# Contribuţii la îmbunătăţirea dependabilităţii si securităţii informaţiei

Teza de doctorat

Rodica Ţirtea

Coordonator ştiinţific: prof. dr. ing. Mircea Vlăduţiu
Universitatea Politehnica Timişoara

Preşedinte: Prof. dr. ing. Proştean Octavian
Conducător de doctorat: Prof. dr. ing. Vlăduţiu Mircea
Membrii:
1. Prof. dr. ing. Petrescu Mircea, Universitatea „Politehnica” din Bucureşti
2. Prof. dr. ing. Deconinck Geert, Universitatea Catolica Leuven din Belgia
3. Prof. dr. ing. Mang Ioan, Universitatea din Oradea.

# Dependability and information security enhancements

Ph.D. thesis
Rodica Ţirtea

# CUVÂNT ÎNAINTE

Această teza de doctorat a fost elaborată sub coordonarea d-lui profesor dr. ing. Mircea Vlăduţiu din cadrul Universităţii „Politehnica” din Timişoara. Doresc să-i mulţumesc şi pe această cale conducătorului de doctorat, prof. dr. ing. Mircea Vlăduţiu, pentru sugestiile şi încurajările deosebit de utile în munca de cercetare dar si pentru promptitudinea cu care a răspuns întotdeauna. Pe parcursul anilor care au trecut de când m-am înscris la doctorat, deşi o buna perioadă mi-am desfăşurat activitatea de studiu si cercetare în străinătate, discuţiile profesionale, coordonarea şi îndrumarea nu s-au diminuat.

Doresc de asemenea să mulţumesc membrilor comisiei de doctorat, referenţilor, pentru că au acceptat sa facă parte din comisia de doctorat. Le mulţumesc d-lui academician prof. dr. ing. Mircea Petrescu, d-lui prof. dr. ing. Ioan Mang de la Universitatea din Oradea şi nu în ultimul rând d-lui prof. dr. ing. Geert Deconinck căruia ţin sa-i mulţumesc şi pe această cale pentru colaborarea foarte bune pe care am avut-o pe durata studiilor mele de doctorat în K.U.Leuven, Belgia.

Aş dori de asemenea să mulţumesc membrilor colectivului din Catedra de Calculatoare a Facultăţii de Automatică şi Calculatoare a Universităţii „Politehnica” din Timişoara pentru sugestiile utile pe care le-au exprimat cu ocazia susţinerii examenelor sau referatelor.

Le mulţumesc de asemenea profesorilor mei din Universitatea din Oradea, colegilor din Catedra de Calculatoare, pentru îndrumare şi sprijin.

Aceasta teză nu ar fi fost posibilă fără susţinerea permanentă şi necondiţionată a părinţilor mei. Le mulţumesc şi pe această cale pentru răbdarea de care au dat dovadă în toţi aceşti ani dedicaţi studiului.

Oradea, ianuarie 2007 Rodica Ţirtea

Părinţilor mei.

Ţirtea, Rodica
**Contribuţii la îmbunătăţirea dependabilităţii si securităţii informaţiei**
Teze de doctorat ale UPT, Seria 1, Nr. 23, Editura Politehnica, 2006, 104 pagini, 33 figuri, 17 tabele.
ISSN:1224-6069
Cuvinte cheie:
dependabilitate, toleranta la defecte, securitatea informatiei, algoritmi criptografici
Rezumat,
Conform ultimelor cercetări conceptele de dependabilitate şi securitate nu mai pot fi separate chiar dacă până recent s-au dezvoltat independent. Aceasta lucrare adresează nevoia de securitate în sistemele cu cerinţe de dependabilitate, şi în acelaşi timp, nevoia de dependabilitate a mecanismelor de securitate implementate. Dependenţele dintre securitate şi dependabilitate sunt identificate şi sunt introduse contribuţii în domeniul implementărilor criptografice tolerante la defecte. Toleranţa la defecte este necesară în implementările criptografice pentru a preveni noile tipuri de atacuri şi vulnerabilităţi (atacuri bazate pe analiza erorilor). Data fiind gama largă de algoritmi criptografici, în această lucrare este analizat cazul cifrurilor bloc (algoritmi care asigură confidenţialitatea). In această lucrare tehnici noi de detectare a erorilor sunt propuse, analizate şi evaluate pentru cifruri bloc. Rezultatele obţinute în urma simulărilor hardware sunt comparate cu cele ale cercetărilor anterioare. Aceasta lucrare aduce noi argumente care susţin dependenţa dintre dependabilitate şi securitate şi contribuie, cu tehnici noi, avantajoase din punct de vedere al costului, la asigurarea dependabilităţii implementărilor criptografice.

# TABLE OF CONTENT

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| 3GPP | 3rd Generation Partnership Project |
| AES | Advanced Encryption Standard |
| ATM | Asynchronous Transfer Mode |
| CBC | Cipher Block Chaining |
| CD | Complemented Duplication error detection |
| CED | Concurrent Error Detection |
| CFB | Cipher FeedBack |
| CRYPTREC | CRYPTography Research and Evaluation Committees |
| CTR | Counter Mode |
| DES | Data Encryption Standard |
| DFA | Differential Fault Analysis |
| ECB | Electronic Code Block |
| FIPS | Federal Information Processing Standards |
| FPGA | Field-Programmable Gate Array |
| GSM | Global System for Mobile Communications |
| IETF | Internet Engineering Task Force |
| IPsec | Internet Protocol security |
| ISO | International Organization for Standards |
| ITUA | Intrusion Tolerance by Unpredictable Adaptation |
| LFSR | Linear Feedback Shift Register |
| LUTs | Look-Up Tables |
| MAC | Message Authentication Code |
| MAFTIA | Malicious and Accidental Fault Tolerance for Internet Applications |
| NDFA | Non-Differential Fault Analysis |
| NESSIE | New European Schemes for Signature, Integrity and Encryption |
| NIST | National Institute of Standards and Technology |
| OFB | Output FeedBack |
| TLS | Transport Layer Security |

# REZUMAT

Dezvoltarea tehnologiei informaţiei, a sistemelor de calcul şi a aplicaţiilor lor, au determinat şi continuă să determine expansiunea unor domenii cu o istorie lungă. *Securitatea informaţiei*, mii de ani utilizată pentru secretizarea comunicaţiilor din domeniul militar sau diplomatic, este influenţată astăzi de evoluţia tehnologiei informaţiei. Proliferarea calculatoarelor şi a informaţiei in format digital determină dezvoltarea unor tehnici şi concepte noi care sa răspundă cerinţelor cunoscute şi utilizate de secole (concepte cum ar fi semnătura digitală, autentificarea sursei, etc. sunt dezvoltate pentru a extinde conceputul clasic de semnătura de pe pergament sau hârtie). Mai mult, dată fiind tendinţa spre o societate informatizată, alte domenii, cum ar fi telecomunicaţiile civile, domeniul medical, etc. au nevoie de astfel de tehnici.

Un factor important în dezvoltarea şi extinderea aplicaţiilor care beneficiază de tehnologia informaţiei este încrederea. Pentru a dezvolta sisteme sigure şi fiabile în care sa putem avea justificat încredere este nevoie ca aceste sisteme sa răspundă cerinţelor de *dependabilitate* si *securitate*. Altfel, ignorând aceste cerinţe, consecinţele pot fi grave (pierderi materiale şi umane etc.) periclitând inclusiv încrederea societăţii în noile tehnologii.

Conform ultimelor cercetări, conceptele de dependabilitate şi securitate nu mai pot fi separate, chiar dacă până recent s-au dezvoltat independent. Securitatea informaţiei, prin câteva din obiectivele ei (integritatea datelor, autentificarea părţilor sau mesajelor etc.) aparţine conceptului de dependabilitate, în timp ce conceptul de securitate utilizează mijloacele dependabilităţii (toleranta la defecte, prognosticarea defectelor) pentru a combate defectele intenţionate şi vulnerabilităţile. Având în vedere acest context, această lucrarea se concentrează pe dependenţa care există între dependabilitate şi securitate.

Aceasta teza adresează nevoia de securitate în sistemele cu cerinţe de dependabilitate, şi în acelaşi timp, nevoia de dependabilitate a mecanismelor de securitate implementate. Deoarece aceste domenii continuă să se extindă, sunt incluse analize ale stadiului actual de dezvoltare. Dependenţele dintre securitate şi dependabilitate sunt identificate şi sunt introduse contribuţii în domeniul implementărilor criptografice tolerante la defecte. Toleranţa la defecte este necesară în implementările criptografice pentru a preveni noile tipuri de atacuri şi vulnerabilităţi (atacuri bazate pe analiza erorilor). Data fiind gama largă de algoritmi criptografici, în această lucrare este analizat cazul cifrurile bloc (algoritmi care asigură confidenţialitatea). In această lucrare tehnici noi de detectare a erorilor sunt propuse, analizate şi evaluate pentru cifruri bloc. Rezultatele obţinute în urma simulărilor hardware sunt comparate cu cele ale cercetărilor anterioare. Aceasta lucrare aduce noi argumente care susţin dependenţa dintre dependabilitate şi securitate şi contribuie, cu tehnici noi, avantajoase, la asigurarea dependabilităţii implementărilor criptografice.

# ABSTRACT

Since long, advances in computer technologies and networks determine developments in other fields. *Information security*, with its roots thousands of years ago in the need for secrecy associated to diplomacy and military, benefits nowadays from these advances. The widespread use of computers and information in digital form, demands new techniques and concepts, equivalent to long time known and used ones (e.g. digital signature, non-repudiation, data origin authentication are information security objectives relying on signature concept). Nevertheless, not only diplomacy and military are using information security today - security products are developed and deployed to answer security needs in other public services in an information intensive society (e.g. telecommunications, banking, health care).

However, trust is an important factor in deployment of new technologies and services in general and computer technologies in particular in an information society. Without consideration of *dependability* and *security* aspects in the design and development of new products, severe consequences may occur (accidents, security attacks etc.), besides jeopardizing public trust. Dependability and security through their means and objectives supply methods and techniques to develop systems and deliver services in which we can justifiably trust.

Recent research advocates that security and dependability cannot be separated anymore, even if the two fields developed separately. Information security through its objectives (data integrity, entity and message authentication, etc.) is part of the dependability concept, while security uses dependability means (e.g. fault tolerance, fault forecasting) to address malicious faults. Starting from this context, this work addresses the dependencies between security and dependability.

This work covers both security and dependability fields concentrating on the need for security in dependable systems and on the need for dependability of security mechanisms implementations. Due to extensive research and permanent evolution in the fields, state-of-the-art surveys are included. The dependencies between security and dependability are identified and contributions are introduced in the topic of fault-tolerant cryptographic implementations. Fault tolerance is required in security implementations to overcome the new types of vulnerabilities and attacks (e.g. fault analysis attacks). Given the wide area of cryptographic primitives, this work focuses on confidentiality algorithms, specifically block ciphers. New error detection techniques are proposed, applied and evaluated in this thesis for block ciphers. The results of hardware implementations and simulations are compared with the ones of previous proposed solutions. The findings of this work bring new arguments supporting the dependencies between security and dependability and contribute with new, low cost techniques for dependable security implementations.

# 1. INTRODUCTION

## 1.1. Context of the work

Critical and non-critical applications rely increasingly on information technologies for their operation. From telephony, or on-board computers, navigation systems and other technologies in today's automobiles to control systems in power plants or flight systems, information technology is part of our lives more and more every day. The introduction of information technology, especially for general public relies on the *trust* that the delivered services meet requirements for safety, reliability, etc. Trust is defined as the *expectation* that a service will be provided or a commitment will be fulfilled [1]. The engineering aspects related to trust are covering issues such as dependability and security.

The *dependability* of information technologies has to be address to secure correct and continuous services delivery. *Dependability* is that property of a computer system such that reliance can justifiably be placed on the service it delivers [2].

Besides the requirements for dependability of the systems/applications using information technologies, the requirements for *security* have to be addressed. The market pressure to deploy low cost solutions, the move from centralized to distributed solutions and from wired to wireless connectivity generate new type of challenges in order to meet increasingly stringent privacy and security requirements. *Information security* has as objectives, between others, privacy or confidentiality, entity and data authentication, data integrity, non-repudiation [3].

Security is progressively more needed in different fields. Computing applications deployed for instance in financial systems or health care systems have to address by default requirements for confidentiality, data and entity authentication, data integrity, non-repudiation, etc. in order to deliver their services. However, nowadays, other applications relying on distributed systems, for instance deployed in infrastructures (e.g. electric power infrastructure), take advantage of the widespread use of Internet and mobile communications to deliver their services [4]. In this case the communication between different components of the applications/systems has to address the challenges of a public environment which can become hostile (e.g. malicious attempts to jeopardize the security of the services can be encountered). From these examples can be noticed that security is needed not only in the fields where traditionally secrecy was required, but also in other fields where dependability and survivability of services are mandatory. As such security is indispensable in order to deliver dependable services.

The two concepts, dependability and security, are used together lately [5] to express requirements for reliable, available, safe, secure etc. services delivered by computing systems. We illustrated that security is used to deliver dependable services. Nevertheless, dependability is needed to assure correct, fault-free security implementations. Hence the requirements for dependability and security cannot be separated anymore.

In this work the focus is on security as a means to address dependability requirements. This objective is achieved by identifying vulnerabilities and proposing

solutions to diminish the impact of malicious attempts on the dependability of computing systems. Furthermore, because dependable systems rely on security techniques (e.g. implemented algorithms) the reliability of cryptographic implementations is another issue addressed in this thesis. Hence the dependencies between dependability and security are covered in this work. However this work has no pretension to cover all aspects of these dependencies.

## 1.2. Contributions

This thesis addresses the relationship between dependability and security, and motivates the need for security in dependable systems. However, cryptographic means (e.g. algorithms, protocols) are used to implement security systems. As such, these implementations need to be reliable. Thus the reliability of cryptographic systems facing new type of implementation related attacks is addressed. Starting from the state-of-the-art research on security and dependability, in this work both theoretical and practical contributions are introduced. A list of the contributions follows:

- state-of-the-art investigation of
  - security and cryptographic algorithms,
  - intrusion tolerance approaches,
  - mechanisms to detect/tolerate errors in cryptographic implementations;
- focusing errors and their impact on cryptographic systems
  - identifying vulnerabilities for operation modes;
  - proposing solutions to avoid vulnerabilities in operations modes;
  - proposing and applying new error detection techniques for cryptographic implementations;
  - cost analysis of new error detection techniques and suitability analysis of these new error detection techniques to other algorithms;
  - implementation of a new low-cost error detection mechanism for a cryptographic algorithm;
  - comparison of/with related work results.

## 1.3. Thesis structure

This work has the following structure:

**Chapter 2** introduces the need for security in dependable systems. After a terminology section, the current trends in infrastructures relying on information systems are analyzed. In this chapter we show that the need for security is not only justified by the classical requirements i.e. for confidentiality of certain transferred data, but also by the need for correct/normal functionality/operation, e.g. allowing protection in case of malicious faults (e.g. intrusion attempts).

In **Chapter 3** security is addressed. An up-to-date state-of-the-art investigation of cryptographic techniques and cryptanalysis is presented. Related work on intrusion tolerance is introduced. The challenges given by new type of attacks from the category of side-channel attacks (e.g. fault analysis attacks) are summarized.

**Chapter 4** is focused on security and its need for fault tolerance. Fault tolerance is a dependability means needed in security implementation not only for

correct operation but also for protection against fault analysis attack. Security analysis of a new operation mode is included. Vulnerabilities are identified for different standardized recommendations (e.g. IPsec) and solutions are proposed to avoid such vulnerabilities.

In **Chapter 5** error detection and tolerance mechanisms designed to protect cryptographic implementation against fault analysis attacks are presented. Security analysis of a new proposed error detection mechanism is included and also a case study regarding cost analysis and applicability of available fault tolerance mechanisms for the Triple-DES algorithm. Properties of cryptographic algorithms (e.g. complementation property) are analyzed in order to build new error detection mechanisms. Cost analysis is provided for the new error detection mechanism relying on complementation property.

**Chapter 6** contains evaluation and cost analysis of a new error detection mechanism, using complemented duplication, applied for Misty1 algorithm. The implementations costs are compared with the results of other methods, and the comparison shows that our method has a low hardware and time overhead. Besides the low cost implementation, this detection technique is more secure than other techniques from related work. A clear distinction is made between theoretical assumptions of cost of implementation and the implementation/simulation results reported by simulation tools, where optimization is used. The related work is criticized and we show that in case of implementation using FPGA the reported values are not always in range with the theoretical assumption – due to the composition of the logical units used in the FPGA and due to the mapping process.

In **Chapter 7** conclusions are summarized and future work is proposed.

# 2. ON THE NEED FOR SECURITY IN DEPENDABLE SYSTEMS

Information technologies deployed in various fields, where dependability and survivability are required, rely on security techniques. This reliance on security is justified for instance by traditional requirements for confidentiality and secrecy (e.g. in finance) and by requirements for authentication, integrity, etc. due to the environment in which the underlying system is integrated (e.g. the way the connectivity is assured, for instance using Internet). In this chapter, after introducing the terminology, the problems addressed in this work are presented together with the solutions proposed such that the requirements for security in dependable systems are fulfilled.

## 2.1. Terminology

### 2.1.1. Basic concepts

A **system** is an entity that interacts with other entities, i.e., other systems, including hardware, software, humans, and the physical world with its natural phenomena. Those other systems are the **environment** of the given system [5].

Computing and communication systems are characterized by *functionality*, *performance*, *dependability*, and *cost*. Other system properties are *usability*, *manageability* and *adaptability*. The **function** of a system is what the system is intended to do and is described by *functional specifications*. The **behavior** of a system is what the system does to implement its function, and is described by a sequence of *states*. The **service** delivered by a system, in its role as **provider**, is its behavior as it is perceived by its user(s); a **user** is another system that receives services from the provider [5]. **Correct service** is delivered when the service implements the system function. A **service failure** is an event that occurs when the delivered service deviates from correct service [5]. The period of delivery of incorrect service is a **service outage**.

A deviation from correct state is called an **error**. The adjudged or hypothesized cause of error is called **fault**. Faults can be internal or external to a system. The prior presence of a **vulnerability**, i.e., an internal fault that enables an external fault to harm the system, is necessary for an external fault to cause an error an possible subsequent failure(s). A fault is **active** when it causes an error, otherwise it is **dormant**. The manifestation mechanisms of faults, errors and failures are presented in figure 2.1.

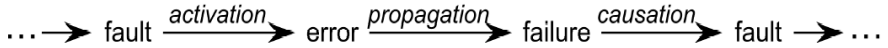


Figure 2.1. Chain of threats [5]

When more services are delivered by a system, the failure of one or more of the services may leave the system in a degraded mode, still offering a subset of the needed services to the user (e.g. slow service). In this context it can be said that the system suffered a partial failure of its functionality or performance.

### 2.1.2. Dependability, security and their attributes

**Dependability** is that property of a computer system such that reliance can justifiably be placed on the service it delivers [2]. The term dependability is used to encapsulate the concepts of reliability, availability, safety, maintainability, performability, and testability according to [6], while [5] includes also integrity and confidentiality.

The **reliability** of a system is a function of time, *R(t)*, defined as the conditional probability that the system performs correctly throughout the interval of time, $[t_0, t]$, given that the system was performing correctly at time $t_0$.

**Availability** is a function of time, *A(t)*, defined as the probability that a system is operating correctly and is available to perform its functions at the instance of time, *t*.

**Safety** is the probability, *S(t)*, that a system will either perform its functions correctly or will discontinue its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system. Safety is a measure of the **fail-safe** capability of a system [6].

The **performability** of a system is a function of time, *P(L,t)*, defined as the probability that the system performance will be at, or above, some level, *L*, at the instance of time, *t*. **Graceful degradation** is the attribute of a system to automatically decrease its level of performance to compensate for hardware or software faults, allowing performance at some reduced level [6]. **Robustness** characterizes a system's dependability with respect to external faults.

