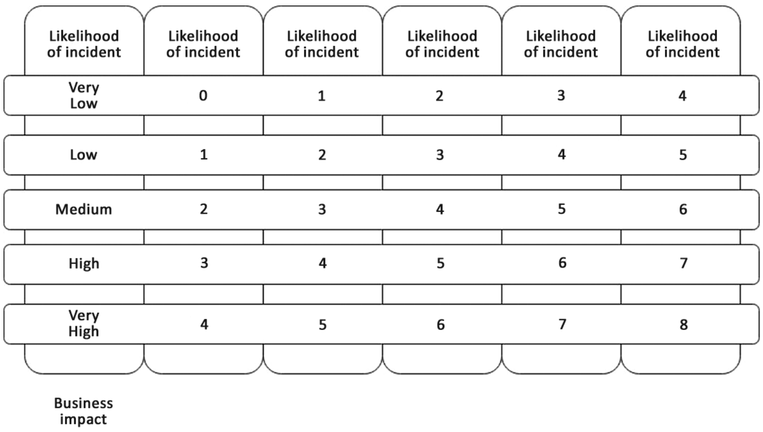
- Risk 3: Regulatory Compliance: Each country (or union) has its own law concerning data security and security system requirements. Security systems considered as secure in one country may not be perceived as secure in another [92], which may be a problem in naturally distributed CC systems.
- Risk 4: Business Continuity and Resiliency: In CC systems, business continuity aspects are delegated (fully or partially) to the Cloud Provider [92]. The business organisations depend on Cloud Provider competences and possibilities in case of disaster, as well as when it comes to the Quality of Service (QoS). Thus, appropriate Service Level Agreement (SLA) and other contractual solutions are mandatory [92].
- Risk 5: User Privacy and Secondary Usage of Data: Policies of web services concerning personal data usage are mostly imprecise [92]. In many cases, it is easy to obtain personal data directly or deduct it indirectly based on behaviour (e.g. via clicks) [92]. It should be clearly pointed out what data can be used for secondary purposes.
- Risk 6: Service and Data Integration: This point concerns the risk of capturing data during transfer over the Internet. This risk is much higher for CC users during transfer between a user and a Cloud Data Centre [92].
- Risk 7: Multi-tenancy and Physical Security: Multi-tenancy means sharing resources and services between multiple users [92]. Logical segregation should be done in a way that one user will not influence the security of the other users [92].
- Risk 8: Incidence Analysis and Forensics: After a security violation in the CC system, it may be hard to obtain all necessary data needed for investigation and forensic recovery [92]. This is caused by the distributed nature of CC systems. The Cloud Provider should ensure appropriate measures for incident analysis.
- Risk 9: Infrastructure Security: A Cloud system must be configured securely, which means: it should base on Industry Best Practices, applications should be configured with security zones, access must be configured to allow only required networks and protocols, administrative access must be role-based and based on need-to-know practice, risk assessment must be done [92]. Cloud Provider should also ensure security patches based on any appearing security issues [92].
- Risk 10: Non-production Environment Exposure: Non-production environments, such as development activities or testing activities, are very often not secured to the same extent as production environments [92]. This should be taken into consideration, especially when the non-production environment is in the Cloud, because it may lead to unauthorised access, information modification and information theft [92].

4.4.3. ENISA Risks List

The European Network and Information Security Agency (ENISA) in *Cloud Computing: Benefits, Risks and Recommendations for Information Security* [114] proposed another list of eight top security risks based on likelihood and the impact risk assessment process.

ENISA proposes formulating the level of risk based on the likelihood of an incident. The likelihood of an incident is defined by a threat exploiting a vulnerability with a given likelihood, see fig. 4.5. Such a measure defines the risk level as depending on the business impact and likelihood of the incident. The risk is measured on a scale from 0 to 8 and may be presented in numerical form as follows:

- Low risk: 0-2
- Medium Risk: 3-5
- High Risk: 6-8



Likelihood of incident	Likelihood of incident	Likelihood of incident	Likelihood of incident	Likelihood of incident	Likelihood of incident
Very Low	0	1	2	3	4
Low	1	2	3	4	5
Medium	2	3	4	5	6
High	3	4	5	6	7
Very High	4	5	6	7	8
Business impact					

Fig. 4.5. Risk level as a function of the business impact and likelihood of the incident scenario presented in [114]

ENISA's list takes into consideration the following aspects: Loss of governance: in Cloud infrastructures, it is necessary for the client to give control to the Cloud Provider (CP) over some issues that may affect security. At the same time, SLAs may not offer a commitment to provide such services by the Cloud Provider, thus leaving a gap in security defenses. This also includes compliance risks, because of investment in achieving certification (e.g., industry standard or regulatory requirements) may be put at risk by migration to the Cloud: if the CP cannot provide evidence of his own compliance with the relevant requirements, if the CP does not permit audit by the Cloud customer. In certain cases, it also means that using a public Cloud infrastructure implies that certain kinds of compliance cannot be achieved (e.g. PCI DSS).