**Maintainability** is the probability, *M(t)*, that a failed system will be restored to an operational state within a period of time *t*.

**Testability** is the ability to test for certain attributes within a system [6].

**Integrity** is defined as the absence of improper system alteration.

Consideration of other type of faults, such as intentional malicious and nonmalicious faults, justified the introduction of integrity as an attribute for dependability, and, in the same time bringing together dependability and security to characterize systems.

**Security** encapsulates attributes of *availability, confidentiality* and *integrity* requiring the concurrent existence of 1) availability for authorized actions only, 2) confidentiality, and 3) integrity, as already defined, but with "improper" meaning "unauthorized" [5]. **Confidentiality** is the absence of unauthorized disclosure of information; is a service used to keep the content of information from all but those authorized to have it. **Non-repudiation** prevents an entity from denying previous commitments or actions [3]. **Authentication** is related to identification and is applied to both entities and information: *entity authentication* and *data origin authentication*. Data origin authentication implicitly provides data integrity (if a message is modified, the source has changed). **Authorization** is defined as a conveyance, to another entity, of official sanction to do or be something [3].

In figure 2.2 the relationship between dependability and security is illustrated. As can be seen, *availability* (for authorized action) and *integrity* are common, while other attributes belong to one or the other concepts.

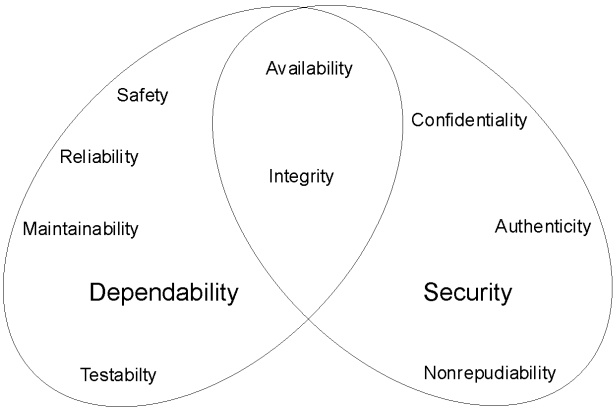


Figure 2.2. Dependability and security relationship

All security attributes mentioned in figure 2.2 are used according to the situation and requirements, so that all parties of a transaction must have confidence that certain objectives associated with information security are met. Besides confidentiality, integrity, entity or data authentication, authorization and non-repudiation other concepts are part of *information security* objectives [3] and some of them are listed below:

**- signature** – a means to bind information to an entity;

**- validation** – a means to provide timeliness of authorization to use or manipulate information or resources;

**- access control** – restriction access to resources to privileged entities;

**- mcertification** – endorsement of information by a trusted entity;

**- confirmation** – acknowledgement that services have been provided;

**- anonymity** – concealing the identity of an entity involved in some process;

**- revocation** – retraction of certification or authorization.

Security can be achieved through the information itself or through other physical means (e.g. physical documents recording it, for banknotes special material and ink are used to avoid counterfeiting). As lately an increased amount of information is managed in electronic form, information security relies on digital information itself. *Algorithms* and *protocols* have been developed to answer the objectives of information security. **Cryptography** supplies a set of *techniques* for information security. *Cryptography* is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, data origin authentication [3]. In section 2.1.3 further explanations are given for cryptography and other related topics.

There are four major categories of *means* to attain *dependability and security* [5]: **fault prevention**, to prevent the occurrence or introduction of faults, **fault tolerance**, to avoid service failure in the presence of faults, **fault removal**, to reduce the number and severity of faults and **fault forecasting**, to estimate the present number, the future incidence, and the likely consequences of faults (figure 2.3).

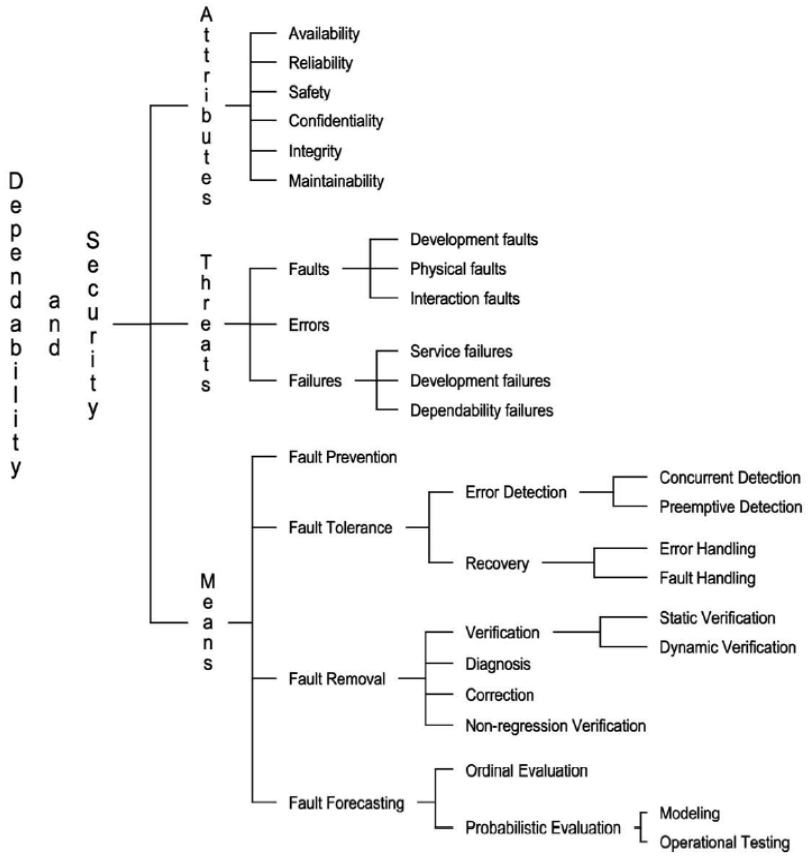


Figure 2.3. Dependability and security tree [5]

**Survivability** is the ability to continue to provide services (even degraded e.g. supporting graceful degradation) in the case of fault or changes/events causing degradation of the system or of its operational environment.

### 2.1.3. Cryptography and cryptanalysis

The art and science of keeping messages secure is **cryptography**, and it is the work of cryptographers. Cryptanalysts are practitioners of **cryptanalysis**, the art and science of *breaking ciphertext*; that is, seeing through the disguise. The branch of mathematics encompassing both cryptography and cryptanalysis is **cryptology** and its practitioners are cryptologists [7].

Cryptography provides a set of techniques to support the four major objectives of the information security: confidentiality, data integrity, authentication and non-repudiation. *Cryptographic tools*, also called *primitives* are used to provide information security [3]. Primitives have been designed to answer the need for confidentiality (e.g. encryption schemes), authentication (e.g. digital signature schemes), etc.

#### 2.1.3.1. Encryption and decryption

One of the purposes of *cryptography*, especially for confidentiality, is to protect transmitted information from being read and understood by anyone except the intended recipient. **Encryption** means the conversion of the original message (plaintext) to encrypted text (ciphertext). The reverse process (conversion of chipertext in plaintext) is called **decryption**.

Usually the following notation is used. M denotes plaintext for message, or P, for plaintext. Ciphertext is denoted by C. The encryption function E, operates on M to produce C:

$$E(M) = C \qquad (2.1)$$

In the reverse process, the decryption function D operates on C to produce M:

$$D(C) = M \qquad (2.2)$$

Since the whole point of encrypting and then decrypting a message is to recover the original plaintext, the following identity must hold true:

$$D(E(M)) = M \qquad (2.3)$$

A **cryptographic algorithm**, also called a **cipher**, is the mathematical function used for encryption and decryption. (Generally, there are two related functions: one for encryption and the other for decryption.)

If the security of an algorithm is based on keeping the way that algorithm works a secret, it is a *restricted* algorithm. Restricted algorithms are inadequate by today's standards. If someone accidentally reveals the secret, everyone must change their algorithm.

Modern cryptography solves this problem with a **key**, denoted by K. The key might be any one of a large number of values. The range of possible values of the key is called the **key space**. Both the encryption and decryption operations use this key (i.e., they are dependent on the key), so the functions now become:

$$E_K(M) = C \qquad (2.4)$$

$$D_K(C) = M \qquad (2.5)$$

These functions have the property that:

$$D_K(E_K(M)) = M \qquad (2.6)$$

Some algorithms use a different encryption key and decryption key. That is, the encryption key, $K_1$, is different from the corresponding decryption key, $K_2$. In this case:

$$E_{K1}(M) = C \qquad (2.7)$$

$$D_{K2}(C) = M \qquad (2.8)$$

$$D_{K2}(E_{K1}(M)) = M \qquad (2.9)$$

All of the security in these algorithms is based in the key (or keys); none is based in the details of the algorithm. This means that the algorithm can be published and analyzed. Products using the algorithm can be mass-produced. If an eavesdropper knows the algorithm he/she cannot read the messages as long as he/she does not have knowledge about the particular key.

There are two general types of key-based algorithms: **symmetric** and **public-key.**

*Symmetric algorithms*, sometimes called conventional algorithms, are algorithms where the encryption key can be calculated from the decryption key and vice versa. In most symmetric algorithms, the encryption key and the decryption

key are the same. These algorithms, also called *secret-key algorithms* require that the sender and receiver agree on a key before they can communicate securely. The security of a symmetric algorithm rests in the key; divulging the key means that anyone could encrypt and decrypt messages. As long as the communication needs to remain secret, the key must remain secret.

Encryption and decryption with a symmetric algorithm are denoted as in equations (2.4) and (2.5) and the equation (2.6) holds for any key.

Symmetric algorithms can be divided into two categories:

Some operate on the plaintext, a single bit (or sometimes byte) at a time; these are called **stream algorithms** or **stream ciphers**.

Others operate on the plaintext in groups of bits. The groups of bits are called **blocks**, and the algorithms are called **block algorithms** or **block ciphers** (the block size can be of 64 bits).

Symmetric block ciphers can be further described based on the techniques used for encryption e.g. **substitution ciphers**, **transposition ciphers**, **product ciphers**, **Feinstel ciphers** [3]. Substitution ciphers replace symbols (or groups of symbols) by other symbols or group of symbols. Transposition ciphers rely on permutation of symbols in a block. A product cipher combines two or more substitutions and/or transpositions in a manner intended to generate a more secure cipher. A **substitution-permutation network (SPN)** is a product cipher composed of a number of stages each involving substitutions and permutations.

**Iterated block ciphers** encrypt a plaintext block by a process that has several **rounds**. In each round, the same transformation (also known as a round function) is applied to the data using a **subkey**. The set of subkeys is usually derived from the user-provided secret key by a special function. The set of subkeys is called the **key schedule**. The number of rounds in an iterated cipher depends on the desired security level and the consequent trade-off with performance. In most cases, an increased number of rounds will improve the security offered by a block cipher, but for some ciphers the number of rounds required to achieve adequate security will be too large for the cipher to be practical or desirable [8].

In a **Feistel cipher** (see Figure 2.4), the text is split into two halves ($L_0$ and $R_0$). The round function F is applied to one half using a subkey and the output of F is XORed with the other half. The two halves are then swapped. Each round follows the same pattern except for the last round where there is no swap. For Feistel ciphers encryption and decryption are structurally identical, though the subkeys used during encryption at each round are taken in reverse order during decryption (i.e. the input in the decryption algorithm is the pair ($R_r$, $L_r$) instead of the pair ($L_0$, $R_0$), and the $i^{th}$ subkey is $k_{r-i+1}$, not $k_i$. This means that we obtain ($R_{r-i}$, $L_{r-i}$) instead of ($L_i$, $R_i$) after the $i^{th}$ round.)

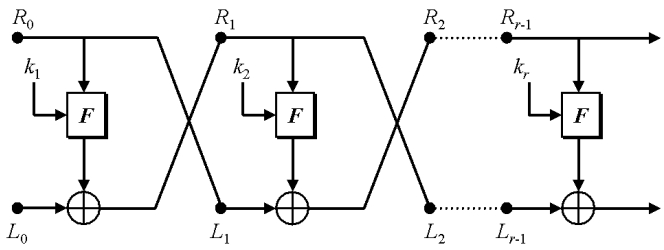


Figure 2.4. Feistel cipher [8]

In case of block ciphers, for encryption/decryption the plaintext/ciphertext is

split into blocks. The way these blocks are manipulated during encryption/decryption is described by so called **modes of operation**. Several modes are also used for hiding existing patterns in the plaintext/ciphertext. Some of the standardized *modes of operation* are presented in Chapter 4.

*Public-key algorithms* (also called asymmetric algorithms) are designed so that the key used for encryption is different from the key used for decryption. Furthermore, the decryption key cannot (at least in any reasonable amount of time) be calculated from the encryption key. The algorithms are called *public-key* because the encryption key can be made public: a complete stranger can use the encryption key to encrypt a message, but only a specific person with the corresponding decryption key can decrypt the message. In these systems, the *encryption key* is often called the **public key**, and the *decryption key* is often called the **private key** (sometimes also called the secret key). Encryption using public key $K_1$ is denoted by as in equation (2.7), decryption using private key $K_2$ as in (2.8).

Sometimes, messages will be encrypted with the private key and decrypted with the public key; this is used in digital signatures.

A **cryptosystem** is an algorithm, plus all possible plaintexts, ciphertexts, and keys.

#### 2.1.3.2. Attacks

We are going to introduce shortly only the relevant attacks for this work. An attempted *cryptanalysis* is called an **attack**. A fundamental assumption in cryptanalysis is that the secrecy must reside entirely in the key [7]. It is assumed that the cryptanalyst has complete details of the cryptographic algorithm and implementation.

The standard technique for defeating a cryptosystem is known as the **brute force method/attack**. This assumes trying all the keys until a key is found which produces the plaintext message. Even if this method is simple and inglorious, brute force is still a form of cryptanalysis. However, using larger keys can make *brute force* a less than feasible technique. For example, a message encrypted with a 56-bit DES key could be broken within a few days of intense computing. A 128-bit key makes the cipher much more effective, ad there are algorithms using even larger keys e.g. RSA encryption commonly uses 512-bit keys. As a result, even with the increased computation power of the computers, better techniques for cryptanalysis are needed. Such a method (any which takes *less time or energy* than brute force) is termed a **break**.

The *parts of the cryptosystem* subjects for attacks are:

*The key*. Though the effectiveness of brute force is inversely proportional to the size of they key space, some ciphers have revealing characteristics which help find the proper key. These "hints" can dramatically reduce the key space that must be searched.

*The data*. Not only the keys, but also the ciphertext itself may have relevant and revealing information embedded within it (e.g. text pattern).

*The algorithm*. The strength of an algorithm is often based upon another problem that has been considered mathematically "difficult". If this characteristic problem can be solved more easily (than previously thought), the cipher may no longer be effective. For example, factoring large numbers is at the heart of RSA encryption. When factoring becomes more efficient than brute force, the cipher will be broken.

The *complexity* of an attack can be measured in different ways:

**Data complexity**. The amount of data needed as input to the attack.

**Processing complexity**. The time needed to perform the attack. This is often called the work factor.

**Storage requirements**. The amount of memory needed to do the attack.

The *complexity* of an attack is taken to be the minimum of these three factors. Some attacks involve trading off the three complexities: a faster attack might be possible at the expense of a greater storage requirement.

While the *complexity* of an attack is constant (until some cryptanalyst finds a better attack, of course), *computing power* is not. There have been phenomenal advances in computing power during the last half-century and there is no reason to think this trend is not going to continue. Many cryptanalytic attacks are perfect for parallel machines: the task can be broken down into billions of tiny pieces and none of the processors need to interact with each other. Pronouncing an algorithm secure simply because it is infeasible to break, given current technology, is hazardous. Good cryptosystems are designed to be infeasible to break with the computing power that is expected to evolve many years in the future [7].

There are numerous techniques for performing cryptanalysis, depending on what access the cryptanalyst has to the plaintext, ciphertext, or other aspects of the cryptosystem. Below are some of the *most common types of attacks*:

**Ciphertext-only attack**. A ciphertext-only attack is one in which the cryptanalyst obtains a sample of ciphertext, without the plaintext associated with it. This data is relatively easy to obtain in many scenarios, but a successful ciphertext-only attack is generally difficult, and requires a very large ciphertext sample;

**Known-plaintext analysis**. With this procedure, the cryptanalyst has knowledge of a portion of the plaintext from the ciphertext. Using this information, the cryptanalyst attempts to deduce the key used to produce the ciphertext. A *known-plaintext attack* is one in which the cryptanalyst obtains a sample of ciphertext and the corresponding plaintext as well;

**Chosen-plaintext analysis** (also known as **differential cryptanalysis**). There is often a statistical correlation between a key and the ciphertext which it produces. Understanding the specifics of this correlation and by using sufficient chosen plaintext can help find an unknown key. The cryptanalyst is able to have any plaintext encrypted with a key and obtain the resulting ciphertext, but the key itself cannot be analyzed. A chosen-plaintext attack is one in which the cryptanalyst is able to choose a quantity of plaintext and then obtain the corresponding encrypted ciphertext;

**Ciphertext-only analysis**. The cryptanalyst has no knowledge of the plaintext and must work only from the ciphertext. This requires accurate guesswork as to how a message could be worded. It helps to have some knowledge of the literary style of the ciphertext writer and/or the general subject matter.

**Man-in-the-middle attack**. This attack relies on tricking individuals into surrendering their keys. The cryptanalyst/attacker places him or herself in the communication channel between two parties who wish to exchange their keys for secure communication (via asymmetric or public key infrastructure cryptography). The cryptanalyst/attacker then performs a key exchange with each party, with the original parties believing they are exchanging keys with each other. The two parties then end up using keys that are known to the cryptanalyst/attacker. This type of attack can be defeated by the use of a hash function.

**Side-channel attacks**. The ciphers are implemented on many different platforms. Certain implementations of cryptosystems allow an attacker to derive the secret key with very low effort - an algorithm which is strong with respect to conventional cryptanalytic attacks can be useless if it cannot be implemented

securely [9]. Side-channel attacks address ciphers implementations.

**Power analysis attacks** are based on the assumption that the instantaneous power consumption of a circuit is dependent to some small extent on instructions and processed data. Such patterns can be detected also measuring the electromagnetic radiation of the unit. Simple power-analysis attacks exploit instruction dependence where every instruction has its unique power-consumption trace. Such an attack typically targets implementations which use key-dependent branching. For example, one can exploit a strong relationship between Hamming weight of the processed data and the power-consumption trace [9][10]. The leakage of Hamming weight information is used to determine the secret key. Masks are used in implementations that try to achieve protection against differential power-analysis attacks.

**Differential power-analysis** correlates processed data with instantaneous power consumption. Output(s) of the real physical device and output(s) of a hypothetical model (based on a hypothetical key) of the device are correlated. If the hypothetical model only outputs a single value (i.e. it predicts the power consumption of the real device for only one moment in time), then the attack is called *first-order differential power-analysis attack*. If a model can output more values then such an attack is called a *higher-order differential power-analysis attack*. For the two most common types of block ciphers, Feistel and Substitution-Permutation Networks (SPNs), different hypothetical models can be developed and the differences between the two structures will only have an impact on dedicated hardware implementations. However it is easier to perform effective differential power attacks on Feistel ciphers rather than on SPN networks [9].

**Timing analysis**. Information is gained based upon how long encryption takes, and used to reveal the algorithm, key, or data. Particularly useful against the smart card, that measures differences in electrical consumption over a period of time when a microchip performs a function to secure information. This technique combined with differential power analysis can be used to gain information about key computations used in the encryption algorithm and other functions related to security. The technique can be less effective by introducing random noise into the computations, or altering the sequence of the executables to make it harder to monitor the power fluctuations.

**Fault Analysis**. Using this attack, specific faults are introduced into the technology (e. g. smart cards) which can help to reveal the keys. It is also possible to force errors to occur during encryption or decryption, and these errors can lead to hardware faults or software error messages that give away information about the key of the cipher. Different evaluation methodologies have not concentrated to any great extent on side-channel attacks. This is because, although they are very important, there is currently little theoretical strategy one can use to assess such attacks. It is left to the hardware/software designer to implement these encryption algorithms without leaving the system open to such attacks (i.e., by masking data and/or introducing randomness into the order or function of inherent logic operations) [9]. In this context, for instance, in the evaluation of primitives submitted to NESSIE only if a side-channel attack applies, regardless of implementation, is considered as a selection criterion [10].

### 2.1.4. Systems and their characteristics

A **distributed system** is a system composed of several computers which communicate through computer network(s), hosting processes that use a common

set of distributed protocols to assist the coherent execution of distributed activities [11].

A **real-time** (RT) system is a system [11] for which the progression is specified in terms of *timeliness* requirements dictated by the environment. Three classes of real-time systems are defined: *hard* real-time systems, where timing failures are to be avoided (e.g. on-board flight control systems); *soft* real-time systems, where occasional timing failures are acceptable (e.g. on-line reservation systems) and *mission-critical* real-time systems, where timing failures should be avoided and occasional failures are handled as exceptional events (e.g. air-traffic control system) [11].

The **system-of-systems** concept describes the large-scale integration of many independent, self-contained systems in order to satisfy a global need. Systems-of-systems are characterized by *operational independence* of elements (components operate independently), *managerial independence* of elements (the component systems maintain a continuing operational existence that is independent of the system-of-systems), *evolutionary development* (functions and purposes can be added, removed, and modified), *emergent behaviors* (the system-of-systems performs functions and carries out purposes that do not reside in any component system) and *geographic distribution* (component systems mainly interact through the exchange of information) [12].

## 2.2. Security as requirement for dependable systems

As defined in the Terminology section, dependability relies on fault prevention, fault tolerance, fault removal and fault forecasting and addresses threats as [5]:

*Development faults* such as software flaws, hardware errata, malicious logic;

*Physical faults* generated by product defects, physical deterioration;

*Interaction faults* e.g. physical interference, input mistakes, attacks including viruses, worms, and intrusions.

Interaction faults are all *operational* faults [5] as they occur during the use phase, and they are all *external* as they are caused by the environment. Most of them are *human-made* (e.g. configuration faults, reconfiguration faults), but they can be caused also by external factors such as cosmic rays.

Interaction faults are more common with the development of more complex distributed systems such as *system-of-systems*. An example is the Internet – a collaborative system-of-systems. Internet component sites collaboratively exchange information using documented protocols. This concept can be considered in other fields as well. For instance integrated air defense system consists of geographically distributed network of semi-autonomous elements (e.g. surveillance systems, radars, launch batteries, control systems). All these components are tied together by a communication network with command and control applied at local, regional and national centers [12]. Real-time communication is required in order to consider the system dependable.

Systems-of-systems cannot deliver all their services without *communication*. Because the elements of the system-of-systems are independent, i.e. from energy point of view for instance, they collaborate only through *information exchange*.

For an interaction fault to have an impact on the functionality of the system, usually a prior presence of a *vulnerability* is required. Such vulnerability can be an internal fault that enables an external one to harm the system. Vulnerabilities can be development or operational faults, and they can be malicious or non-malicious.

Based on the above examples we can distinguish two ways to exploit vulnerabilities of a system-of-systems through the information exchange process:

- by *faulty information* (e.g. a faulty component sends incorrect data – deliberate or not deliberate - to other(s) components, or the response time overcomes the defined requirements).
- by *faulty communication* (e.g. due to failures in communication or due to malicious faults when malicious attempts to manipulate the communications are successful).

For the second way, when vulnerabilities are exploited due to *faulty communication*, security is required to address malicious attempts using cryptographic algorithms for assuring confidentiality, data integrity, entity and data authentication, etc.

This is the *first* motivation supporting the need for security in dependable systems. Security is needed to avoid failure of systems due to errors that could be caused in *the system* by interaction faults (e.g. malicious faults in system implementation).

We are going to bring a *second* motivation for the need for security in dependable systems. This comes in the context of fault tolerance mechanisms developed in dependable systems. Fault tolerance mechanisms in such systems also require protection i.e. security.

Fault tolerance aims to avoid failures using error detection and system recovery. Fault tolerance techniques are designed to improve the availability of the distributed systems, based on adaptation to changing run-time conditions. Given the transition from a single application per machine to distributed systems, new problems have to be solved (e.g. new type of faults left after fault removal). These problems are generated by the interaction between different components of a system, components that usually have a different location, and rely on communication channels for the normal operation of the applications [13]. *New types* of fault tolerance techniques have to address new interaction problems and they need to adapt to the changes in the environment (e.g. due to crash fault, or due to dynamic nature of the system). *Proactive techniques* (such as software rejuvenation) rely on monitoring the behavior and the resources of the system, in order to predict possible failures and to trigger preventive actions accordingly. Some of the decisions generated by the fault tolerance mechanisms are based on monitoring [14], detection, or prediction of the behavior of the components of the distributed system [15]. However, if a third party can modify the input data for monitoring, detection or prediction of fault tolerance mechanisms the decisions taken based on those data are compromised (this is an example of exploited vulnerability using faulty information) [16]. In the same time, by faulty communication, if messages are manipulated (modified or duplicated by an unauthorized third party) the fault tolerance mechanism can trigger an unsuitable action, or the component executing recovery action may execute a wrong action [17]. Furthermore the overall purpose of fault tolerance (improving reliability, dependability) can be compromised.

We can conclude that a dependable system needs security both for the part delivering services for its normal functionality and for the fault tolerance mechanisms designed to assure the dependability of the system.

Authorization, authentication, data integrity, confidentiality required for security purposes are achieved by implementing cryptographic tools. In Chapter 3 cryptographic tools are addressed.

## 2.3. Intrusion tolerance approaches. Using fault tolerance to address security issues

Introduced with malicious objectives, **malicious faults** intend to alter the functionality of the system during use. The goals of malicious faults [5] are:

- to disrupt or halt service, causing denials of service;
- to access confidential information;
- or to improperly modify the system.

Malicious faults are grouped in two classes: **malicious logic faults** and **intrusion attempts**. First class covers (internal) development faults e.g. Trojan horses, and operational faults e.g. viruses.

*Intrusion attempts* are operational (external) faults, so interaction faults. Intrusion attempts may use physical means such as radiations, variation on temperature, power fluctuation to cause faults.

Even if the fault tolerance techniques and mechanisms do not always consider malicious faults (generated by intrusion attempts on the security of the system by both insiders and outsiders), fault tolerance techniques are used to build intrusion-tolerant systems.

To address the specificity of malicious faults, the fault-error-failure model (represented in figure 2.1.) has been extended to security. The extended model (figure 2.5) presents the manifestation mechanism of a successful attack exploiting existing vulnerabilities, which results in intrusion.

**Intrusions** are resulting from (at least) partially successful **attacks** exploiting existing **vulnerabilities**.

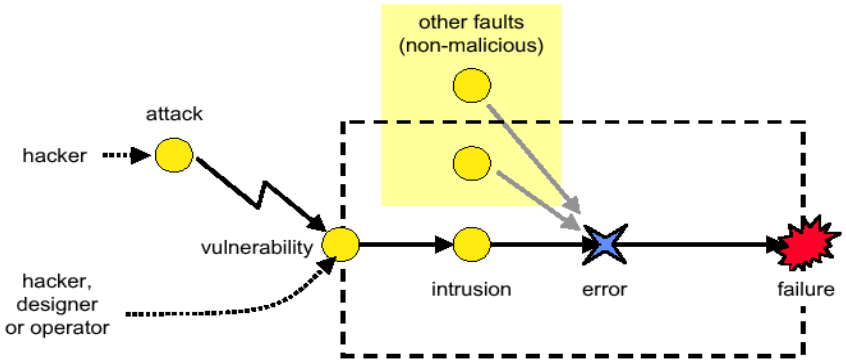


Figure 2.5. Attack-vulnerability-intrusion chain [18]

Besides extending the fault model, based on the dependability means (fault prevention, fault tolerance, fault removal and fault forecasting) presented in the dependability tree in figure 2.3, security means were defined in *MAFTIA (Malicious and Accidental Fault Tolerance for Internet Applications)* project [19]. Figure 2.6 summarizes security means and the methods to handle attacks, vulnerabilities and intrusions.

The use of fault tolerance techniques to built intrusion-tolerant systems has been explored in the European funded research project MAFTIA. Intrusion detection concerns the set of practices and mechanisms used towards detecting errors that may lead to security failure, and/or diagnosing attacks.

**Intrusion tolerance** is the ability of a system to continue providing a secure service (even degraded) despite the presence of malicious faults (i.e.

deliberate attacks on the security of the system by both insiders and outsiders) [19]. In MAFTIA, intrusion tolerance is achieved using an intrusion-tolerant group communication protocol and an intrusion-tolerant distributed authorization service. MAFTIA uses fault masking to achieve intrusion tolerance and does not address fault removal or system reconfiguration.

| | | Attack | Vulnerability | Intrusion |
|---|---|---|---|---|
| Prevention | How to prevent the occurrence or introduction of… | Deterrence, laws, social pressure, secret service… | Semi-formal and formal specifications, rigorous design and management…# | Firewalls, authentication, authorization (+attack prevention and vulnerability prevention) |
| Tolerance | How to provide a service capable of or implementing the system function despite… | Vulnerability prevention<br>Vulnerability removal<br>Intrusion tolerance | = intrusion tolerance | Detection/recovery/ masking, + intrusion tolerance for fault management* # |
| Removal | How to reduce the presence (number, severity) of… | No meaning | Formal proof, model-checking, inspection, test…* | No meaning |
| Forecasting | How to estimate the creation and consequences of… | Intelligence gathering, threat assessment, attack warning… | Assess presence of vulnerabilities, exploitation difficulty, potential consequences | Vulnerability forecasting, attack forecasting |

* targeted by MAFTIA project, based on [18], # targeted by this work

Figure 2.6. Security means and methods

The use of unpredictable adaptation was proposed in another intrusion tolerance related project. The purpose of the *ITUA (Intrusion Tolerance by Unpredictable Adaptation)* project (supported by U.S. Defense Advanced Research Projects Agency) is to develop a middleware based intrusion tolerance solution that would help applications survive certain kinds of attacks (staged attacks which assume that the attacker infiltrates some domains before others) [20]. The ITUA project proposed to add uncertainty in intrusion tolerance technology so that the adaptive responses become unpredictable to the attacker.

This work addresses vulnerability prevention and intrusion tolerance. Vulnerability prevention has a positive impact on the security of any system. Preventing vulnerabilities helps preventing intrusions and failures. Vulnerability prevention can be achieved by using rigorous specifications and resistant implementations. On the other hand not all vulnerabilities can be prevented (e.g. generated by external natural factors) and intrusion tolerance is required. Error detection mechanisms combined with recovery and masking are used to address intrusion tolerance. In this work we propose and apply new error detection mechanisms useful to improve the security of dependable systems.

## 2.4. Contributions and conclusions

In this chapter we have shown that security is required in dependable systems, and security mechanisms are deployed to answer the requirements for security. Security is not only needed in systems where secrecy is traditionally required but also in other systems and infrastructure where safety, reliability, survivability are not achieved in case of malicious attempts if security mechanisms are not implemented.

At the same time we have shown that security is needed not only for the system but also for its fault tolerance mechanisms. We presented that fault tolerance mechanisms need to be secure, otherwise they can be misused, and their entire purpose of tolerating faults can be altered.

Related work on intrusion tolerance has been surveyed. Proposed solutions rely on intrusion-tolerant group communication and distributed authorization services and unpredictable adaptation.

In order to answer to the need for security, the use of cryptographic algorithms and protocols is required. A state-of-the-art survey of cryptographic algorithms follows in Chapter 3.

# 3. SECURITY – MEANS, TRENDS, CHALLENGES

In this chapter the state-of-the-art in cryptography is presented. We are addressing the competitions, organized in Europe, USA and Japan, to select the cryptographic algorithms answering the requirements for complexity and strength against attacks and facing today's computational power. Cryptanalysis is targeted and the trends and challenges of secure cryptographic implementations are presented. The need for fault tolerance in cryptographic implementation is also analyzed in this chapter.

## 3.1. State-of-the-art in cryptography. Competitions and selected algorithms

### 3.1.1. NIST selection process

In 1997, the USA *National Institute of Standards and Technology (NIST)* initiated a process to select a symmetric-key encryption algorithm as *Advanced Encryption Standard (AES)*. In 1998, NIST announced the acceptance of fifteen candidate algorithms and requested the assistance of the cryptographic research community in analyzing the candidates. This analysis included an initial examination of the security and efficiency characteristics for each algorithm.

NIST reviewed the results of this preliminary research and selected MARS, RC6™, Rijndael, Serpent and Twofish as finalists. Having reviewed further public analysis of the finalists, NIST has decided to propose Rijndael as the AES.

#### 3.1.1.1. AES requirements and finalists

The minimum acceptability requirements for AES candidates mandated in the Request for Candidate Algorithm Nominations [21] are:

- the algorithm must implement symmetric (secret) key cryptography,
- the algorithm must be a block cipher, and
- the candidate algorithm shall be capable of supporting key-block combinations with sizes of 128-128, 192-128, and 256-128 bits.

Using the analyses and comments received during AES Round 1, NIST selected five finalist algorithms from the fifteen. The selected algorithms are MARS, RC6, Rijndael, Serpent and Twofish.

#### 3.1.1.2. Comparative performance for AES finalists

In this section the five finalists are compared. Besides performance, the costs generated by masking power consumption (i.e. to protect against power analysis attacks) and the support for instruction-level paralelism are analysed.

Tables 3.1 –3.3 present the hierarchy based on the comparative performance for encryption/decryption, key set-up and the overall comparative performance [22]. The performance values are varying depending on platforms, implementation language, etc. - for instance, in case of 32-bit CPUs, using C, the number of clock cycles varies from 260 clock cycles for RC6 to 1030 clock cycles for Serpent for the encryption and from 850 clock cycles for Rijndael to 8600 clock cycles for Serpent for key setup.

Table 3.1. AES finalists. Encryption and decryption performance by platform.

| | 32-bit (C) | 32-bit (Java) | 64-bit (C and assembler) | 8-bit (C and assembler) | 32-bit smartcard (ARM) | Digital Signal Processors |
|---|---|---|---|---|---|---|
| MARS | II | II | II | II | II | II |
| RC6 | I | I | II | II | I | II |
| Rijndael | II | II | I | I | I | I |
| Serpent | III | III | III | III | III | III |
| Twofish | II | III | I | II | III | I |

Table 3.2. AES finalists. Key scheduling performance by platform.

| | 32-bit (C) | 32-bit (Java) | 64-bit (C and assembler) | 8-bit (C and assembler) | Digital Signal Processors |
|---|---|---|---|---|---|
| MARS | II | II | III | II | II |
| RC6 | II | II | II | III | II |
| Rijndael | I | I | I | I | I |
| Serpent | III | II | II | III | I |
| Twofish | III | III | III | II | III |

Table 3.3. AES finalists. Overall performance.

| | Encryption/Decryption | Key Setup |
|---|---|---|
| MARS | II | II |
| RC6 | I | II |
| Rijndael | I | I |
| Serpent | III | II |
| Twofish | II | III |

Rijdael has the best performance values for most encryptions/decryptions and the best values for key scheduling.

Table 3.4. presents the results of a case study analysing the cost of masking power consumption in order to defend smart card implementations of the finalists. In this study [22], implementations were improved with defenses against *power analysis attacks*. The performance degradation caused by these defenses is measured. The study compares the results of implementations both with and without masking (used as a defense against power analysis attacks). A generalization of software balancing is used. In software balancing, the bit-wise

complements of data words are generated; in [23], random strings of bits, called masks, were generated to combine with the input data and key data. The fundamental algorithm operations were then carried out on the masked data, after which the masks were removed. Because different random masks were used for every execution of the algorithm, over a statistical sample, the power consumption should be uncorrelated with the secret key and the input and output data.

The implementations are performed on a high-end, 32-bit ARM-based card. In the cases of four of the finalists (all but Twofish) the RAM requirements were similar and the major distinctions came in speed and ROM requirements.

Table 3.4. AES finalists. A smart card study of power analysis defence [22] from [23]

| | Cycle count, no masking | Cycle count, with masking | RAM, no masking | RAM, with masking | ROM, no masking | ROM, with masking |
|---|---|---|---|---|---|---|
| MARS | 9425 | 73327 | 116 | 232 | 2984 | 7404 |
| RC6 | 5964 | 46282 | 232 | 284 | 464 | 1376 |
| Rijdael | 7086 | 13867 | 52 | 326 | 1756 | 2393 |
| Serpent | 15687 | 49495 | 176 | 340 | 2676 | 9572 |
| Twofish | 19274 | 36694 | 60 | 696 | 1544 | 2656 |

Table 3.5. AES finalists. Critical path and instruction-level parallelism. [22]

| | The first estimate of critical path (clock cycles) | The second estimate of critical path (clock cycles) | Est. throughput (bit/cycle) on a hypothetical VLIW proc. 5 instr. issue slots | Throughput (bit/cycle) on an actual VLIW 5 instr. issue slots, in feedback mode | Est. of max. no. of processing elem. that can be effectively used in parallel (Par) | IPC |
|---|---|---|---|---|---|---|
| MARS | 258 | 214 | 0,56 | 0,57 | 2 | 2 |
| RC6 | 185 | 181 | 0,69 | 0,69 | 2 | 2 |
| Rijdael | 86 | 71 | 0,93 | 0,93 | 7 | 10 |
| Serpent | 556 | 526 | 0,27 | 0,28 | 3 | 3 |
| Twofish | 166 | 162 | 0,69 | 0,70 | 3 | 6 |

*Potential for Instruction-Level Parallelism.* Future processors will support various modes of parallelism to a greater extent than existing processors. The support for parallelism is investigated and the five algorithms are evaluated from instruction-level parallelism perspective. The purpose is to answer question as to what extent can the finalist take advantage of this situation if an unlimited number

of instruction issue slots are available so that any potential parallelism for single block encryption in a finalist can theoretically be exploited.

Some information can be gathered from an examination of the operations to be executed for an algorithm. One concept, in this regard, is that of a *critical path* through code for a particular instruction set: each instruction can be weighted according to the number of latent clock cycles. Latent clock cycles refer to the number of cycles between the instruction issuance and the availability of the result to another instruction. A critical path could then be defined to be the path from plaintext to ciphertext requiring the largest number of cycles. Table 3.5 presents the results of several studies.

#### 3.1.1.3. Selection conclusions

Each of the finalist algorithms appears to offer adequate security, and each offers a considerable number of advantages. However, each algorithm also has one or more areas where it does not perform quite as well as some other algorithm; none of the finalists is outstandingly superior to the rest.

NIST selected Rijndael as the proposed AES algorithm at the end of evaluation process. During the evaluation, NIST analyzed public comments, papers, verbal comments at conferences, and NIST studies and reports. NIST judged Rijndael to be the best overall algorithm for the AES [22].

Rijndael appears to be consistently a very good performer in both hardware and software across a wide range of computing environments regardless of its use in feedback or non-feedback modes. Its key setup time is better than for the other algorithms, and its key agility is good. Rijndael is also characterized by low memory requirements, (this makes it very well suited for restricted-space environments, in which it also demonstrates excellent performance) and its operations are among the easiest to defend against power and timing attacks. Additionally, it appears that some defense can be provided against such attacks without significantly impacting the performance of Rijndael. Rijndael is designed with flexibility in terms of block and key sizes, and the algorithm can accommodate alterations in the number of rounds. The internal round structure of Rijndael can benefit from instruction-level parallelism.

Based on these considerations and evaluations results combining security, performance, efficiency, implementability, and flexibility, Rijndael has been selected as AES for current and future use [22].