Lock-in: there is still deficiency in tools, procedures or standard data formats, as well as service interfaces that could guarantee data, the application and service portability. This can make it difficult for the customer to migrate from one provider to another or migrate data and services back to an in-house IT environment. This introduces a dependency on a particular CP for service provision, especially if data portability, as the most fundamental aspect, is not enabled.

Isolation failure: multi-tenancy and shared resources define the characteristics of Cloud Computing. This risk category covers the failure of mechanisms separating storage, memory, routing and reputation between different tenants (e.g. so-called guest-hopping attacks). However, it should be noted that attacks on resource isolation mechanisms (e.g. against hypervisors) aren't very common, and they are much more difficult for an attacker to put into practice in comparison to attacks on traditional OSs.

Management interface compromise: customer management interfaces of a public Cloud Provider are accessible through the Internet and mediate in access to larger sets of resources (than traditional hosting providers). Therefore, these pose an increased risk, especially when combined with remote access and web browser vulnerabilities.

Data protection: Cloud Computing poses several data protection risks for Cloud customers and providers. In some cases, it may be difficult for the Cloud customer (in his role as a data controller) to effectively check the data handling practices of the Cloud Provider and thus to be sure that the data is handled law-fully. This problem is exacerbated in cases of multiple transfers of data, e.g. between federated Clouds. On the other hand, some Cloud providers do provide information on their data handling practices. Some also offer certification summaries on their data processing and data security activities and the data controls they have in place, e.g. SAS70 certification.

Insecure or incomplete data deletion: when a request to delete a resource in the Cloud is made, as in most operating systems, it may not result in a true wiping of the data. Adequate or timely data deletion may also be impossible (or undesirable from a customer perspective), either because extra copies of data are stored but are not available, or because the disk to be destroyed also stores data from other clients. In the case of multiple tenancies and the reuse of hardware resources, this represents a higher risk to the customer in comparison with dedicated hardware. Malicious insider: the damage which may be caused by malicious insiders is often far greater. Cloud architectures necessitate certain roles which are extremely high--risk. Examples include CP system administrators and managed security service providers. Customers security expectations: the perception of security levels by customers might differentiate from the actual security (and availability) offered by the CP or the actual temptation of the CP to reduce costs further by sacrificing some security aspects.

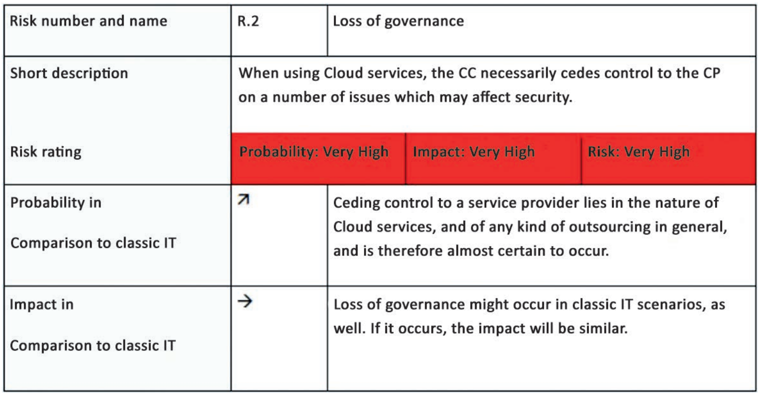
Availability Chain: reliance on Internet connectivity at the customers' end device creates a potential single point of failure.

4.4.4. NIST ISSUES AND CONCERNS LIST

The National Institute of Standards and Technology (NIST) in [14] proposed 3 lists of issues, each concerning a different type of Cloud environment.

The first list concerned the SaaS type and raised the following problems [14]:

- Browser-based Risks and Risk Remediation: In this point, NIST pointed that even appropriately secured client web browsers allow attackers to gain some important information. Message sizes, time of sending these messages and all metadata related to network traffic may lead to the leak of important information [14]. Another important fact is that not all client web browsers are secure enough. Potential vulnerabilities are implementation mistakes in cryptographic protocols, insecure cryptographic keys or passwords generation and man in the middle (MITM) attacks on cryptographic protocols (e.g. SSLStrip) [14]. Other risks come from the Cloud architecture and human imperfections: visiting malicious websites (phishing) and inaccurate services or data separation [14].