### 3.1.2. NESSIE research project

The main objective of the *NESSIE (New European Schemes for Signature, Integrity, and Encryption)*, European funded IST project, was to put forward a portfolio of strong cryptographic primitives of various types. The project started with an open call for the submission of cryptographic primitives as well as for evaluation methodologies for these primitives. This call includes a request for the submission of not only block ciphers (as for the AES call), but also of other cryptographic primitives including hash functions, stream ciphers, and digital signature algorithms. The call also asked for evaluation methodologies for these primitives. The scope of the call was defined in conjunction with the project industry board, and was published in March 2000 [24].

The NESSIE project proposed to disseminate the project results widely and to build consensus based on these results by using the appropriate bodies: a project industry board, NESSIE workshops, the 5$^{th}$ Framework programme, and various

standardization bodies [24].

The open call leaded to the submission of forty cryptographic primitives to the NESSIE project. These submitted primitives were evaluated (with some external assistance) from both a security and performance perspective.

#### 3.1.2.1. Evaluation and selection criteria

The *evaluation criteria* published in the NESSIE call are:

- An attack should be at least as difficult as the generic attacks against the type of primitive (exhaustive search, birthday attack etc.).
- Primitives will be *evaluated against the security claims of the submitter*. An attack requiring lower computing resources than claimed would usually disqualify the submission.
- Primitives will be *evaluated within the stated environment*. Thus, consideration of vulnerability to side channel attacks (e.g. timing attacks, power analysis) may be appropriate.

The main *selection criteria* specified in [24] are:

- *Long-term security*. Security is the most important criterion, because security of a cryptographic primitive is essential to achieve confidence and to build consensus. Evaluation process considers the evolutions and developments outside the project (such as new attacks or analysis techniques).
- *Market requirements.* Market requirements are related to the need for a primitive, its usability, and the possibility for world-wide use.
- *Efficiency.* From performance perspective, for software, the range of environments considered include 8-bit processors (as found in inexpensive smart cards), 32-bit processors (e.g., the Pentium family) to the 64-bit processors. For hardware, both FPGAs and ASICs are considered.
- *Flexibility.* It is clearly desirable for a primitive to be suitable for use in a wide-range of environments.

Some of the methodological issues were:

- *Resistance to cryptanalysis*. Submitted primitives were required to be resistant at the relevant security level to cryptanalytic attacks. The failure to be resistant to such an attack disqualifies a submission. However, when assessing the relevance of a cryptanalytic attack, other factors such as the volume and type of data required to mount the attack are considered.
- *Design philosophy and transparency*. An important consideration when assessing the security of a cryptographic primitive is the design philosophy and transparency of the design of that primitive. It is easier to have confidence in the assessment of the security of a primitive if the design is clear and straightforward, and is based on well-understood mathematical and cryptographic principles. This is particularly relevant when making relative comparisons between primitives.
- *Strength of modified primitives*. One common technique used to assess the strength of a primitive is to assess a modified primitive, for example by changing or removing a component or reducing the number of rounds.

Conclusions about the original primitive based on an assessment of the modified primitives have to be carefully considered as the inference may or may not be straightforward.

- *Relative security*. When assessing primitives designed to operate to the same security level in similar environments, it is natural to wish to compare their security. However, care has to be taken when making such comparisons. One measure that has been suggested for primitives based on an iterative algorithm is the security margin, which measures the gap between the maximum number of broken rounds and the total number of rounds, but there is no general consensus about its definition or use. Furthermore, whilst the NESSIE project tried to ensure that each submitted primitive receives equivalent cryptanalysis, it is the case that some designs are easier to analyze than others. However, it is felt that there should be some security margin to protect against cryptanalytic advances.
- *Cryptographic environment.* In certain cryptographic environments, a cryptographic primitive may have been designed to possess intrinsic security advantages or disadvantages. An example would be a primitive that is resistant to power or timing attacks when implemented on a smart card. Such properties would be considered when assessing the security of a primitive.
- *Statistical testing*. Statistical testing of submitted primitives by NESSIE project was carrying out. The purpose of this statistical testing is to highlight anomalies in the operation of the primitive that may indicate cryptographic weakness and require further investigation.

#### 3.1.2.2. NESSIE selected algorithms

On February 27, 2003, NESSIE project consortium announced final selection of cryptographic algorithms [25]. The evaluation process was open, based on the published evaluation criteria. Feedback has been received from the global cryptographic community; these comments have been made public.

Table 3.6 lists the selected NESSIE algorithms: 12 algorithms from the 42 submissions; other 5 well established standard algorithms have been added to the NESSIE portfolio (indicated with * in table 3.6).

No weaknesses have been identified in these 17 algorithms till the end of selection process, however, later, SFLASH was broken, and SFLASHv2, is not considered secure enough [26]. None of the six submitted stream ciphers meets the security requirements of NESSIE.

*Licenses*. The 10 symmetric primitives in this portfolio (4 block ciphers, 4 MAC algorithms and 2 hash functions) can be used for free. The asymmetric primitives RSA-KEM, RSA-PSS and SFLASH are also in the public domain. PSEC-KEM is available under favorable conditions. Licenses need to be negotiated for ACE Encrypt, ECDSA and GPS, but the owners have promised to offer reasonable and non-discriminatory terms [25].

Table 3.6. NESSIE portfolio

| | | |
|---|---|---|
| Block ciphers | MISTY1 | Mitsubishi Electric Corp., Japan |
| | Camellia | Nippon Telegraph and Telephone Corp., Japan and Mitsubishi Electric Corp., Japan |
| | SHACAL-2 | Gemplus, France |
| | AES * | (Advanced Encryption Standard) (USA FIPS 197) (Rijndael) |
| Public-key encryption | ACE Encrypt | IBM Zurich Research Laboratory, Switzerland |
| | PSEC-KEM | Nippon Telegraph and Telephone Corp., Japan |
| | RSA-KEM* | (draft of ISO/IEC 18033-2) |
| MAC algorithms and hash functions | Two-Track-MAC | K.U.Leuven, Belgium and debis AG, Germany |
| | UMAC | Intel Corp., USA, Univ. of Nevada at Reno, USA, IBM Research Laboratory, USA, Technion, Israel and Univ. of California at Davis, USA |
| | CBC-MAC* | (ISO/IEC 9797-1) |
| | HMAC* | (ISO/IEC 9797-1) |
| | Whirlpool | Scopus Tecnologia S.A., Brazil and K.U.Leuven, Belgium |
| | SHA-256*, SHA-384* and SHA-512* | (USA FIPS 180-2). |
| Digital signature algorithms | ECDSA | Certicom Corp., USA and Certicom Corp., Canada |
| | RSA-PSS | RSA Laboratories, USA |
| | SFLASH | Schlumberger, France |
| Identification schemes | GPS | Ecole Normale Supérieure, Paris, France Télécom and La Poste, France |

### 3.1.3. CRYPTREC IPA research project (CRYPTography Research and Evaluation Committees)

The *Information-technology Promotion Agency (IPA)* in Japan has initiated the *CRYPTREC project (CRYPTography Research and Evaluation Committees)* with the scope to define standard cryptographic algorithms for use within the Japanese e-Government infrastructure [27].

CRYPTREC project started in 2000. Different types of cryptographic techniques were submitted to the formal *Call for Cryptographic Techniques* dated June 13, 2000. As in case of NESSIE initiative, CRYPTREC call was open for different types of primitives. Also, the purpose was to select a set of techniques not only one as in AES selection.

Some of the algorithms evaluated in NESSIE project (e.g. RC6, MISTY1, Camellia, AES) were also submitted to CRYPTREC for evaluation [27].

In the table 3.7, in the second column, there are listed the submissions for CRYPTREC, and in the last one the primitives added for evaluation.

Table 3.7. CRYPTREC evaluated primitives

| Category | Submissions to CRYPTREC | Other eval. primitives |
|---|---|---|
| Asymmetric Cryptographic techniques (Confidentiality) | ACE Encrypt, ECAES(Elliptic Curve Augmented Encryption Scheme) in SEC1, EPOC, HIME-2, PSEC | RSA OAEP |
| Asymmetric Cryptographic techniques (Authentication) | ESIGN-identification | - |
| Asymmetric Cryptographic techniques (Signature) | ACE Sign, ECDSA(Elliptic Curve Digital Signature Algorithm) in SEC1, ESIGN-signatures, MY-ELLTY ECMR-h | DSA, RSA PSS |
| Asymmetric Cryptographic techniques (Key-sharing) | ECDHS (Elliptic Curve Deffie-Hellman Scheme) in SEC1, ECMQVS (Elliptic Curve MQV Scheme) in SEC1, HDEF-ECDH, HIME-1 | DH Key Exchange |
| Symmetric Ciphers (Stream ciphers) | MULTI-S01, TOYOCRYPT-HS1 | - |
| Symmetric Ciphers (64-bit block ciphers) | CIPHERUNICORN-E, FEAL-NX, Hierocrypt-L1, MISTY1 | Triple DES |
| Symmetric Ciphers (128-bit block ciphers) | Camellia, CIPHERUNICORN-A, Hierocrypt-3, MARS, RC6, SC2000 | Rijndael |
| Hash Functions | - | MD5, RIPEMD-160, SHA-1 |
| Pseudo-Random Number Generators | TOYOCRYPT-HR1 | PRNG based on SHA-1 (BFIPS186) |

A second call, *Call for attack to evaluate the Cryptographic Techniques*, in this case for public analysis and comments was published in October 23, 2000 by IPA. Other cryptographic techniques were added for evaluation by CRYPTREC.

Table 3.8 contains the selected CRYPTREC primitives, including the notes and special recommendations as published on [27]. It can be noticed that some of the algorithms are selected for the time being due to their integration in widely used security mechanisms; however stronger algorithms are recommended to be used if possible.

Table 3.8. CRYPTREC selected primitives

| | | |
|---|---|---|
| Public-key ciphers | Signature | DSA, ECDSA, RSAASSA-PKCS1-v1 5, RSA-PSS |
| | Confidentiality | RSA-OAEP, RSAES-PKCS-v1 5*[1] |
| | Key agreement | DH, ECDH, PSEC-KEM*[2] |
| Symmetric-key ciphers | 64-bit block ciphers *[3] | CIPHERUNICORN-E, Hierocrypt-L1, MISTY1, 3-key Triple DES *[4] |
| | 128-bit block ciphers | AES, Camellia, CIPHERUNICORN-A, Hierocrypt-3, SC2000 |
| | Stream ciphers | MUGI, MULTI-S01, 128-bit RC4 *[5] |
| Others | Hash function | RIPEMD-160 *[6], SHA-1*[6], SHA-256, SHA-384, SHA-512 |
| | Pseudo-random number generator *[7] | PRNG based on SHA-1 in ANSI X9.42-2001 Annex C.1, PRNG based on SHA-1 for general purpose in FIPS 186-2 (+ change *[1]) Appendix 3.1, PRNG based on SHA-1 for general purpose in FIPS 186-2 (+ change *[1]) revised Appendix 3.1. |

Notes

*[1] this permitted for the time being because it is used in SSL3.0/TLS1.0

*[2] On the assumption that is used in the KEM (Key Encapsulation Mechanism) –DEM (Data Encapsulation Mechanism) construction

*[3] 128-bit block ciphers are preferable if possible

*[4] Using the 3-key Triple DES is permitted for the time being under the following conditions 1) it is specified as FIPS 46-3, 2) it is positioned as de facto standard

*[5] It is assumed that the 128-bit RC4 will be used ONLY in for SSL3.0/TLS(1.0 or later). If any other cipher listed above is available, it should be used instead.

*[6] If any cipher with a longer hash value are available, it is preferable that a 256-bit (or more) hash function to be selected. However, this does not apply in cases where the hash to be used has already been designed according to the public-key cryptographic specifications.

*[7] Since pseudo-random number generators do not require interoperability due to their usage characteristics, no problems will be generated from the use of a cryptographically secure pseudo-random number generating algorithm. Therefore, these algorithms are examples.

### 3.1.4. Summary on the state-of- the- art in cryptography

In the first part of this chapter we analyzed the last three main competitions and evaluation processes focusing selection of competitive cryptographic algorithms. NIST competition, compared with NESSIE and CRYPTREC addressed only 128-bit block ciphers and aimed only one algorithm, while the other two covered several algorithms from all types of primitives and also evaluation criteria. For comparison purposes, for NESSIE and CRYPTREC, were added as well standardized algorithms such as AES. Some algorithms were submitted and evaluated for all competitions and only Rijndael was selected (e.g. RC6 was not selected, rejection was based also

on license reasons). Several algorithms evaluated only by NESSIE and CRYPTREC were selected in both competitions – this is the case of the Japanese algorithms MISTY1 and Camellia for block ciphers, ECDSA and RSA-PSS for digital signature, etc. Algorithms such as 3-key Triple-DES were included in the recommendation list by CRYPTREC due to its usage in SSL3.0/TLS1.0 network security standards. Further description of these algorithms can be found on the web pages of the selection projects. However Triple-DES and MISTY1 are further presented in Chapters 5 and 6 respectively, where error detection mechanisms to protect against fault analysis attacks are covered.

The selections of cryptographic algorithms are made based on a number of criteria such as security, effectiveness, cost, implementability, intellectual property status etc. Establishing the level of security is not an easy task. Most cryptographic mechanisms rely on one or more unproven assumptions or hard problems. An independent evaluation is required and this implies a substantial amount of research. Public key cryptography relies on 'hard' mathematical problems such as factoring problem or discrete logarithm problem. In general, the conclusions of all evaluations are valid 'for the time being' [26]. Unexpected breakthroughs may always occur. As such the state-of-the-art in cryptography can change quickly.

Due to the nature of cryptographic techniques, the results of security evaluations described in previous subsections may not remain valid in the future. Every year several new weaknesses are identified, leading to changes or even vulnerabilities are identified in deployed systems that require immediate modification. Thus, it is considered necessary to continue such evaluations.

Any design of security mechanism must consider this context. Security mechanisms must be *upgradeable* (to face changes and new context). *Composability* is required in security mechanisms (cryptographic systems) in order to allow security mechanisms' components (algorithms, operational modes) to be modified/switched/updated according to the 'reality' (e.g. new algorithms, new implementations).

## 3.2. Cryptanalysis. Attacks based on implementation. Fault analysis attacks

Cryptographic algorithms, including symmetric ciphers, public-key ciphers, and hash functions, are used as building blocks to construct security mechanisms that target specific objectives. For example, network security protocols, such as SSH, combine these primitives to provide authentication between communicating entities, and ensure the confidentiality and integrity of communicated data. These security mechanisms only specify what functions are to be performed. The specification of a security protocol is usually independent of whether the encryption algorithms are implemented in software running on a general processor, or using custom hardware units. The specifications do not consider whether the memory used to store intermediate data during these computations is on the same chip as the computing unit or on a separate chip [28].

This separation between security mechanisms and their implementation has as main advantage the ability to allow for theoretical analysis and design of cryptosystems and security protocols. Differential cryptanalysis and linear cryptanalysis are examples of cryptanalysis focusing the mathematical paradigm and using high-powered mathematical tools to break different ciphers. Such techniques exploit weaknesses in algorithms and do not address implementation aspects. Still, various assumptions are made about the implementation of security

mechanisms (i.e. that the implementations can neither be observed nor interfered by any malicious entity). Due to such assumptions the level of security is quantified in terms of the mathematical properties of the cryptographic algorithms and their key sizes.

However cryptographic algorithms are always implemented in software or hardware. Physical devices are used for implementation and they interact and are influenced by their environment. These physical interactions can be triggered by an attacker and monitored, and may result in additional information useful in cryptanalysis. The additional information can provide enough information to compromise the security of the system.

Such additional information are power consumption, timing information, electromagnetic radiations emitted, special behavior due to internal faults, etc. of the circuit implementing the algorithm (cryptosystem). The attacks based on the use of such specific information are called *side-channel attacks*.

### 3.2.1. Cryptosystems and side-channel attacks

The underlying idea of *side-channel attacks* is to exploit the way cryptographic algorithms are implemented, rather than the algorithm itself.

As already mentioned, there are two ways to look at a cryptographic primitive (i.e. block cipher, digital signature function, etc). The first is only as mathematical problem, where a message M is encrypted with a key $K_1$ ($E_{K1}$) to produce a cipher text C (or, the cipher text C is decrypted with a key $K_2$ ($D_{K2}$) to produce the plaintext M), as in figure 3.1.

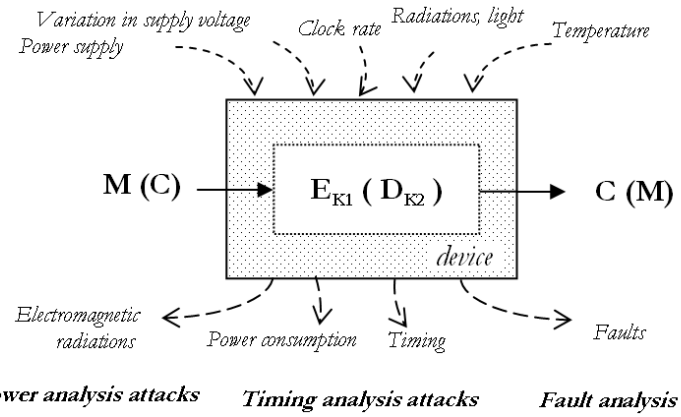


Figure 3.1. Encryption (decryption), real world interactions and side-channel attacks

If $K_1=K_2$ we have a symmetric algorithm. The second way is as physical (or software) implementation. The implementation both in software and in hardware requires the use of devices, and, these devices are interacting with the environment (figure 3.1). Data (e.g. power consumption, timing) can be collected based on these interactions in order to gather information (e.g. operation being performed) about algorithms. The implementation is in this way vulnerable to the external environment, and intended changes (e.g. variation of power supply, clock rate) may cause effects useful for an attacker (e.g. faults).

The first side-channel attack, official recorded, was using in 1965 a microphone to record the sounds produced by a rotor-cipher machine. This method

deduced the position of couple of rotors adding sufficient information to break the cipher and to allow British intelligence agency to spy on the embassy's communication [28].

Side-channel attacks (timing attacks, power analysis, fault analysis etc.), all make assumptions about implementation, and use additional information gathered from attacking these implementations (i.e. how the power consumption changes as the cipher executes, what the output looks like when you cut some wires) in an attempt to recover the key. These attacks work because there is a correlation between the values of different physical parameters at different points during the computation and the internal state of the processing device, which is itself related to the secret key [28].

Side-channel attacks do not always generalize. A fault-analysis attack is not possible against an implementation that does not allow an attacker to create and exploit the required faults. For instance, attacks that measure power consumption of a cryptographic device can be possible if the device is a smart card that draws power from an external, untrusted source. On the other hand, if the device is a workstation located in a secure office, then power consumption attacks are not a significant threat.

However side-channel attacks are powerful. For example, differential fault analysis of DES requires between 50 and 200 ciphertext blocks to recover a key, while the best non-side-channel attack against DES requires about 64 terabytes of plaintext and ciphertext encrypted using the same key [28].

To protect the cryptosystems against side-channel attacks there are two types of measures:

- to reduce the amount of side-channel information that leaks (using protection, shielding to eliminate or reduce radiations, light), or
- to make the side-channel information irrelevant (by adding redundant component to make more difficult extracting information from monitored i.e. power consumption).

The protection methodology cannot be generalized. It depends

- on the algorithm,
- on the implementation and environment, and
- on the side channel attack considered.

And besides these dependencies, protection generates extra costs. In case of energy masking (to protect against power analysis attack) additional energy is consumed in the circuits added for masking. In case of fault analysis attack, protection mechanisms may generate extra hardware or time requirements. Information about the key can leak (timing attack) in case of ciphers with non-constant execution time but making the execution time constant reduces performance.

Protection against side-channel attacks is one of research areas where more work is required. In the following sections and chapters vulnerabilities are identified and protection is proposed for limiting the effect of faults and fault analysis attacks in cryptosystems.

### 3.2.2. Fault attacks. Fault injection methods

As represented in figure 3.1, besides functional parameters such as power consumption, timing aspects, etc., faults can be useful for side-channel attacks.

In order to mount fault attacks there are two stages required. In the first

stage the fault is injected and in the second stage the errors are exploited using cryptanalysis.

For the first stage, the efficiency of a fault attack depends on the type of faults that can be induced. According to [28], such a *fault model* is described by the following aspects

- *The precision of the fault* - the time and location on which the fault occurs during the execution of a cryptographic module.
- *The length of the data affected* by a fault i.e. only one bit, or one byte.
- *The persistence* of the fault i.e. transient or permanent fault.
- *The type of the fault* i.e. flip one bit; flip one bit, but only in one direction (e.g. from 1 to 0); byte changed to a random (unknown) value; and so on.

There are several methods for fault injection. The precision, the length of data affected, the persistence and the type of faults depend on the fault injection method. According to [29], some of the fault injection methods are shortly presented below

- *Variation in supply voltage*. During execution, such variation determines missing executions or skipping of instructions.
- *Variations in external clocks*. Due to higher frequency clock, the circuit starts executing next instruction before current instruction is executed and not all results or data are available.
- *Temperature*. Circuits have defined upper and lower temperature thresholds for correct functionality. For instance in the case of non-volatile memories the temperature thresholds for read and write do not coincide and an attacker may expose the circuit to a temperature where write operates and read does not operate (or the other way around) to mount attacks.
- *White light*. Due to photoelectric effects, all electric circuits are sensitive to white light. If the circuit is exposed to intense white light, the current induced by photons can induce faults.
- *Laser* can target a small circuit area with an effect similar with the effect of white light.
- *X-rays and ion beams*. Even if is not common, this method allows fault attacks implementations without requiring de-packaging the chip.

As was mentioned above, from persistency point of view, there are two types of faults that can be induced into electronic circuits: transient and permanent faults. Transient faults allow a large number of experiments until the desired effects are obtained. After the attack ends, the system remains functional. These faults are preferred compared with permanent ones, where another system/circuit is required to inject different faults.

### 3.2.3. Fault analysis attack

A new theoretical model for breaking various cryptographic schemes by taking advantage of *random hardware faults* has been presented in 1996 [30], by Boneh, Demillo and Lipton.

The model consists of a black-box containing some cryptographic secret [31]. The box interacts with the outside world by following a cryptographic protocol. The model supposes that from time to time the box is affected by a random hardware fault causing it to output incorrect values. For example, the hardware fault flips an internal register bit at some point during the computation. In [31] was shown that for many digital signatures and identification schemes these incorrect outputs completely expose the secrets stored in the box.

At the beginning this attack was considered to be applicable to public key cryptosystems and not to secret-key algorithms. However, Biham and Shamir propose a related attack called *Differential Fault Analysis* (DFA) in [32]. They showed that DFA was applicable to almost any secret key cryptosystem proposed in the open literature at that moment.

The main criticism against DFA was that the transient fault model that was claimed to be unrealistic. Starting from this, Biham and Shamir decided to develop a more practical fault model based on *permanent hardware faults*. They showed that their model could be used to break Data Encryption Standard (DES) [32]. They called this *Non-Differential Fault Analysis* (NDFA). For this attack they proposed to cut a wire or permanently destroy a memory cell. A smartcard implementation of DES was used to describe the attack. Two types of implementation have been analyzed. In the first case, DES was implemented in hardware as a single round that is used 16 times (for the 16 rounds of the algorithm). The second uses an unrolled implementation, where all 16 rounds use different, separate hardware modules. For second implementation, the attacker is able to retrieve more easily bits of the subkeys.

As shown in previous section, faults can be induced by radiation, extreme temperature, incorrect voltage (voltage spikes), atypical clock rate – all of these difficult to control, and light – which generates much easier to control location and type of fault. Those external factors can determine malfunction of a part of a circuit (e.g. wire stuck-at zero, output gate stuck-at one, etc). The implementations of the security primitives need to be fault-tolerant and mechanisms for error detection are required to prevent failures generated by such faults.

The rest of the thesis analysis how injected faults can affect the security of block ciphers implementations and proposes solutions to reduce the vulnerabilities and to detect errors. Chapter 4 addresses vulnerabilities caused by faults in case of the standardized modes of operation. In Chapter 5, a more detailed description of fault analysis attacks is presented together with available methods for protection against such attacks in case of block ciphers; furthermore new methods are proposed and evaluated.

## 3.3. Standards

Security standards, as standards in general can bring benefits such as interoperability, guarantee of quality and reduced development time and costs. Security standards are useful also for non-experts in security i.e. they can implement an evaluated algorithm without investing in security assessment. However the benefits are not always there. In case of security, due to the long standardization process, some of the standards are outdated (e.g. algorithms are not suited anymore to the intended purpose). The extensive use of the same standard can have a twofold impact in case of security: on one hand it can bring better evaluation and fast detection of weaknesses, while on the other, it can increase the security risks (e.g. viruses can spread exploiting a single security vulnerability if the same operating system is used) [26].

According to [26] standards can be

- *base standards*, including standards for cryptographic algorithms, for modes of use, etc.;
- *functional standards,* where is explained the way base standards are used in network security (e.g. IPsec, TLS), financial transactions etc.;
- *evaluation criteria standards,* addressing, as the name states, the

evaluation of products and systems;

- interpretative documents and best practices standards, where, for instance guidelines are presented.

Example of base standards are FIPS 197 [33] describing the selected AES algorithm after NIST selection process presented in section 3.1.1 and FIPS 81 [34] which addresses the modes of operations described in Chapter 4.

As in the case of FIPS 197, standards describing algorithms are created after selection and evaluation processes. Some standards may also be supported by industry with large market share. In any case, standards require maintenance. In case of information security standards, state-of-the-art can change quickly [26] and due to vulnerabilities immediate modifications are required.

From geographical perspective, standards can be at international level, European level or national level. Examples of international standards are *ISO (International Organization for Standards, with IEC - International Electrotechnical Commission)* standards (e.g. ISO/IEC 10116 standard dedicated for the modes of operation of an n-bit block cipher algorithm [35]), *IETF* (*Internet Engineering Task Force*) standards (e.g. IETF standard for IPsec [36]) or *IEEE (Institute of Electrical and Electronic Engineers)* standards.

At European level, the equivalent of ISO/IEC are *CEN (Comite Europeen de Normalisation)* and *CENELEC*. Examples of European level standards are the 3GPP/UMTS security specifications developed by *ETSI (European Telecommunication Standardization Institute)* for GSM security. For instance for 3GPP (*3rd Generation Partnership Project*) security specifications, which solves the GSM previous flaws, a modified version of MISTY1, algorithm selected both by NESSIE and CRYPTREC, called KASUMI is used for confidentiality and integrity purposes (see Chapter 6 for more details on MISTY1).

For national level, the US NIST (National Institute for Standards and Technology) is one of the key players in information security (for example AES standard FIPS 197 [33]).

There are also industry standards. An example is the ATM Security Specifications elaborated by the ATM Forum Technical Committee [37] (see Chapter 4 for some details).

It can be noticed that there are a large number of standards and standardization bodies in the area of security. However, as the security field knows a fast evolution, security standards are required to answer the need for changes. Algorithms and specifications which are included in standards are bringing the certitude that they have been evaluated and they are under a maintenance process. Some standards shortly mentioned in this section could be found in the following chapters of this work. We chose to address in this work standardized algorithms and recommendations in order to address secure, robust and evaluated primitives and specifications.

## 3.4. Conclusions and contributions

In this chapter we introduced the recent major cryptographic selection competitions organized in USA, Europe and Japan. These selections addressed block ciphers i.e. NIST selection initiative in USA or secret and public cryptographic techniques in NESSIE in Europe or CRYTREC in Japan.

The selection criteria are security, effectiveness, cost, intellectual property status, etc. Due to the nature of cryptographic techniques, the results of security evaluations described in this chapter may not remain valid in the future. Every year

several new weaknesses are identified leading to changes, or even vulnerabilities are identified in deployed systems that require immediate modification. Thus, it is considered necessary to continue such evaluations.

Even if these evaluations are focusing on mathematical models (algorithms) and protocols, cryptographic algorithms are always implemented in software or hardware. Physical devices are used for implementation and they interact and are influenced by their environments. These physical interactions can be investigated and monitored, and may result in additional information useful in cryptanalysis. Such additional information can provide enough information to compromise the security of the cryptographic system. As such, only a secure mathematical model is not sufficient in practice, the cryptographic systems must be secure also in front of side-channel attacks.

Fault analysis attack, one of the side-channel attacks, takes advantage of faults induced in cryptographic implementations, and may considerably reduce the effort of cryptanalysis. Fault injection methods are introduced in this chapter and fault analysis attacks are shortly presented.

The entire chapter investigates and presents the state-of-the-art in cryptology. It shows the most secure cryptographic primitives selected in world-level competitions, and illustrates the latest research in cryptanalysis addressing implementations vulnerabilities. The standardization process is also considered, and the main conclusion is that even if an algorithm is standardized this does not guarantee the security of the algorithm in front of new attacks, neither in front of implementation flows.

The contributions of this chapter are:

- State-of-the-art survey of cryptographic selection competitions;
- Analysis of new trends in cryptanalysis, mainly side-channel attacks addressing cryptographic implementations vulnerabilities;
- Focusing on fault attacks, analysis of fault injection methods and fault analysis attacks.
- Analysis of security standards, their evolution and standardization bodies.

This chapter underlines the permanent need for evaluation and evolution of both cryptographic algorithms and their implementations. Starting from these findings, the rest of the thesis addresses the need for error detection mechanisms in cryptographic implementations. These requirements for error detection are meant to improve the security of cryptographic implementations.

# 4. MODES OF OPERATION AND THEIR SECURITY IN CASE OF FAULTS

Modes of operations are used to encrypt messages of arbitrary length. To be useful, a mode must be at least as secure and as efficient as the underlying cipher. Modes may have properties in addition to those inherent in the basic cipher.

The standard DES modes of operation have been published in FIPS 81 [34] and as ANSI X3.106 [38]. A more general version of the standard [35] generalized the four modes of DES to be applicable to a block cipher of any block size. In [39] an additional confidentiality mode is added, the *Counter Mode (CTR)*, for use with any FIPS-approved block cipher.

In this chapter we address the five modes of operations standardized by NIST, recommended to be used with AES. After presenting the five modes of operation, the error propagation for these modes is analyzed. Afterwards in the chapter we analyze the effect of faults in three implementation recommendations of the CTR mode: NIST Recommendations [39], the IETF RFC (Request for Comments) no. 3686, the standard regarding IPsec [36] and the ATM Security Specifications [37].

It is shown in the previous chapters that faults can reduce the overall confidentiality of the cryptosystems. In this chapter we address the vulnerabilities of the modes of operation. We show in this chapter that faults may generate vulnerabilities in case of counter mode. If faults are affecting one or more bits of the encrypting sequences (called counter blocks in CTR mode) then, consecutive plaintext blocks are XOR-ed with the same counter block to generate the ciphertext blocks and this, independent of the key value. To overcome failures and to reduce the vulnerabilities of Counter Mode, several implementation recommendations are introduced and solutions are identified and presented in this chapter.

## 4.1. Modes of operation

One of the conditions for a secure encryption requires that the plaintext contains no pattern, as these will leak to the ciphertext. For this condition to be satisfied independent of the plaintext to be encrypted, the block ciphers are used in special ways, called *modes of operation*. In 1980 four modes of operation were standardized in [34] *ECB (Electronic Code Block)* mode (this mode does not hide patterns); *CBC (Cipher Block Chaining)* mode; CFB *(Cipher FeedBack)* mode; *OFB (Output FeedBack)* mode. For example, for implementation, the plaintext is split into blocks and those blocks are encrypted, for CBC each block of plaintext is XORed with the previous ciphertext block before being encrypted, etc.

In 2001 the NIST added a fifth mode, the *Counter Mode (CTR)*, all of them recommended as modes of operation to be used with AES [39]. The counter mode has efficiency advantages over the previous modes of operation (ECB, CBC, CFB, and OFB). Used with large block size ciphers such as AES, CTR mode is not weakening the security of the algorithm.

In the following sections the five standardized operation modes are presented.

### 4.1.1. ECB (Electronic Code Book) mode

The simplest mode is the ECB (Electronic Code Book) mode. This mode is not hiding patterns. The plaintext is divided into n-bit blocks (in total m blocks), and is encrypted block by block (figure 4.1.).

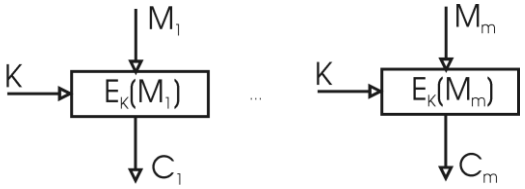


Figure 4.1. Electronic Code Book mode

The decryption also operates on individual blocks:

$$C_i = E_K (M_i) \text{ and } M_i = D_K (C_i)$$

where $M_i$ are blocks of the plaintext (message) M, and $C_i$ are blocks, of same length, of the ciphertext C, and $K$ is the key. (The same notation is used for the following operation modes).

Errors in the ciphertext do not propagate beyond the block boundaries (as long as these can be recovered). However, the ECB mode does not hide patterns (such as repetitions) in the plaintext), as these are copied to the ciphertext. As such, this mode can only be used in cases where the plaintext is already random, such as the encryption of cryptographic keys.

ECB mode is as secure as the underlying block cipher. Because plaintext patterns are not masked, each identical block of plaintext gives an identical block of ciphertext. The plaintext can be easily manipulated by removing, repeating, or interchanging blocks.

The speed of each encryption operation is identical to that of the block cipher. ECB allows parallelization for higher performance. However, no preprocessing is possible before a block is available (except for key setup).

### 4.1.2. CBC (Cipher Block Chaining) mode

The standard mode of operation of a block cipher is the CBC (Cipher Block Chaining) mode. In this mode the different blocks are coupled by adding modulo 2 (XOR–ed) to a plaintext block the previous ciphertext block before the encryption operation:

$$C_i = E_K (M_i \oplus C_{i-1}) \text{ and } M_i = D_K(C_i) \oplus C_{i-1}.$$

This mode "randomizes" the plaintext, and hides patterns.

An the *initial value IV* is used to enable the encryption of the first plaintext block (figure 4.2.). By varying this initial value, the same plaintext is encrypted into a different ciphertext under the same key. Sender and receiver have to agree on the value of *IV*.

The CBC mode has limited error propagation: errors in the $i^{th}$ ciphertext block will twist the $i^{th}$ plaintext block completely, and will be copied into the $i + 1^{th}$ plaintext block. The CBC mode allows for random access on decryption: if necessary, one can decrypt only a small part of the ciphertext.

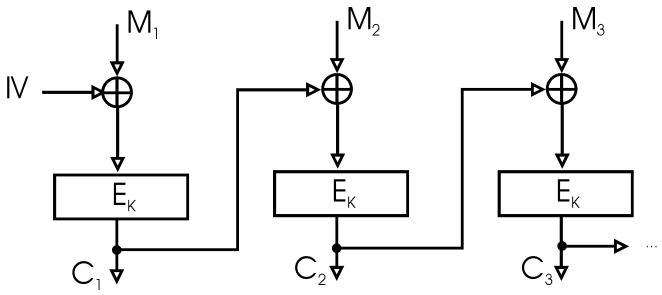


Figure 4.2. Cipher Block Chaining mode

CBC mode is as secure as the underlying block cipher against standard attacks. In addition, any patterns in the plaintext are masked by the XORing of the previous ciphertext block with the plaintext block. The plaintext cannot be directly manipulated except by removal of blocks from the beginning or the end of the ciphertext. The initialization vector should be different for any two messages encrypted with the same key and is preferably randomly chosen. It does not have to be encrypted and it can be transmitted with (or considered as the first part of) the ciphertext.

The speed of encryption is identical to that of the block cipher, but the encryption process cannot be easily parallelized, although the decryption process can be.

### 4.1.3. CFB (Cipher FeedBack) mode

In CFB mode (see Figure 4.3.), the previous ciphertext block is encrypted and the output produced is combined with the plaintext block using XOR to produce the current ciphertext block. It is possible to define CFB mode using feedback that is less than one full data block. An initialization vector *IV* is used as a "seed" for the encryption.

$$C_i=E_k(C_{i-1})\oplus M_i,\ M_i=E_k(C_{i-1})\oplus C_i$$

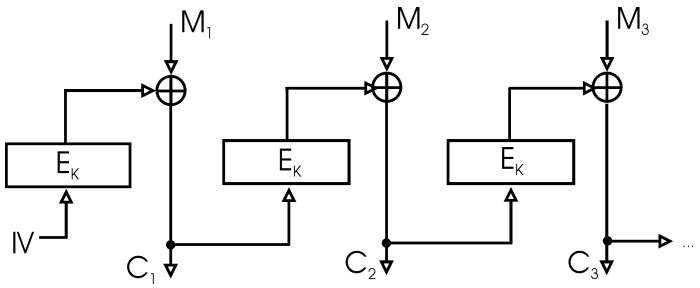


Figure 4.3. Cipher Feedback mode

CFB mode requires a parameter, the length of a segment, where the length $s$ is $1 \le s \le n$, where $n$ is the length of an encryption block. As such the encryption/decryption functions are executed on a number of bits larger than $s$, but only the most significant $s$ bits are XOR-ed with the $s$ bits of plaintext/ciphertext to produce $s$ bits of ciphertext/ plaintext. The value of $s$ is sometimes incorporated in the name of the mode e.g. the 8-bit CFB mode, or 128-bit CFB mode [39]. The

figure 4.3 corresponds to the 128-bit CFB mode.

CFB mode is as secure as the underlying cipher and plaintext patterns are masked in the ciphertext by the use of the XOR operation. Plaintext cannot be manipulated directly except by the removal of blocks from the beginning or the end of the ciphertext.

The speed of encryption is identical to that of the block cipher, and the encryption process cannot be easily parallelized.

### 4.1.4. OFB (Output FeedBack) mode

OFB mode (see Figure 4.4) is similar to CFB mode except that the blocks XORed with each plaintext block are generated independently of both the plaintext and ciphertext. An initialization vector $IV=s_0$ is used as a "seed" for a sequence of data blocks $s_i$, and each data block $s_i$ is derived from the encryption of the previous data block $s_{i-1}$. The encryption of a plaintext block is derived by taking the XOR of the plaintext block with the relevant data block.

$$C_i=M_i \oplus s_i\ ,\ M_i=C_i \oplus s_i\ ,\ s_i=E_k(s_{i-1})$$

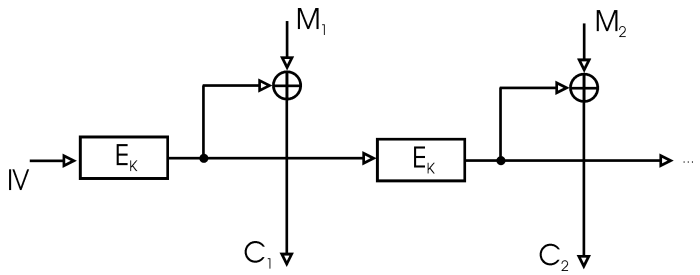


Figure 4.4. Output Feedback mode

Feedback widths less than a full block are not recommended for security reasons [8]. OFB mode has an advantage over CFB mode in that any bit errors that might occur during transmission are not propagated to affect the decryption of subsequent blocks.

A problem with OFB mode is that the plaintext is easily manipulated. Namely, an attacker who knows a plaintext block $M_i$ may replace it with a false plaintext block $x$ by XORing $M_i \oplus x$ to the corresponding ciphertext block $C_i$. There are similar attacks on CBC and CFB modes, but in those attacks some plaintext block will be modified in a manner unpredictable by the attacker. Yet, the very first ciphertext block (that is, the initialization vector) in CBC mode and the very last ciphertext block in CFB mode are just as vulnerable to the attack as the blocks in OFB mode. Attacks of this kind can be prevented by using for example a digital signature scheme or a MAC scheme.