Risk number and name	R.2	Loss of governance	
Short description	When using Cloud services, the CC necessarily cedes control to the CP on a number of issues which may affect security.		
Risk rating	Probability: Very High	Impact: Very High	Risk: Very High
Probability in Comparison to classic IT	↗	Ceding control to a service provider lies in the nature of Cloud services, and of any kind of outsourcing in general, and is therefore almost certain to occur.	
Impact in Comparison to classic IT	→	Loss of governance might occur in classic IT scenarios, as well. If it occurs, the impact will be similar.	

Fig. 4.6. Impact of loss of governance and control on an organisation's strategy presented in [114]

- Network Dependence: An SaaS Cloud application depends on access to the Internet. The Internet is not controlled by either the customer or by the Cloud Provider, and thus potential availability problems may occur [14]. This risk can be reduced by dedicated and protected links (which demand additional costs) or an application structure which allows one to perform some operations offline [14].

- Lack of Portability between SaaS Clouds: The format of export and import of data to and from the Cloud may not be unified among SaaS providers [14]. Even unified formats may not concern some special categories of data, such as application settings or data extensions [14].
- Isolation vs. Efficiency: The more security measures, the more costs or the less system efficiency. In this point, NIST focuses on data and service isolation problems in SaaS systems. One approach is to launch an instance of the application for each client separately (separation is done by the server operating system), which is more secure but also more expensive (individual copies of an application and databases) [14]. The second approach assumes that a server uses a combined database to serve multiple customers simultaneously, which is more efficient but less secure [14].

The second list concerns the PaaS type, which shares all SaaS type issues mentioned above [14]. Three additional concerns were [14]:

- Lack of Portability between PaaS Clouds: Use of the PaaS Cloud environment for application development may involve usage of Cloud Provider specific interfaces. This reduces the portability of such application [14]. One possible solution which allows one to avoid this problem is the use of generalised interfaces. However, this involves additional costs and may reduce the possibility of use of specialised tools of specific Clouds [14].
- Event-based Processor Scheduling: Event-driven applications containing HTTP requests may be cost-effective, though they involve some additional constraints (e.g. response time interval, processing long-running requests) [14]. Because of this, tasks execution in a PaaS Cloud may exceed execution of the same set of tasks performed locally [14].
- Security Engineering of PaaS Applications: PaaS applications are more complex in comparison to applications executed in isolated environments, which makes them also harder to secure [14]. PaaS apps have to use explicit cryptography, interact with many browsers and use a set of web protocols and languages (such as XML, HTML or JavaScript), which increases the number of potential security exploits [14].

When it comes to the IaaS model, NIST pointed out six basic issues:

- Compatibility with Legacy Security Vulnerabilities: Users can use legacy software in IaaS Clouds. In such a case, Cloud users are exposed to all security vulnerabilities related to the outdated software [14].
- Virtual Machine Sprawl: Cloud users can create or maintain many Virtual Machines (VMs), and each of them may be in a different state. A VM which is currently suspended or off will not receive a software update, which may make it vulnerable [14]. Such a VM may be used as an entry point for an attacker.
- Verifying the Authenticity of an IaaS Cloud Provider Web Site: Even a perfectly secured IaaS Cloud will not help a customer when he connects

with the wrong IaaS website. Responsibility for a secure connection is on the client's (especially the client's browser) side. It is also the client's responsibility to verify the identity of the IaaS Cloud Provider [14].
- Robustness of VM-level Isolation: In the IaaS Cloud, customers can use resources from the same pool. Thus, appropriate isolation is crucial, at least to the point when clients want to interact with each other. To avoid problems related to inadequate isolation, such as eavesdropping or data tampering, Cloud Providers introduced an additional layer called: hypervisor [14]. The hypervisor layer allows one to split the physical machine (and its parts) into many VMs in a secure way [14]. This is called hardware virtualisation.
- Features for Dynamic Network Configuration for Providing Isolation: the Cloud network infrastructure also creates a pool which is used by a client's VMs. These configurations should allow one to communicate with client and external entities; however, it should also be isolated to prevent one from observing other client network packets [14]. What's more, this network should be easy and fast to configure (as fast as the VM creation time) [14]. This can be reached, for example, via appropriately configured VLANs (Virtual Local Area Networks).
- Data Erase Practices: This point concerns data erase, replication and backup practices. After the client releases some resources, data should be removed permanently so that another client using the same pool will not gain access to it [14]. The problem is that strong erase policies are time-consuming and not always compatible with high-performance solutions [14].

4.5. STRATEGIC MODELS

Strategy modelling is related to all methods that are able to find not the security threat model itself, but also the plan to defend resources under given conditions and uncertainty.

4.5.1. GAME THEORETIC MODELS

Yuzhe Li et al. [74] proposed using a non-zero sum Nash game for protecting wireless sensor networks serving for remote state estimation of the system. The threat that was prevented by using this procedure was one chosen type of threat, i.e. fake information substitution. On the contrary, a zero-sum was used in [35] for protecting a network of computers (that includes users and servers) against node-capturing and false signal transmission.