The speed of encryption is identical to that of the block cipher. Even though the process cannot easily be parallelized, time can be saved by generating the keystream before the data is available for encryption.

### 4.1.5. CTR (Counter) mode

As mentioned earlier, in [39] a fifth mode of operation for encryption was added.

The Counter (CTR) mode is a confidentiality mode that uses encryption of a

set of input blocks, called counters, to produce a sequence of output blocks that are XORed with the plaintext to produce the ciphertext, and vice versa. The sequence of counters must have the property that each block in the sequence is different from every other block. This condition is not restricted to a single message: across all of the messages that are encrypted under the given key, all of the counters must be distinct. In this recommendation, the counters for a given message are denoted $ctr_1$ , $ctr_2$ , … , $ctr_m$.

Given a sequence of counters, $ctr_1$ , $ctr_2$ , … , $ctr_m$ , the CTR mode is defined as follows:

CTR Encryption:

$$C_i = M_i \oplus E_K(ctr_i)$$

CTR Decryption:

$$M_i = C_i \oplus E_K(ctr_i).$$

In CTR encryption, each counter block is encrypted and the resulting output blocks are XORed with the corresponding plaintext blocks to produce the ciphertext blocks. For the last block, which may be a partial block of $u$ bits, while the length of the blocks is $n$, the most significant $u$ bits of the last output block are used for the exclusive-OR operation; the remaining $n$-$u$ bits of the last output block are discarded.

In CTR decryption, also, each counter block is encrypted and the resulting output blocks are XORed with the corresponding ciphertext blocks to recover the plaintext blocks. For the last block, which may be a partial block of $u$ bits, the most significant $u$ bits of the last output block are used for the exclusive-OR operation; the remaining $n$-$u$ bits of the last output block are discarded.

In both CTR encryption and CTR decryption, the encryption functions can be performed in parallel; similarly, the plaintext block that corresponds to any particular ciphertext block can be recovered independently from the other plaintext blocks if the corresponding counter block can be determined. For faster encryption, the encryption functions can be applied to the counters prior to the availability of the plaintext or ciphertext.

The CTR mode is illustrated in figure 4.5.

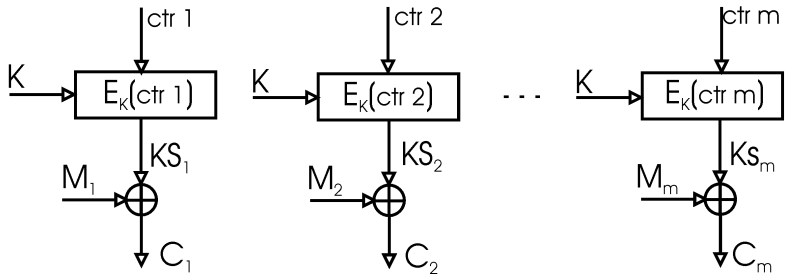


Figure 4.5. The CTR mode.

The specification of the CTR mode requires a unique counter block for each plaintext block that is ever encrypted under a given key, across all messages. Otherwise, if, a counter block is used repeatedly, then the confidentiality of all of the plaintext blocks corresponding to that counter block may be compromised. In particular, if any plaintext block that is encrypted using a given counter block is known, then the output of the forward cipher function can be determined easily

from the associated ciphertext block. This output allows any other plaintext blocks that are encrypted using the same counter block to be easily recovered from their associated ciphertext blocks [39].

## 4.2. Modes of operation and bit errors

If there are bit errors in any ciphertext block (e.g. due to communication), then the decryption of that ciphertext block is incorrect, i.e., it differs from the original plaintext block. The effects of error propagation (bit errors, insertion, or deletion of bits) in ciphertext blocks (or part of them), counter blocks, and *IV*s on the modes introduced in this chapter are presented in this section. By bit error is understood the substitution of a '0' bit for a '1' bit, or vice versa.

Concerning the *bit errors* in the decrypted ciphertext block, they occur in the same bit position(s) as in the ciphertext block for the case of CFB, OFB, and CTR modes, while the other bit positions are not affected. In the ECB and CBC modes, a bit error may occur, independently of the error position, in any bit position of the decrypted ciphertext block, depending on the underlying block cipher [39].

Regarding *error propagation*, for the ECB, OFB, and CTR modes, bit errors within a ciphertext block do not affect the decryption of any other blocks. In the CBC mode, any bit positions that contain bit errors in a ciphertext block will also contain bit errors in the decryption of the succeeding ciphertext block; the other bit positions are not affected. In the CFB mode, bit errors in a ciphertext segment affect the decryption of at least one successive ciphertext segment, in any bit position.

*Errors in counter blocks and initial values (IV)*. For the CTR mode, a bit error in a counter block is determining bit errors in any bit position of the decryption of the corresponding ciphertext. For *IV*, bit errors in the OFB mode, affect the decryption of every ciphertext block. In the CFB mode, bit errors in the *IV* affect, at a least, the decryption of the first ciphertext segment, and possibly successive ciphertext segments, depending on the bit position of the rightmost bit error in the *IV*. Such bit errors may occur, in any bit position of the affected ciphertext blocks for both the OFB and CFB modes, with an expected error rate of fifty percent. In the CBC mode, if bit errors occur in the *IV*, then the first ciphertext block will be decrypted incorrectly, and bit errors will occur in exactly the same bit positions as in the *IV*; the decryptions of the other ciphertext blocks are not affected.

*Vulnerabilities*. If the integrity of the *IV* is not protected for the CBC mode, the decryption of the first ciphertext block is vulnerable to the (deliberate) introduction of bit errors in specific bit positions of the *IV*. In the case of OFB and CTR modes, the decryption of any ciphertext block is vulnerable to the introduction of specific bit errors into that ciphertext block if its integrity is not protected. This holds for the ciphertext segments in the CFB mode; however, for every ciphertext segment except the last one, the existence of such bit errors may be detected by their randomizing effect on the decryption of the succeeding ciphertext segment.

In figure 4.6, based on [39], the effects of bit errors in ciphertext respectively *IV* are presented during decryption of ciphertext block $C_j$ (and successive blocks). *Random bit errors* occur independently in any bit position with an expected probability of ½, while *specific bit errors* occur in the same bit position(s) as the original bit error(s).

The *deletion or insertion of bits* into a ciphertext block (or segment) destroys the synchronization of the block (or segment) boundaries, as such bit errors may occur in the bit position of the inserted or deleted bit, and in every subsequent bit position. Therefore, the decryptions of the subsequent ciphertext

blocks (or segments) will almost certainly be incorrect until the synchronization is restored [39].

| Mode | Effect of bit errors in $C_j$ during decryption of $C_j$ and succesive block(s) | Effect of bit errors in IV during decryption of $C_j$ and succesive block(s) |
|---|---|---|
| ECB | random bit error for decryption of $C_j$ | not applicable |
| CBC | random bit error for decryption of $C_j$<br>specific bit error for decryption of $C_{j+1}$ | specific bit error for decryption of $C_1$ |
| CFB | specific bit error for decryption of $C_j$<br>random bit error for decryption of $C_{j+1}$, … | random bit error for decryption of $C_1$, … |
| OFB | specific bit error for decryption of $C_j$ | random bit error for decryp. of $C_1$, …, $C_n$ |
| CTR | specific bit error for decryption of $C_j$ | bit errors in the $j^{th}$ counter block result in random bit error for decryption of *Cj* * |

**in Section 4.3 vulnerabilities due to injection of faults in counter blocks are analysed and the effect of bit errors is presented*

Figure 4.6. Summary of effect of bit errors on decryption

## 4.3. Counter Mode standardized implementations

Diffie and Hellman introduced Counter Mode in 1979 [40]. Different institutions or consortia such as NIST [39] or the ATM Forum [37] standardized Counter Mode. The advantages [41] of this mode compared to others are:

- high speed implementations. CTR is fully parallelizable; Also pre-processing can be used to increase speed;
- low rate of error propagation;
- arbitrary length of the messages;
- all these without weakening the security.

The encryption and decryption processes using counter mode of operation are presented in figure 4.7. We use the following notation:

*n* - number of bits of the encryption/decryption block,

*l* - length of a message encrypted with the same key K,

*m* is *l/n* rounded up to the nearest integer; number of blocks to be encrypted,

*u* - value smaller than *n* so that

$$l=n^* (m-1)+u$$

A set of input blocks (of length *n*), called *counter blocks (ctr 1, …, ctr m)*, are encrypted using the *key K* to produce a sequence of output blocks, called *key stream blocks ($KS_1$, …, $KS_m$)*, which are XOR-ed with the *plaintext blocks ( $M_1$, …, $M_m$)* to produce the *ciphertext blocks ($C_1$, …, $C_m$)*. With $E_K(ctr\ j)$ is denoted encryption of the counter block *ctr j,* with the given key *K.* For decryption, the ciphertext is XOR-ed with the key stream to produce the plaintext. In this way the same function is used for encryption and decryption process, only the inputs are different. This represents an advantage for hardware implementation - the same

hardware can be used for encryption and for decryption even if the algorithm is not identical for both operation (encryption/decryption) - such as in Rijndael (the AES selected standard) [22].

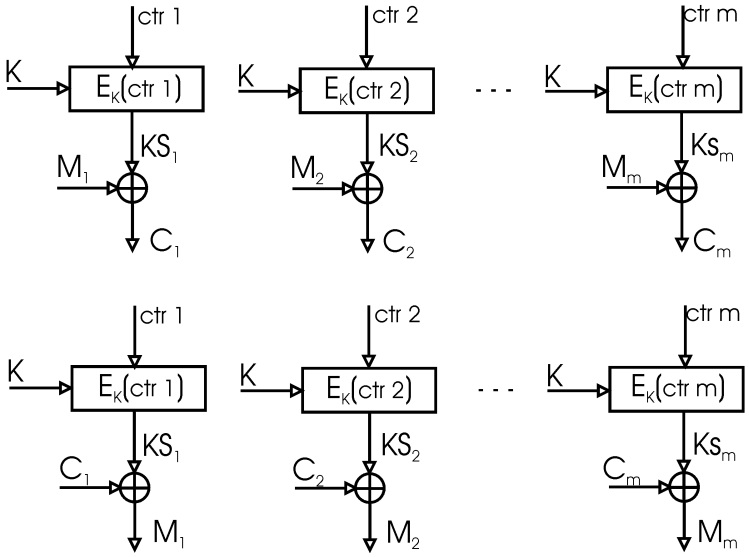


Figure 4.7. Counter mode. Encryption and decryption

Equations (4.1) and (4.2) contain the formulas for encryption and decryption respectively, where $E_K$ represents encryption function with the key $K$.

$$KS_j=E_K(\text{ctr } j), \quad \text{for } j=1, 2, \ldots m;$$
$$C_j= M_j \oplus KS_j, \quad \text{for } j=1, 2, \ldots m-1;\ C_m= M_m \oplus \text{MSBu}(KS_m). \qquad (4.1)$$

$$KS_j=E_K(\text{ctr } j), \quad \text{for } j=1, 2, \ldots m;$$
$$M_j= C_j \oplus KS_j, \quad \text{for } j=1, 2, \ldots m-1;\ M_m= C_m \oplus \text{MSBu}(KS_m). \qquad (4.2)$$

If the last block of the message encrypted with the same key has a length $u$ smaller than the block size $n$, then only the first $u$ bits are XOR-ed with the first $u$ bits of the key stream $KS_m$, the rest of them being discarded.

From figure 4.7 and from the equations can be noticed that the encryption function can be executed for the counters before the plaintext is available for producing the ciphertext.

The sequence of counter blocks must have the property that each block is different from others while the same given key is used. If this requirement is not satisfied, then, the confidentiality of all the plaintext blocks encrypted with the same counter block may be compromised [41].

So, the security of the encryption can be reduced in case that more plaintext blocks are encrypted with the same counter block.

### 4.3.1. Standard Incrementing Function in NIST Recommendations

The function used for generation of the counter blocks has to satisfy the uniqueness requirements, meaning that for the same key, all counter blocks should

be different.

Starting from an initial counter block *ctr1*, the successive counter blocks are derived by applying an *incrementing function*.

In [39] this is called *Standard Incrementing Function*. The Standard Incrementing Function can be applied to entire block or to part of a block. If $p$ is the number of bits in the part to be incremented $(p < n)$, and $x<2^p$ a positive integer, then the function takes $[x]_p$ (the binary representation of the last $p$ bits of the integer $x$) and returns $[x+1 \bmod 2^p]_p$.

An example with small values of $p=5$ and $n=8$ is given in [39]. The symbol * represents an unknown bit in the example, and ***11110* is the initial value, which is incremented to generate the rest of the counter blocks. After four applications of the Incrementing Function the output is the following:

* * * 1 1 1 1 0
* * * 1 1 1 1 1
* * * 0 0 0 0 0
* * * 0 0 0 0 1
* * * 0 0 0 1 0.

This function satisfies the uniqueness requirements in case of $m<=2^p$ blocks encrypted with the same key. The recommendations are not restrictive. There is also mentioned that in case of a non–zero initial string, a linear feedback shift register can be used.

The next sections present two other modes for implementing of the Incrementing Function for the counter block. The first one is using 128 bits blocks and the AES encryption algorithm in an Internet Draft concerning IPsec [36] and the second one, is the one used in the ATM Security Specifications [37].

### 4.3.2. Counter Mode and IPsec

In [36] an Internet Engineering Task Force's RFC is presented, and it describes the use of AES Counter Mode of operation (called here AES-CTR). It contains also the explicit initialization vector as an *IPsec Encapsulating Security Payload (ESP)* confidentiality mechanism.

It also shows the need for a unique combination of initial value and key, and the requirement that the same counter block is not repeated during the use of a key.

- *Counter Block Format.* The counter block contains 128 bits (see figure 4.8). The components of the counter block are as follows:
- *nonce*, a single use value field of 32 bits. It is assigned at the beginning of the security association;
- *initial vector*, a field of 64 bits, chosen only once for a given key;
- *block counter*, the last 32 bits of the counter block, starts with a value of one and is incremented to generate the next counter blocks.

The *block counter* field starts with the value of one and is incremented to generate the subsequent field of the counter block.

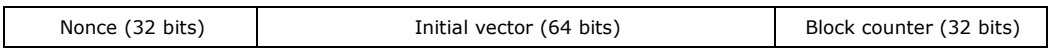


Figure 4.8. IPsec counter block format

This assures $2^{32}-1$ distinct counter blocks, or 4,294,967,295 blocks, which is considered to be sufficient to handle IPv6 requirements.

The IETF RFC [36] contains also 9 test vectors. The test vectors contains maximum the first 3 consecutive counter blocks.

Every time when a security association is established or a key is changed (meaning new nonce or initial vector are established between parties) the initial value for the least significant 32 bits is set to 1 and then it is incremented till a new association or key is established.

### 4.3.3. Counter Mode in ATM Security Specifications

ATM Security Specifications [37] presents the utilization model for the counter mode with any 64-bit block encryption algorithm. Version 1 of the specification was published in 1999 and Version 1.1 in 2001. The part relevant for our dissertation has no major changes.

In [37] the counter mode is considered the most efficient mode of operation in ATM encryption due to the parallel encryption capabilities.

*State Vectors Fields.* The counter blocks are called in [37] *State Vectors* (*SVs*). In order to ensure unique key stream value for each block that is encrypted with the same key, each State Vector contains fields with various counters and a Linear Recurring Sequence.

The State Vector has 64 bits belonging to five fields (figure 4.9). The fields of the State Vector are as follows:

- Galois LFSR, 21 bits;
- Initiator/Response (I/R) bit;
- Sequence Number, 4 bits;
- Segment Number, 3 bits;
- Jump Number, 35 bits.

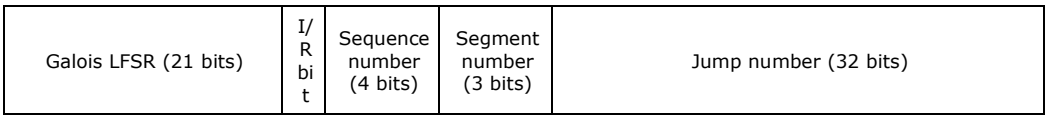


Figure 4.9. ATM state vector fields

The first field is composed of a *Galois Linear Feedback Shift Register* (LFSR). The *Galois* implementation of the shift register is used. The LFSR is pre-set back to initial value at each resynchronization of the communication pairs. The maximum interval of time between resynchronizations determinates the selection of a 21-bit size LFSR, generating $2^{21}$-1 values, meaning 2,097,151 distinct blocks.

The *I/R bit* is used to avoid the same cipher text to be used with the original message and with the response message in case of the same key and SV used in duplex connections. This determines that the responder's key stream to be different, so enclosing the original plaintext would produce different ciphertext.

The *Sequence Number* bits are set to different values depending on the context of use (AAL1 connections, AAL3/4 connections or other connections).

The *Segment Number* is a 3-bit field that defines which 64-bit segment within the payload is encrypted/decrypted. (The 384-bit ATM cell payload is segmented into 6 segments of 64 bits for encryption and decryption). The LFSR is constant for the entire cell payload.

The *Jump Number* starts from all zeros and it is incremented each time a resynchronization occurs or in case of AAL5 with each end-of-message cell. The 35-bit field allows $2^{35}$-1 resynchronizations without repetition and is incremented as

binary counter.

From all 64 bits of State Vector, only the Jump number requires to be transmitted to the receiver during a resynchronization or key changeover. The other fields are preset to their default values. The generation of the next counter block between resynchronizations is mainly based on the output of the LFSR. Even if at the resynchronization the LFSR has the same initial value, the Jump number (incremented at every resynchronization) gives the differentiation. The specifications contain also a requirement. The jump number should be always greater than the previous jump number. If the new Jump number is less or equal to the previous Jump number, then this is rejected and considered as an error condition.

## 4.4. Faults and their impact on security in case of Counter Mode

In context of secure encryption algorithms such as AES, the operation modes should not reduce the overall cipher security. As mentioned in section 4.3, the main concern for Counter Mode of operation is regarding the generation of the counter. The counter should generate a unique value for all messages encrypted with the same key. In this section the security of the Standard Incrementing Functions presented in section 4.3 are analyzed.

We consider the following sequence (adding one more counter block to the initial sequence from section 4.3.1):

ctr 1 = * … * 1 1 1 1 0
ctr 2 = * … * 1 1 1 1 1
ctr 3 = * … * 0 0 0 0 0
ctr 4 = * … * 0 0 0 0 1
ctr 5 = * … * 0 0 0 1 0
ctr 6 = * … * 0 0 0 1 1

It is easy to notice that the Hamming distance between each two pairs of counters *(ctr1, ctr2), (ctr3, ctr4), (ctr5, ctr6)* etc. is 1. This is true for all pairs of (*even, odd*) counter block values.

In this example we consider that the Standard Incrementing Function is applied to the least significant bits of the initial value counter (as it is also in [39] and [36]). If the positioning is different, the example is valid also, but we call the least significant bit – the one situated on the least significant position of segment affected by the incrementing function.