Nicola Basilico [17] considered the case of protecting a set of software modules by using a multi-stage two-player competitive game with a infinite horizon.

Jing Zhu [123] adopted a two-player Bayesian game to defend elastic optical networks against cross-domain physical-layer attacks.

Maria Pia Fanti [38] described how to defend Satellite Base Station networks with the use of stochastic game models. Several types of attacks were considered, for example, attack via HTTP, sniffing, virus, cracking root password and capturing data.

The Smart Grid was protected against a coalitional attack by using Iterated Public Goods Game (IPGG), as presented in [117]. Multiple adversaries were considered.

Ahmed H. Anwar et al. [10] successfully applied games for wireless network protection against stealthy decoy attacks that disrupt network operations by creating cascading channel conflicts.

Rui Zhang et al. [122] used a bi-level game theoretic model for Computer Network protection. The authors used a zero-sum game in a moral-hazard type of principal-agent game.

Table 4.1

Game Theoretic Models for attack modelling

Sol.	Player 1	Payoff 1 (minimised or maximised)	Payoff 2 (minimised or maximised)	Equilibrium	Type
[74]	sensor	sensor error	sensor error cost of attack	Nash	non-zero sum
[35]	network administrator	network bandwidth or	attacker cost and the defender cost	Nash	zero sum
[17]	software developer	the prob. of software update	the prob. of successful attack	Leader--Follower	non-zero sum
[123]	domain manager	harm of the attack	harm of the attack	Bayesian Nash	non-zero sum
[38]	network administrator	damages	importance of the data stolen	Nash	non-zero sum
[117]	grid administrator	gain from attack penalty when the attack is detected	importance of the data stolen	Zero--determinant	zero sum
[10]	wireless network administrator	the cost of a attack – the cost of defense	importance of the data stolen	Nash	zero sum
[122]	computer network administrator	risk	importance of the data stolen	Nash	zero sum

REFERENCES

[1] Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data.

[2] Cvss special interest group. common vulnerability scoring system v3.0: Specification document. First.org Inc., 2017.

[3] I. M. Abbadi. *Cloud Management and Security*. Wiley Publishing, 1st edition, 2014.

[4] N. M. AbdElnapi, F. A. Omara, and N. F. Omran. *A hybrid hashing security algorithm for data storage on cloud computing*. International Journal of Computer Science and Information Security, 14(4):175, 2016.

[5] A. Achuthshankar and A. Achuthshankar. *A novel symmetric cryptography algorithm for fast and secure encryption*. In 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO), pages 1–6, Jan 2015.

[6] S. Agrawal, S. Gorbunov, V. Vaikuntanathan, and H. Wee. *Functional encryption: New perspectives and lower bounds*. In Advances in Cryptology – CRYPTO 2013 – 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II, pages 500–518, 2013.

[7] B. Alese, A. Adetunmbi, O. Adewale, et al. *Elliptic curve cryptography for securing cloud computing applications*. International Journal of Computer Applications, 66(23):10–17, 2013.

[8] Amazon. *Networking and content delivery*. CloudFront documentation, 2018.

[9] Amazon. *Amazon ec2 key pairs*. Amazon Web Services documentation, 2019.

[10] A. H. Anwar, G. Atia, and M. Guirguis. *Game theoretic defense approach to wireless networks against stealthy decoy attacks*. In 2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pages 816–821, Sept 2016.

[11] AWS. *AWS Encryption SDK*. Amazon Web Services documentation, developer guide, 2019.

[12] AWS. *Cryptographic algorithms*. Amazon Web Services documentation, 2019.

[13] M. Babitha and K. R. Babu. *Secure cloud storage using AES encryption*. In 2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT), pages 859–864. IEEE, 2016.

[14] L. Badger, T. Grance, R. Patt-Corner, and J. Voas. Nist special publication 800--146: *Cloud computing synopsis and recommendations: Recommendations of the national institute of standards and technology*. National Institute of Standards and Technology, 2012.

[15] E. Barker. Nist special publication 800-57 part 1 rev. 4: *Recommendation for key management part 1: General*. National Institute of Standards and Technology, 2016.

[16] E. Barker, W. Barker, and A. Lee. Nist special publication 800-21: *Guideline for implementing cryptography in the federal government*. National Institute of Standards and Technology, 2005.

[17] N. Basilico, A. Lanzi, and M. Monga. *A security game model for remote software protection*. In 2016 11th International Conference on Availability, Reliability and Security (ARES), pages 437–443, Aug 2016.

[18] D. Beaver. *Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority*. Journal of Cryptology, 4(2):75–122, Jan 1991.

[19] G. Blakley and C. Meadows. *Security of ramp schemes*. Volume 196, pages 242––268, 01 1984.

[20] W. Bogorad, S. Schneider, and H. Zhang. *Norton zone: Symantec's secure cloud storage system*. In 2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS), pages 81–90, Sept 2016.

[21] D. Boruah and M. Saikia. *Implementation of elgamal elliptic curve cryptography over prime field using c*. In International Conference on Information Communication and Embedded Systems (ICICES2014), pages 1–7. IEEE, 2014.

[22] C. Brenton. *The basics of virtualization security*. Cloud Security Alliance, 2011.

[23] A. Castiglione, F. Palmieri, C.-L. Chen, and Y.-C. Chang. *A blind signature-based approach for cross-domain authentication in the cloud environment*. International Journal of Data Warehousing and Mining (IJDWM), 12(1): 34–48, 2016.

[24] Certicom. *Ecc tutorial*. Certicom Corp. a subsidiary of BlackBerry, 2016.

[25] D. Chaum. *Blind signatures for untraceable payments*. In Advances in cryptology, pages 199–203. Springer, 1983.

[26] B. Chevallier-Mames, P. Paillier, and D. Pointcheval. *Encoding-free elgamal encryption without random oracles*. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography – PKC 2006*, pages 91–104, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[27] R. Cramer, I. Damgrd, and Y. Ishai. *Share conversion, pseudorandom secret-sharing and applications to secure computation*. Volume 3378, pages 342–362, 02 2005.

[28] CSA. *Cloud security alliance releases (secaas) implementation guidance*. Cloud Security Alliance, 2012.

[29] CSA. *The treacherous 12: Cloud computing top threats in 2016*. Cloud Security Alliance, 2016.

[30] P. Czernik. *Cryptographically secure pseudorandom number generators in low power distributed measurement and control systems*. Transactions of the Institute of Aviation, 6(201):5–19, 2009.

[31] C. A. B. de Carvalho, M. F. De Castro, and R. M. de Castro Andrade. *Secure cloud storage service for detection of security violations*. In Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pages 715–718. IEEE Press, 2017.

[32] M. J. Dworkin. Nist special publication 800-38a: *Recommendation for block cipher modes of operation, methods and techniques*. National Institute of Standards and Technology, 2001.

[33] M. J. Dworkin. Federal inf. process. stds. (nist fips) – 202: *Sha-3 standard: Permutation-based hash and extendable-output functions*. National Institute of Standards and Technology, 2015.

[34] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, and J. F. Dray Jr. Federal inf. process. stds. (nist fips) – 197: *Advanced encryption standard (AES)*. National Institute of Standards and Technology, 2001.

[35] E. Eisenstadt and A. Moshaiov. *Novel solution approach for multi-objective attack-defense cyber games with unknown utilities of the opponent*. IEEE Transactions on Emerging Topics in Computational Intelligence, 1(1):16–26, Feb 2017.

[36] N. E. El-Attar, W. A. Awad, and F. A. Omara. *Empirical assessment for security risk and availability in public cloud frameworks*. In 2016 11th International Conference on Computer Engineering Systems (ICCES), pages 17–25, Dec 2016.

[37] K. Eldefrawy, S. Faber, and T. Kaczmarek. *Proactively secure cloud-enabled storage*. In 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pages 1499–1509, June 2017.

[38] M. P. Fanti, M. Nolich, S. Simi, and W. Ukovich. *Modeling cyber attacks by stochastic games and timed petri nets*. In 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pages 2960–2965, Oct 2016.

[39] D. George Amalarethinam and H. Leena. *Enhanced RSA algorithm for data security in cloud*. Int. J. Control Theor. Appl., 9:147–152, 2016.

[40] Google. *Key purposes and algorithms*. Google Cloud documentation, 2018.

[41] Google. *Encrypting and decrypting data with an asymmetric key*. Google Cloud documentation, 2019.

[42] Google. *Encryption at rest in google cloud platform*. Google Cloud documentation, 2019.

[43] D. Gordon. *Discrete logarithm problem*. In H. C. A. van Tilborg and S. Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 352–353, Boston, MA, 2011. Springer US.

[44] N. C. C. S. W. Group. Nist special publication 500-299: *Nist cloud computing security reference architecture*. National Institute of Standards and Technology, 2013.

[45] K. K. Hingwe and S. M. S. Bhanu. *Two layered protection for sensitive data in cloud*. In 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pages 1265–1272. IEEE, 2014.

[46] R. Holland. *The CISOs Guide To Virtualization Security*. Forrester, Inc., 2012.

[47] IBM. *IBM marketplace*, 2019.