If a fault occurs on the least significant position of the counter, denoted *f*, then we have the following sequence:

ctr 1 = * … * 1 1 1 1 f
ctr 2 = * … * 1 1 1 1 f
ctr 3 = * … * 0 0 0 0 f
ctr 4 = * … * 0 0 0 0 f
ctr 5 = * … * 0 0 0 1 f
ctr 6 = * … * 0 0 0 1 f

where *f* can be 0 or 1. Which is the value of *f* is not important, as long as it is the same for consecutive *(ctr j, ctr j+1)* pairs of block counter values (with *j* being odd, and the value of *ctr j* in this context being even). In this case the encryption process is generating the following sequence of ciphertext blocks:

$$KS_j=E_K(\text{ctr j}),\ KS_{j+1}=E_K(\text{ctr j+1}),\ \ldots$$

$$M_j \oplus KS_j,\ M_{j+1} \oplus KS_{j+1},\ M_{j+2} \oplus KS_{j+2},\ M_{j+3} \oplus KS_{j+3},\ldots$$

with *ctr j=ctr j+1* and further $KS_j=KS_{j+1}$ due to the fault *f*. So, a XOR operation between two consecutive ciphertext blocks will give:

$$C_j \oplus C_{j+1} = M_j \oplus M_{j+1},\ \text{with j odd;}$$

independent of the key *K*.

This is not revealing directly the plaintext, but if patterns exist, the plaintext could be extracted. And, as mentioned in section 2, the uniqueness requirement of the Incrementing Function generating the counter blocks is not fulfilled any more.

In hardware implementation this fault can be generated cutting the wire connection of the LSB of the Counter Module to the Encryption Module in figure 4.10. The same effect can be obtained due to trap implementation.

The fault assumption can be extended. If the two least significant bits are faulty, then four consecutive plaintext blocks are XOR-ed with the same key stream block, independent of the key.

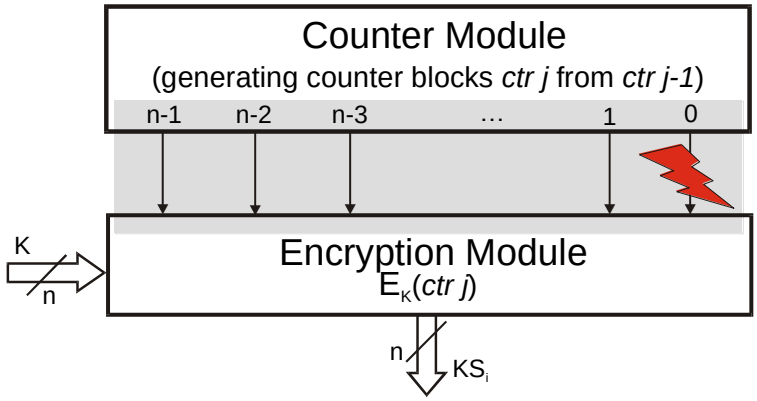


Figure 4.10. Fault on the LSB of the Counter module output (n =the size of the encryption/decryption block).

We consider that a pair sender/receiver is communicating encrypted data using CTR mode. From the model of fault presented before, there are different situations:

- no fault present;
- fault(s) present at one side. In this case errors are detected, and the receiver will have one of the following situations after the decryption process:
  - every second block of decrypted data is unintelligible – due to a single fault at the least significant position of the counter segment;
  - every plaintext block is followed by three unintelligible blocks – due to a number of two faults, etc;
- same fault(s) present on both sides. In this case the presence of fault(s) is undetected; no errors are detected;

- different faults present at the two sides. Errors are detected; at the receiver side, certain (or all) blocks are unintelligible.

From these situations, the most dangerous one is the case of the same fault(s) present at both sides, when no error is detected due to such faults. This can be due to a malicious implementation of the encryption mode of operation if all the parties involved in the secure communication are using the same corrupted implementation.

## 4.5. How to avoid vulnerabilities for the modes of operation

In this section we propose solution to avoid vulnerabilities of Counter Mode in front of our model of fault and then we analyze the case of other modes for the same model of fault.

*Counter Mode.* In NIST Recommendations and IPsec Specifications the least significant bit of the counter blocks assures the differentiation of the key stream blocks which are XOR-ed with the plaintext blocks to produce the ciphertext blocks. If a fault affects the least significant bit, then two consecutive plaintext blocks are using the same key stream block for encryption. The confidentiality of all the plaintext blocks encrypted with the same counter block can be compromised. The change of the key is not removing the problem.

However there are two options to avoid such vulnerability. An option would be to use other counter generator, for instance instead of consecutive values, to use LFSR.

The use of a LFSR is recommended to avoid the vulnerability of a single or multiple bit faults based on the model presented in section 4.4. We consider that the ATM Security Specifications are more reliable in context of such a fault. The NIST recommendation [39] mentions but not establishes the use of LFSR for Standard Incrementing Function.

Another mode to avoid this model of fault is to test if the output of the Counter module (Fig. 2.10) used for encryption of current plaintext block is different from the one used for previous plaintext block. However, one cannot be sure that the fault cannot occur just after the point of test, before the input of encryption module.

*Our model of fault in case of other operation modes*. Operation modes such as CBC (Cipher Block Chaining), CFB (Cipher FeedBack) and OFB (Output FeedBack) are designed to hide existing patterns in the plaintext. In context of these modes, the blocks that are consecutive encrypted are not consecutive values, output of a counter module. If the ECB (Electronic Code Block) mode is used, where blocks of n-bits are encrypted one by one, no counter module is involved. So, in case of these operation modes our model of fault is not causing security concerns and the faults are detectable due to unintelligible blocks at decryption. The effect of error propagation for the operation modes is presented in section 4.2. Other modes of operation (e.g. Statistical Cipher Feedback (SCFB) mode [42]) may be considered for further research.

## 4.6. Conclusions

Operation modes are used in the implementation of block algorithms. The five modes of operation standardized and recommended to be used with standard algorithms are presented in this chapter and the propagation of errors is analyzed.

An error can have different effects: it can determine a single bit error at decryption side, or random bits error depending on the input being affected by fault: plaintext, initial value where applicable.

Also, we present the CTR mode and how it is used based on NIST, IPsec or ATM recommendations. In NIST Recommendations and IPsec Specifications the least significant bit of the counter blocks assures the differentiation of the key stream blocks which are XOR-ed with the plaintext blocks to produce the ciphertext blocks.

We presented a model of fault that can reduce the security of Counter Mode. If a fault affects the least significant bit, then two consecutive plaintext blocks are using the same key stream block for encryption. The confidentiality of all the plaintext blocks encrypted with the same counter block can be compromised. The change of the key is not removing the problem.

Two recommendations are made. The use of a LFSR is recommended to avoid the vulnerability of a single or multiple bit faults based on the model presented in section 4.4. The ATM Security Specifications are more reliable in context of such a fault. The NIST recommendation [39] mentions but not establishes the use of LFSR for Standard Incrementing Function. Another mode to avoid this model of fault is to test if the output of the Counter module (Fig. 2.10) used for encryption of current plaintext block is different from the one used for previous plaintext block.

The main contributions of the chapter are:

- description of the five standardized modes of operation;
- analysis of the bit errors and error propagation for all modes of operation;
- focusing on Counter Mode, analysis of the security of three implementation recommendations, namely NIST recommendations, IPsec specifications and ATM security specifications;
- identification of vulnerabilities and introducing a model of fault that can reduce the security of Counter Mode; analysis of error detection for this model of fault;
- recommendations useful for secure implementations of Counter Mode.

The confidentiality of the Counter Mode presented in specifications (e.g. NIST specifications) can be compromised with the model of fault presented in section 4.4 independent of the key value used for encryption [43]. We have shown that fault detection/tolerance is required to avoid such vulnerabilities.

# 5. FAULT TOLERANCE FOR SECURE IMPLEMENTATIONS OF BLOCK CIPHERS

Fault tolerance is the attribute that enables a system to achieve fault-tolerant operation. A fault-tolerant system is one that can continue to correctly perform its specified tasks in the presence of hardware failures and software errors [44]. Starting from this definition of fault-tolerant system, it is obvious that cryptographic implementations need to be fault-tolerant in order to function correctly. A faulty encryption or decryption would generate faulty output, and this, in case of a message, means unintelligible decrypted messages to the receiver. The request for re-transmission will generate delays and extra cost. But, this is not the only reason why fault tolerance is now needed in cryptographic implementations.

Due to the vulnerabilities that can be generated through fault analysis attack (as seen in previous chapters), the faults can allow the attacker to retrieve e.g. key data (which can be used even later i.e. when no fault is affecting the encryption/decryption process). So, also for security reasons fault tolerance techniques/mechanisms are needed.

In this chapter we identify different techniques proposed for self-testing cryptographic architectures able to detect transient or/and permanent faults injected by an attacker in hardware implemented algorithms.

Trade-off analysis of different resources required for detection mechanisms is also included: the cost of hardware or the delay generated by testing mechanisms vs. the type of detected faults. However given the consequences of a successful attack (which can retrieve key information with a quite low cost), the extra cost generated by detection and avoidance mechanisms is reasonable.

A case study is given for Triple-DES, an algorithm still used in many protocols (e.g. IPsec). In this chapter we identify which method is most appropriate in case of Triple-DES algorithm from the perspective of a realistic attack model.

## 5.1. Available mechanisms protecting block ciphers against fault analysis attacks

### 5.1.1. Fault analysis attacks for block ciphers

Even if from a mathematical point of view, and from applying conventional attacks such as linear and differential cryptanalysis, the cryptographic algorithms are proven to be secure, faults can reduce the overall confidentiality of the cryptosystem. Attacks based on *random faults* were announced by Boneh, DeMillo and Lipton [30] in 1996. From this work, Eli Biham and Adi Shamir proposed next year a new attack, based on both *transient faults* and *permanent faults* targeting secret key cryptosystems, DES (Data Encryption Standard) in this case [32]. As DES was the standard at that moment, all analysis was carried out targeting DES.

### 5.1.2. Error detection mechanisms for block ciphers

Different methods have been proposed to overcome the fault analysis attack. Most of these are focusing the probabilistic attacks (where the attacker has little or no control for the injected fault location and type). These methods include *redundancy-based error detection schemes* inspired from already existing fault tolerance techniques. Time-redundancy based concurrent error detection assumes two (several) encryption (or decryption), one after the other, and comparison of the result.

Such methods, in which encryption is done several times and the results are compared (e.g. in case of fault-tolerant smartcard design), are not always suitable [32]. For some fault models, key information can be retrieved based on the comparison, or, if the plaintext register is damaged, a faulty plaintext will be encrypted every time, and the fault will pass undetected.

Another type of redundancy-based technique, where the encrypted message is decrypted and the input value is compared, was proposed by Karri et al. in [45]. This method, called *Concurrent Error Detection (CED)* was applied at algorithm and round level for AES finalists (the five algorithms submitted for NIST evaluation for Advanced Encryption Standard, selected for the second round). The method can be used for symmetric encryption due to the inverse relationship that exists between encryption and decryption at algorithm level, round level and operation level. A trade-off analysis is presented regarding area overhead, performance penalty and fault detection latency for four cases – no fault tolerance (no concurrent error detection schemes), concurrent error detection schemes at algorithm, round and operation level. Based on their implementations, round level concurrent error detection has the most convenient area overhead, performance penalty and fault detection latency balance.

In [46], Bertoni et al., a scheme based on error detection codes (i.e. parity codes) is described and evaluated for the specific case of AES (Rijndael). The method described in the paper proposes the use of a parity bit with each byte element. It requires prediction of the parity bits from one round to the other, being in this way algorithm dependent. This approach has a low cost (limited hardware overhead and short detection latency) compared with concurrent error detection schemes from [45].

The fault coverage for single bit faults is high; however in case of multiple bit faults the coverage is lower.

The methods presented in [45] were considered to be expensive in terms of area overhead or output delay [47]. Karpovsky &co., in [47], used symmetric nonlinear (cubic) *error detection codes* for a fault analysis attack resistant AES implementation, so, for a fault-tolerant implementation.

From the point of view of fault coverage, the redundancy-based techniques are more efficient and easier to generalize even if the cost is higher. Other techniques, using error detection codes, have to be specifically designed for the given cryptographic algorithm.

However all techniques require considerable overhead if fault-tolerant implementations are used for cryptographic algorithms. And, as it can be seen also from this presentation of proposed methods to detect fault analysis attack, this field requires further research.

After presenting a theoretical case study for Triple-DES algorithm in this chapter and another one, with implementation results for MISTY1 in Chapter 6, a critical analysis of the reference papers is included. In these two chapters new

methods, not applied before in security, are introduced. These methods have good implementation results, showing this way that they are suited in the field of fault tolerant security implementations.

## 5.2. Case study. Triple-DES

Even if new algorithms have been developed, which are presented to be more secure (e.g. AES (Rijndael) [22]), a lot of infrastructures are still using some of the 'old' cryptographic algorithms. This is due to certain latency in implementation and integration caused by the cost involved in those procedures. ATM Security Specifications include usage recommendation for Triple-DES [37]. Even NIST allows the use of the Triple-DES also called Triple Data Encryption Algorithm (TDEA); this in conditions in which on one side the agencies are encouraged to use and to implement the faster and stronger algorithm, the AES, and on the other side NIST proposes to withdraw the standard and the recommendations regarding DES [48].

Also, IPSec (the most commonly used protocol when implementing Virtual Private Networks) supports, next to other encryption protocols, Triple-DES [36].

Due to this extensive use of Triple-DES, different implementations could be subjects of attacks.

### 5.2.1. Short presentation of Triple-DES

Triple-DES, was developed to overcome the short key vulnerability of DES algorithm in front of brute force attack.

Triple-DES key consists of three DES keys. This means that the input data is, in effect, encrypted three times using DES algorithm but with 3 keys. The algorithm encrypts the message with the first key, decrypts it with the second and encrypts it again with the third key using the normal DES algorithm.

For a better understanding we use the following notations. *M* denotes the plaintext (the message to be encrypted). The ciphertext is denoted by *C*. The encryption function *E*, operates on *M* to produce *C* using a key *K*. Or, in mathematical notation:

$$E_K(M) = C$$

In the reverse process, the decryption function *D* operates on *C* using a key *K* to produce plaintext *M.*

$$D_K(C) = M, \text{ and so } D_K(E_K(M)) = M.$$

In figure 5.1 a general representation of Triple-DES is given.

In case of Triple-DES we have three keys: *K1*, *K2*, *K3*. So, the encryption respectively the decryption processes are written as below:

$$C = E_{K3}(D_{K2}(E_{K1}(M))).$$

$$M = DK1(EK2(DK3(C)))$$

Different options can be specified for the keys:

- *K1*, *K2* and *K3* are independent keys;
- *K1* and *K2* are independent and *K3* = *K1;*
- K1=K2 = K3.

The last option allows Triple-DES to be backward compatible with DES.

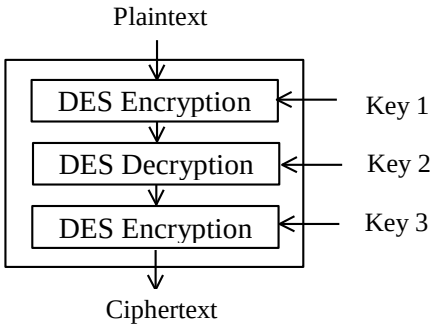


Figure 5.1. Triple-DES - general architecture.

Given the independent nature of initial permutation IP and its inverse $IP^{-1}$, in case of Triple-DES implementation, the entire Triple-DES process of encryption can be described as following:

- 16 rounds encryption with *K1* subkeys,
- 16 rounds decryption with *K2* subkeys,
- 16 rounds encryption with *K3* subkeys.

For Triple-DES the encryption function and the decryption function differ only through the order of the subkeys used inside the rounds. In those conditions, the algorithm can be seen as a 48 round algorithm, using 16 subkeys generated from K1 in order, 16 subkeys generated from K2 in inverse order and 16 subkeys generated from K3 in order.

Given this similarity of Triple-DES with DES algorithm ([34] and others), we are not going to present details regarding DES encryption, decryption or key scheduling algorithm here.

### 5.2.2. Applying error detection methods for Triple-DES algorithm

#### 5.2.2.1. Using CED for Triple-DES

As presented by Karri & co in [45], the *concurrent error detection (CED)* method takes advantage of the inverse relationship between encryption and decryption functions in the symmetric block ciphers. This inverse relationship exists at three levels: algorithm level, round level and operation level.

***Algorithm level CED for Triple-DES*.** This method can be implemented in two modes, depending on the available resources and requirements: *time overhead* or *hardware overhead*. The mode with time overhead requires extra hardware only for storage for the plaintext (till the decryption is complete), for the comparison module and for the switching encryption/decryption mode module. However, from time perspective, the total time increases from the time required to execute 3 x 16 rounds to more than double (2 x (3 x 16) rounds plus comparison time).

The general representation of the algorithm level concurrent error detection mechanism is represented in figure 5.2. An error signal is generated if the comparator detects different values.

The mode with hardware overhead requires a second encryption/decryption module, so, implies 100% hardware overhead. This mode is efficient when a continuous data flow is encrypted/ decrypted, and in this mode, the new plaintext is encrypted in parallel with the decryption of the previous encrypted plaintext. After

the initial delay, the output flow has the same throughput as for the algorithm used without error detection mechanisms. So, for *N* blocks of data, the total time of encryption/decryption *is (N+1) x time to encrypt/decrypt a data block*. This algorithm level CED method can detect both transient and permanent faults as long as the storage of the plaintext is not affected by faults.

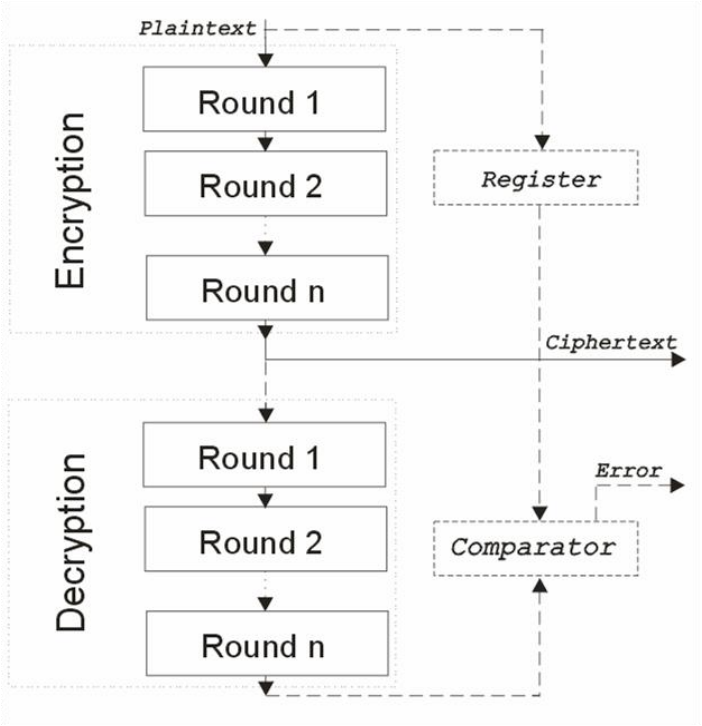


Figure 5.2. Algorithm level error detection. General view.

Regarding *detection latency* (the time between the error occurred and the moment it is detected) this is *2 x (3x16) x time to encrypt/decrypt a round* for both modes (hardware or time overhead).

***Round level CED for Triple-DES*.** Round level CED is similar with the algorithm level CED, only that the inverse function and the comparison are executed after each round. Round level CED is represented in figure 5.3, where E means encryption and D means decryption. There are error signals (*r1,…, rn*) for each round of the algorithms. Given the DES algorithm structure, the only difference between an encryption round and a decryption round is given by the order of the subkeys used for encryption respectively decryption (for encryption from subkey one to sixteen and from decryption from the subkey sixteen to subkey one). In this context the same hardware can be used for both encryption and decryption.

The time required detecting a fault is twice the time required for a round encryption/decryption.

Regarding the hardware/time overhead trade-off, this is similar to algorithm level CED. A 100% overhead is required or for hardware or for time in order to have the other part kept in the original value domain. More details are presented in table 5.1.