[48] A. Ibrahim, W. Cheruiyot, and M. W. Kimwele. *Data security in cloud computing with elliptic curve cryptography*. International Journal of Computer (IJC), 26: 1–14, 06 2017.

[49] ICO. *Key definitions of the Data Protection Act*. Information Commissioner's Office, 2017.

[50] ISO. ISO/IEC 19790:2012 Information technology – Security techniques – Security requirements for cryptographic modules. International Organization for Standardization, 2012.

[51] ISO. ISO/IEC 29192-2:2012 Information technology – Security techniques – Lightweight cryptography. International Organization for Standardization, 2012.

[52] ISO. ISO/IEC 20008-1:2013. International Organization for Standardization, 2013.

[53] ISO. ISO/IEC 19592-2:2017. International Organization for Standardization, 2017.

[54] T. Jaeger, R. Sailer, and Y. Sreenivasan. *Managing the risk of covert information flows in virtual machine systems*. In Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, SACMAT '07, pages 81–90, New York, NY, USA, 2007. ACM.

[55] A. Jakóbik, D. Grzonka, J. Kołodziej, and H. González-Vélez. *Towards secure non-deterministic meta-scheduling for clouds*. In 30th European Conference on Modelling and Simulation, ECMS 2016, Regensburg, Germany, May 31 – June 3, 2016, Proceedings., pages 596–602, 2016.

[56] A. Jakóbik, D. Grzonka, and F. Palmieri. *Non-deterministic security driven meta scheduler for distributed cloud organizations*. Simulation Modelling Practice and Theory, 76:67–81, 2017.

[57] N. Jayapandian, A. M. Z. Rahman, and I. Nandhini. *A novel approach for handling sensitive data with deduplication method in hybrid cloud*. In 2015 Online International Conference on Green Engineering and Technologies (IC-GET), pages 1–6. IEEE, 2015.

[58] M. Jebalia, A. B. Letafa, M. Hamdi, and S. Tabbane. *A secure data storage based on revocation game-theoretic approaches in cloud computing environments*. In 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), pages 435–440, June 2017.

[59] K. Jensen and L. M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[60] P. Johnson, A. Vernotte, M. Ekstedt, and R. Lagerstrm. pwnpr3d: *An attack--graph-driven probabilistic threat-modeling approach*. In 2016 11th International Conference on Availability, Reliability and Security (ARES), pages 278–283, Aug 2016.

[61] N. Kaaniche. *Cloud data storage security based on cryptographic mechanisms*. PhD Thesis, TELECOM SUDPARIS et L'UNIVERSIT PIERRE ET MARIE CURIE, 2015.

[62] X. Kaiqi. *Resource Optimization and Security for Cloud Services*. IT Governance Publishing, 1st edition, 2012.

[63] B. Kaliski. *RSA digital signature scheme*. In H. C. A. van Tilborg and S. Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 1061–1064, Boston, MA, 2011. Springer US.

[64] D. Karras. *On scalable and efficient security risk modelling of cloud computing infrastructure based on Markov processes*. ITM Web of Conferences, 9:03006, 01 2017.

[65] J. Katz, A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[66] N. Kaur and H. Singh. *Efficient and secure data storage in cloud computing through blowfish, RSA and hash function*. International Journal of Science and Research (IJSR), 4(5), 2012.

[67] R. Kaur and A. Kaur. *Digital signature*. In 2012 International Conference on Computing Sciences, pages 295–301. IEEE, 2012.

[68] R. L. Kissel. *Nist interagency/internal report (nistir) – 7298 rev. 2: Glossary of key information security term*. National Institute of Standards and Technology, 2013.

[69] J. Kołodziej. *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems*. Springer Publishing Company, Incorporated, 2012.

[70] R. Kumar and M. Stoelinga. *Quantitative security and safety analysis with attack--fault trees*. In 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE), pages 25–32, Jan 2017.

[71] S. T. Lai and F. Y. Leu. *A security threats measurement model for reducing cloud computing security risk*. In 2015 9th International Conference on Innovative

Mobile and Internet Services in Ubiquitous Computing, pages 414–419, July 2015.

[72] D. Lalar and R. Nahta. *Stream cipher*. International Journal of Advanced Research in Computer Science, 7(7), 2016.

[73] B.-H. Lee, E. K. Dewi, and M. F. Wajdi. *Data security in cloud computing using AES under heroku cloud*. In 2018 27th Wireless and Optical Communication Conference (WOCC), pages 1–5. IEEE, 2018.

[74] Y. Li, D. E. Quevedo, S. Dey, and L. Shi. *A game-theoretic approach to fake--acknowledgment attack on cyber-physical systems*. IEEE Transactions on Signal and In formation Processing over Networks, 3(1):1–11, March 2017.

[75] X. Lin, P. Zavarsky, R. Ruhl, and D. Lindskog. *Threat modeling for CSRF attacks*. In 2009 International Conference on Computational Science and Engineering. Volume 3, pages 486–491, Aug 2009.

[76] X. Liu and Z. Liu. *Evaluating method of security threat based on attacking-path graph model*. In 2008 International Conference on Computer Science and Software Engineering. Volume 3, pages 1127–1132, Dec 2008.

[77] N. Lynch, R. Segala, and F. Vaandrager. *Hybrid i/o automata*. Inf. Comput., 185(1):105–157, Aug 2003.

[78] P. K. Manadhata and J. M. Wing. *An attack surface metric*. IEEE Transactions on Software Engineering, 37(3):371–386, May 2011.

[79] K. Mandeep and M. Manish. *Implementing various encryption algorithms to enhance the data security of cloud in cloud computing*. VSRD International Journal of Computer Science & Information Technology, 2(10):831–835, 2012.

[80] P. Mell and T. Grance. Nist special publication 800-145: *Nist definition of cloud computing*. National Institute of Standards and Technology, 2011.

[81] S. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

[82] Microsoft. *Data encryption in onedrive for business and sharepoint online*. Microsoft OneDrive documentation, 2018.

[83] Microsoft. *Ecdsa class*. Microsoft Azure documentation, 2018.

[84] F. Minfeng and C. Wei. *Elliptic curve cryptosystem elgamal encryption and transmission scheme*. In 2010 International Conference on Computer Application and System Modeling (ICCASM 2010). Volume 6, pages V6–51. IEEE, 2010.

[85] P. Mistry. *OpenStack Barbican, Cryptography for Managing Secrets in the Cloud*. The New Stack, 2014.

[86] M. Mohsin and Z. Anwar. *Where to kill the cyber kill-chain: An ontology-driven framework for IOT security analytics*. In 2016 International Conference on Frontiers of Information Technology (FIT), pages 23–28, Dec 2016.

[87] W. C. Moody, H. Hu, and A. Apon. *Defensive maneuver cyber platform modeling with stochastic petri nets*. In 10th IEEE International Conference

on Collaborative Computing: Networking, Applications and Worksharing, pages 531–538, Oct 2014.

[88] L. Newcombe. *Securing Cloud Services – A pragmatic approach to security architecture in the Cloud*. IT Governance Publishing, 1st edition, 2012.

[89] NIST. Federal inf. process. stds. (nist fips) 186-4: *Digital signature standard (dss)*. National Institute of Standards and Technology, 2013.

[90] NIST. Federal inf. process. stds. (nist fips) 180-4: *Secure hash standard (shs)*. National Institute of Standards and Technology, 2015.

[91] A. Omotunde, F. Adekogbe, E. Onuiri, and P. Uchendu. *An implementation of a one-time pad encryption algorithm for data security in cloud computing environment*. Research Journal of Mathematics and Computer Science, 1:1–8, 12 2017.

[92] OWASP. *Owasp cloud top ten project*. Open Web Application Security Project, 2014.

[93] N. Padmaja and P. Koduru. *Providing data security in cloud computing using public key cryptography*. International Journal of Engineering Sciences Research, 4(1):1059–1063, 2013.

[94] I. Petri, J. Diaz-Montes, M. Zou, O. F. Rana, T. Beach, H. Li, and Y. Rezgui. *In-transit data analysis and distribution in a multi-cloud environment using cometcloud*. In 2014 International Conference on Future Internet of Things and Cloud, pages 471–476, Aug 2014.

[95] L. B. A. Rabai, M. Jouini, M. Nafati, A. B. Aissa, and A. Mili. *An economic model of security threats for cloud computing systems*. In Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), pages 100–105, June 2012.

[96] D. S. Raghuwanshi and M. R. Rajagopalan. *Ms2: Practical data privacy and security framework for data at rest in cloud*. In 2014 World Congress on Computer Applications and Information Systems (WCCAIS), pages 1–8, Jan 2014.

[97] W. Reisig. *Petri nets and algebraic specifications*. Theoretical Computer Science, 80(1):1–34, 1991.

[98] S. Delfin, Rachana Sai. B., Meghana J.V., Kundana Lakshmi. Y., and Sushmita Sharma. *Cloud data security using AES algorithm*. International Research Journal of Engineering and Technology (IRJET), 5(10):1189–1192, 2018.

[99] C. Saravanakumar and C. Arun. *Survey on interoperability, security, trust, privacy standardization of cloud computing*. In 2014 International Conference on Contemporary Computing and Informatics (IC3I), pages 977–982, Nov 2014.

[100] S. Sathish, D. Sumathi, and P. Sivaprakash. *Security services using ECDSA in cloud computing*. International Journal of Advanced Research in Computer Science and Software Engineering, IJARCSSE, 4(5), 2014.

[101] A. Shamir. *How to share a secret*. Communications of the ACM, 22(11): 612–613, 1979.

[102] A. P. U. Siahaan. *Securing short message service using vernam cipher in android operating system*. INA-Rxiv, 2017.

[103] V. Sidorov and W. K. Ng. *Transparent data encryption for data-in-use and data-at-rest in a cloud-based database-as-a-service solution*. In 2015 IEEE World Congress on Services, pages 221–228. IEEE, 2015.

[104] S. Singh, P. Sharma, and D. Arora. *Data integrity check in cloud computing using hash function*. International Journal of Advanced Research in Computer Science, 8(5), 2017.

[105] U. Somani, K. Lakhani, and M. Mundra. *Implementing digital signature with RSA encryption algorithm to enhance the data security of cloud in cloud computing*. In 2010 First International Conference On Parallel, Distributed and Grid Computing (PDGC 2010), pages 211–216. IEEE, 2010.

[106] S. Son, D. J. Kang, and J. M. Kim. *Design considerations to realize automated sla negotiations in a multi-cloud brokerage system*. In 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, pages 466–468, Oct 2014.

[107] M. Souppaya, K. Scarfone, and P. Hoffman. Nist special publication 800--125: *Guide to security for full virtualization technologies*. National Institute of Standards and Technology, 2011.

[108] W. Stallings. *Kryptografia i bezpieczeństwo sieci komputerowych. Matematyka szyfrów i techniki kryptologii*. Helion, 5th edition, 2012.

[109] A. Stiglic. *Safe prime*. In H. C. A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*, pages 541–541, Boston, MA, 2005. Springer US.

[110] A. Stiglic. *Strong prime*. In H. C. A. van Tilborg and S. Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 1265–1266, Boston, MA, 2011. Springer US.

[111] G. Stoneburner, A. Goguen, and A. Feringa. Nist special publication 800-30: *Risk management guide for information technology systems*. National Institute of Standards and Technology, 2002.

[112] M. Szpyrka and B. Jasiul. *Evaluation of cyber security and modelling of risk propagation with petri nets*. Symmetry, 9(3), 2017.

[113] J. Tchórzewski and A. Jakóbik. *Theoretical and experimental analysis of cryptographic hash functions*. Journal of Telecommunications and Information Technology, (1): 125–133, 2019.

[114] H. Thomas and D. Lionel. *Cloud computing: Benefits, risks and recommendations for information security*, rev. b. European Network and Information Security Agency, 2012.

[115] J. Wu, C. Liu, J. Ma, Y. Cheng, J. Ren, and Z. Wang. *A case for the cloud storage system supporting sensitive data application*. In IEEE Conference Anthology, pages 1–4. IEEE, 2013.

[116] A. S. C. *X9. American national standard x9.62-1998 public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA)*. American National Standards Institute, 1998.

[117] X. Yang, X. He, J. Lin, W. Yu, and Q. Yang. *A game-theoretic model on coalitional attacks in smart grid*. In 2016 IEEE Trustcom/BigDataSE/ISPA, pages 435––442, Aug 2016.

[118] A. A. Yassin, H. Jin, A. Ibrahim, W. Qiang, and D. Zou. *Cloud authentication based on anonymous one-time password*. In Ubiquitous Information Technologies and Applications, pages 423–431. Springer, 2013.

[119] P. Yellamma, C. Narasimham, and V. Sreenivas. *Data security in cloud using RSA*. In 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), pages 1–6, July 2013.

[120] J. Yu and H. Wang. *Strong key-exposure resilient auditing for secure cloud storage*. IEEE Transactions on Information Forensics and Security, 12(8): 1931–1940, Aug 2017.

[121] J. Zhang, Y. Yang, Y. Chen, and F. Chen. *A secure cloud storage system based on discrete logarithm problem*. In 2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS), pages 1–10, June 2017.

[122] R. Zhang, Q. Zhu, and Y. Hayel. *A bi-level game approach to attack-aware cyber insurance of computer networks*. IEEE Journal on Selected Areas in Communications, 35(3):779–794, March 2017.

[123] J. Zhu, B. Zhao, and Z. Zhu. *Leveraging game theory to achieve efficient attack-aware service provisioning in eons*. Journal of Lightwave Technology, 35(10):1785–1796, May 2017.

[124] S. Y. Zhu, R. Hill, and M. Trovati. *Guide to security assurance for cloud computing*. Springer, 2016.

[125] *Zscaler Inc. Stopping zero day threats*. SlideShare, 2016.

INDEX

eISBN 978-83-67188-62-3

Cracow University
of Technology