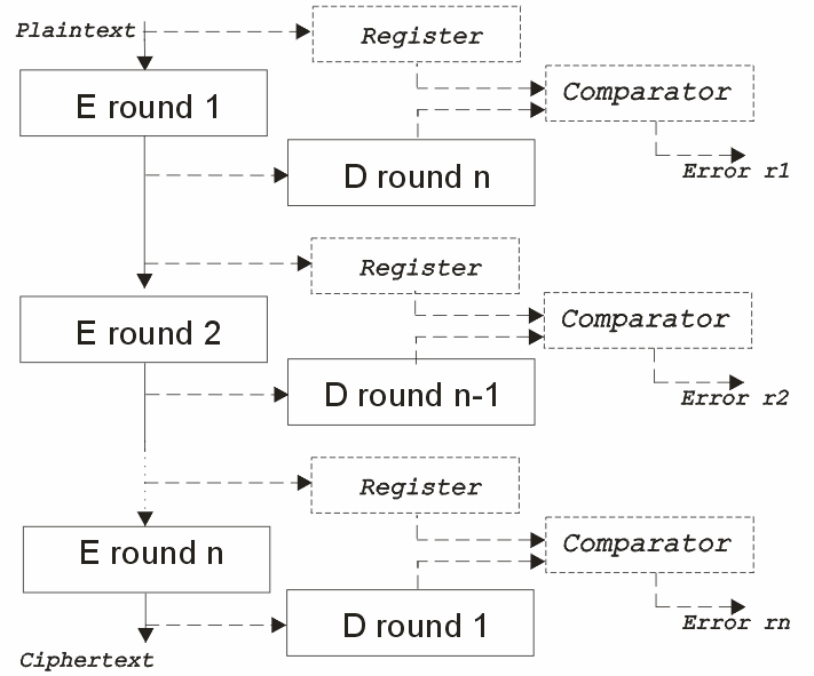


Figure 5.3. Round level error detection. General view.

Round level CED method can detect both transient and permanent faults as long as the storages of the plaintext and intermediate values between rounds are not affected by faults. However, if *only permanent faults* are targeted (such as the ones assumes by the fault model in [32] for Non-Differential Fault Analysis for the attack to be more realistic), CED is not required for all rounds. If same module is used for all rounds of encryption, then, due to the permanent nature of the fault, is enough that the *last round* is using the CED mechanism to detect possible faults. We call this new method applicable for permanent faults, which applies error detection only for the last round, *last round CED*.

***Last round level CED.*** For time overhead mode, last round CED gives a low overhead (2.08 %), equivalent with the time required to encrypt a round, so that the total time to encrypt a block will become *(3 x 16 +1) x time required for a round encryption / decryption*.

This low time overhead is given by the high number of rounds (3 x 16 = 48) used in Triple-DES for encryption/decryption. For hardware overhead mode, this solution is not useful because the 100% hardware overhead is already there, even if the redundant module is used only 2.08 % of the time.

From table 5.1 it can be noticed, that in case that *only permanent fault*s are targeted, the most efficient method is the one using Concurrent Error Detection with time redundancy mode of implementation. Even if the same hardware is used, there are different input values for *the last round respectively error detection round*. The output of the 48$^{th}$ round is decrypted and compared with the output of 47$^{th}$ round, and there is low probability that the output of 2 consecutive rounds to be identical. So the probability to have an undetected permanent fault is low.

If this probability of detection is considered insufficient, the method can be applied to the last 2 rounds. As such, this new error detection method has the lower cost. The only disadvantage is given by the fact that it targets only permanent faults.

Table 5.1. Trade-off analysis for different concurrent error detection methods

| CED level | Overhead mode | Detection latency | Hardware overhead [1] | Time overhead [2] | Total time |
|---|---|---|---|---|---|
| Algorithm | time | 2 x 48 x r | | 100% | 2 x N x 48 x r (for N blocks) |
| | hardware | 2 x 48 x r | 100% | | (N +1) x 48 x r (for N blocks) |
| Round | time | 2 x r | | 100% | 2 x 48 x r (for 1 block) |
| | hardware | 2 x r | 100 % | | (48 +1) x r (for 1 block) |
| Only last round [3] | time | r + 1 [3] | | 2.08% | (48 +1) x r (for 1 block) |
| | hardware | r + 1 [3] | 100 % | | (48 +1) x r (for 1 block) |

[1]plaintext and intermediate results between rounds storage, comparison block, mode selection block not mentioned in the table, but to be considered,

[2]comparison time considered negligible,

[3] allows only permanent faults detection, r= time for a round encryption/decryption.

***Operation level CED for Triple-DES.*** This method relies on the fact that applying input data through an encryption operation and the corresponding decryption inverse-operation yields the original input data [45].

In figure 5.4 the general representation for error detection mechanism at operation level is given.

In case of DES (and Triple-DES), for decryption, the so-called inverse-operation is identical with the encryption operation. The F-function of the round can be splitted in 4 stages: expansion, key mixing, substitution (using the S-boxes) and permutation. However, due to the same order of the operations (for both encryption and decryption) and due to the substitution stage, the decryption operations cannot be executed in parallel with the encryption for the same round. But, decryption operations of the current round could be overlapped with encryption operations of the next round, however reducing the model to the case of round level CED.

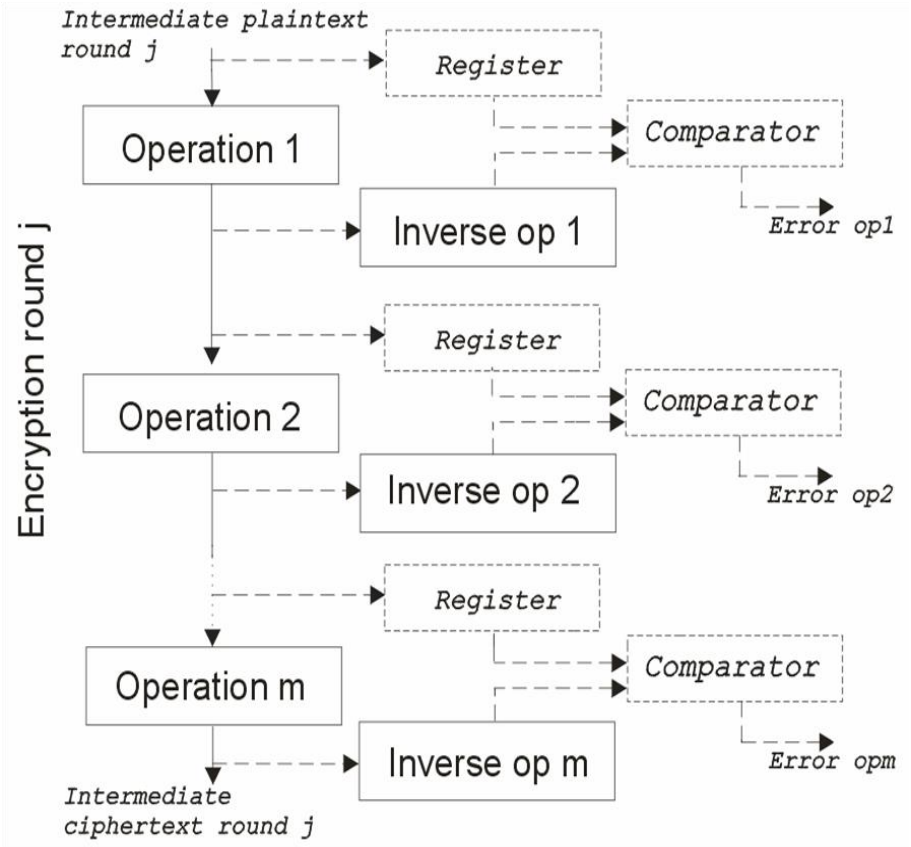


Figure 5.4. Operation level error detection. General view.

**5.2.2.2. Using error detection codes and nonlinear robust codes for Triple-DES**

The method proposed in [46] to associate a parity bit for each byte element requires prediction function for certain operations in the AES algorithm. Due to this, the method is algorithm dependent. Applying this method to the Triple-DES requires generation a prediction function for some operations, e.g. for substitution boxes S-boxes.

The method from [47] relying on nonlinear codes uses a hardware extension for detection purposes. This extension includes a prediction module. As in the case of error detection codes, the error detection architecture is algorithm dependent.

These two methods have less overhead compared to the theoretical CED and coverage for both permanent and transient faults (as it is published in [46], [47]). If the cost assumed by the concurrent error detection methods is above the requirement of the fault detection architecture, then algorithm dependent detection

mechanisms are required.

In this context, using those methods in case of Triple-DES requires development of prediction module/functions according to the specific method.

### 5.2.3. Fault analysis attack resistant key scheduling algorithm for Triple-DES

From the surveyed literature, there is no method presented for the protection of the key-scheduling algorithm.

As presented in [32], for the case of Differential Fault Analysis attack, faulty subkey bit(s) generated by a fault in key scheduling algorithm can be used for cryptanalysis. The number of ciphertext required for such analysis was considered to remain the same as in DFA attack using faults in the encryption/decryption algorithm.

The development of a fault analysis attack resistant key scheduling algorithm is a future research topic.

### 5.2.4. Using complementation property for fault tolerance purposes

DES exhibits the *complementation property*, namely that:

$$E_K(M) = C \Leftrightarrow E_{\overline{K}}(\overline{M}) = \overline{C}$$

where $\overline{K}, \overline{M}, \overline{C}$ are the bitwise complements of *K*, *M* and *C*.

From *security perspective*, the complementation property means that the work for a brute force attack could be reduced by a factor of 2 (or a single bit) under a chosen-plaintext assumption.

However from *fault tolerance perspective* this complementation property can be useful. Encrypting the complement of the plaintext with the complement of the key allows for detection of errors: if both encryptions are executed (encryption of the plaintext using the key and the encryption of complement of the plaintext with the complement of the key) and the ciphertexts are not complements.

#### 5.2.4.1. Specific error detection method for DES/Triple-DES algorithms

Starting from complementation property, error detection mechanisms for DES/Triple-DES can be developed. In this thesis, the new error detection mechanism relying on complementation property is called *complemented error detection*.

In this case (figure 5.5), compared with concurrent error detection technique presented in section 5.2.2, this technique does not need such long detection latency. As long as there are no hardware constraints to run two encryptions in parallel, both direct and complemented plaintexts can be encrypted. In such a case, only the verification - if the ciphertexts are complements - generates delay. So, detection latency is half compared with CED technique.

If from hardware perspective both encryptions (for original plaintext and for complemented plaintext) cannot be executed in parallel, the time overhead is 100% as for the case of concurrent error detection techniques already presented (see table 5.2 for comparison).

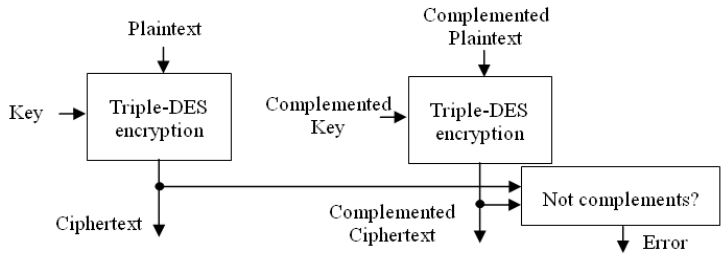


Figure 5.5. Error detection mechanism for Triple-DES relying on complementation property

Based on the theoretical results presented in table 5.2 it is noticeable (last column for total time required to encrypt/decrypt of one block) that complemented error detection brings higher speed for the encryption/decryption when hardware overhead is used.

Table 5.2. Trade-off analysis CED vs. complemented error detection for Triple-DES

| Method | Overhead mode | Detection latency | Hardware overhead (1) | Time overhead (2) | Total encryption/ decryption time for one block |
|---|---|---|---|---|---|
| CED, algorithm level redundancy | time | 2 x 48 x r | | 100% | 2 x 48 x r |
| | hardware | 2 x 48 x r | 100% | | 2 x 48 x r |
| Complemented error detection – algorithm level | time | 2 x 48 x r | | 100% | 2 x 48 x r |
| | hardware | 48 x r | 100 % | | 48 x r |
| CED, round level redundancy | time | 2 x r | | 100% | 2 x 48 x r |
| | hardware | 2 x r | 100 % | | (48 +1) x r |
| Complemented error detection – round level | time | 2 x r | | 100% | 2 x 48 x r |
| | hardware | r | 100 % | | 48 x r |

(1)plaintext storage and intermediate results between rounds storage, comparison block, mode selection block not mentioned in the table, but to be considered,

(2)comparison time considered negligible, r= time for a round encryption/decryption.

### 5.2.4.2. Complementation property and cryptographic algorithms

As mentioned at the beginning of section 5.2.4, for a cryptographic algorithm complementation property is considered a drawback from security perspective. As such, the algorithms designed and published after discovering this property, are tested in order not to have this property.

Thus, other algorithms cannot use the complementation property for error detection purposes. However, a mechanism not used before in cryptographic implementations, relying on redundancy techniques can be deployed for fault detection purposes. Such a mechanism, using *duplication with complementary logic* [44], is described and implemented in Chapter 6.

## 5.3. Conclusions and contributions

After introducing some security considerations, fault analysis attacks developed for block ciphers are shortly presented. These attacks gave reasons for developing fault detection mechanisms inside the cryptographic system. An overview of fault analysis resistant implementations is given in this chapter. Based on these available techniques, in section 5.2, after Triple-DES is shortly presented, a case study is shown.

Because Triple-DES is one of the cryptographic algorithms extensively used in different protocols, we considered that it is important to identify the most appropriate mechanisms to detect the possible injected faults in hardware implementations.

Based on the surveyed literature, keeping in mind the Triple-DES algorithm, we can draw couple of conclusions and point out future research topics. First, if only permanent faults are targeted, concurrent error detection mechanism used for the last round generates the lowest overhead – only 2.08% time overhead (this overhead depends on the number of rounds). This new method that we published in [49] and present here has as advantages simplicity and low overhead in the case that permanent faults are targeted.

If both transient and permanent faults are targeted, and mechanisms with hardware or time overhead lower than 100% are required, algorithm dependent implementations are needed for the specific encryption/decryption rounds of Triple-DES. Also, mechanisms for fault analysis resistant key scheduling algorithm are required due to vulnerability of scheduling algorithm in front of DFA.

DES and Triple-DES have a special characteristic called complementation property. Even if this property is a disadvantage from security perspective, it allows to be used for error detection purposes as we showed here and in [50]. This property cannot be further used for other algorithms, as later developed algorithms are designed not to have such a characteristic.

The contributions of this chapter are:

- overview of available mechanisms for error detection in cryptographic algorithms;
- proposing a new method to reduce the cost of error detection mechanisms, using last round error detection CED;
- suitability analysis of available mechanisms for Triple-DES algorithm;
- cost analysis of the error detection mechanisms in case of Triple-DES;
- analysis of DES/Triple-DES characteristics vs. fault tolerance requirements and error detection costs;
- analysis of complementary property of DES/Triple-DES for fault tolerance purposes;
- identification of possible research paths, e.g. addressing the key scheduling algorithm.

Even if these error detection mechanisms were designed to overcome the effect of fault injection by an attacker, they are useful also from reliability

perspective (also in the case of non-malicious faults).

However, insertion of fault detection mechanisms must be made considering other types of attacks (e.g. power analysis). As such, these mechanisms must not generate vulnerabilities for other types of attacks.

# 6. COST ANALYSIS OF ERROR DETECTION TECHNIQUES FOR MISTY1 CRYPTOGRAPHIC ALGORITHM

Error detection mechanisms using both general and specific techniques were discussed in Chapter 5. However, most of these mechanisms are characterized by a high cost due to redundancy, in hardware or time, estimated up to 100%. Some of these general techniques in case of permanent faults are not able to detect errors (e.g. in the case of a permanent fault if same module is encrypting twice the same plaintext, no error will be detected as both outputs are identical). Other drawbacks of the mechanisms presented in Chapter 5 are due to the requirements for storing keys or plaintext for later comparison; these memory locations can be affected by faults as well. For specific techniques, the main disadvantage is given by the design of prediction logic which is algorithm and method (parity based or error codes based) dependent.

In this chapter we are proposing another technique for error detection in cryptographic implementations. Our method is using *complemented duplication* and it relies on *duplication with complementary logic technique* [44]. This technique is applied to MISTY1 algorithm. A fault tolerant implementation, designed using VHDL language and simulated using ModelSim XE III 6.0d and Xilinx's ISE 8.1i environment targeting Virtex (VIRTEX1000bg560-6) device, is used for evaluation. The results are compared, for evaluation purposes, with other implementations – which are not considering fault detection mechanisms. Considerations regarding parity prediction based error detection mechanisms are also presented in this chapter.

This chapter has the following structure. After introducing MISTY1 algorithm and the reasons for choosing MISTY1 in this work, the experimental environment is presented. The experimental environment is essential for comparison purpose and is similar with the experimental environment used in related research work. Afterwards the error detection mechanism relying on duplication with complementary logic techniques is introduced and the results of simulations for MISTY1 algorithm are presented. These implementation results (hardware costs and maximum path delays) are compared for each component of the algorithm with references and the reduced costs of this fault tolerant implementation are explained. Parity based error detection is also considered and the overhead costs are compared. Related work, already presented in Chapter 5 is analyzed and criticized. The conclusions summarize the findings of this chapter and propose further research paths.

## 6.1. MISTY1 algorithm

MISTY1 detailed specification has been published in Japan in 1996 and presented at the international workshop of Fast Software Encryption in 1997 [51] [52]. MISTY1 has been publicly evaluated without detection of security flaws.

Even if the latest algorithms are designed for 128-bit blocks, MISTY1 with its 64-bit blocks encryption/decryption will coexist with 128-bit block ciphers (due to

i.e. market requirements of low cost implementation and message format compatibility).

MISTY1 is a patented algorithm. However, the owner of its essential patent, Mitsubishi Electric Corporation, has declared that it will give a license without any royalty fee. For more details, see [53].

### 6.1.1. Why MISTY1 algorithm?

For this chapter we chose MISTY1 encryption algorithm to exemplify and apply our specific error detection mechanisms. There are three major reasons for this selection of algorithm.

First, MISTY1 is a 64-bit block cipher with 128-bit key, evaluated and selected in both NESSIE and CRYPTREC evaluation initiatives, as already shown in Chapter 3. Due to these selection and evaluation initiatives there is certitude that the algorithm has a good balance of high security level and performance [54] [27].

Another argument for choosing MISTY1 is given by its design which allows a low cost hardware implementation (MISTY1 is well suited for hardware implementation) [55]. According to its designer Mitsuru Matsui, MISTY1 was designed to be suitable for software and hardware systems as well, particularly for low cost applications [56].

Besides the selections in evaluation processes, and the hardware suitability, another reason for our selection of MISTY1 for this evaluation is given by the usage as confidentiality and integrity algorithm in mobile communication. The *3rd Generation Partnership Project* (3GPP) adopted a variant of MISTY1, which is called KASUMI, as a mandatory algorithm in the confidentiality and integrity mechanisms for GSM mobile communication [55].

### 6.1.2. Encryption with MISTY1

MISTY1 is a Feistel network based on a 32-bit non-linear function; it takes 64-bit plaintext and a 128-bit key. It has a variable number of rounds $n$, where $n$ is multiple of four; with the 8-round version recommended by author [56], version which is most commonly used in real applications. The two main parts of the MISTY1 block cipher, the *data randomizing (data flow)* and the *key scheduling (control flow)* are described in this section.

Data randomizing part of MISTY1 for $n$ rounds is presented in figure 6.1. The 64-bit plaintext $P_{(64)}$ is divided into the left 32-bit string $L_{0(32)}$ and the right 32-bit string $R_{0(32)}$, which are transformed into the 64-bit ciphertext $C_{(64)}$ by means of bitwise XOR operations and sub-functions $FO_i$ (with $i$ taking values from 1 to $n$) and $FL_i$ (with $i$ taking values from 1 to $n+2$). $FO_i$ uses a 64-bit subkeys $KO_i$ and a 48-bit subkeys $KI_i$. $FL_i$ uses a 32-bit subkey $KL_i$. The key scheduling part generates these subkeys from the secret key $K_{(128)}$.

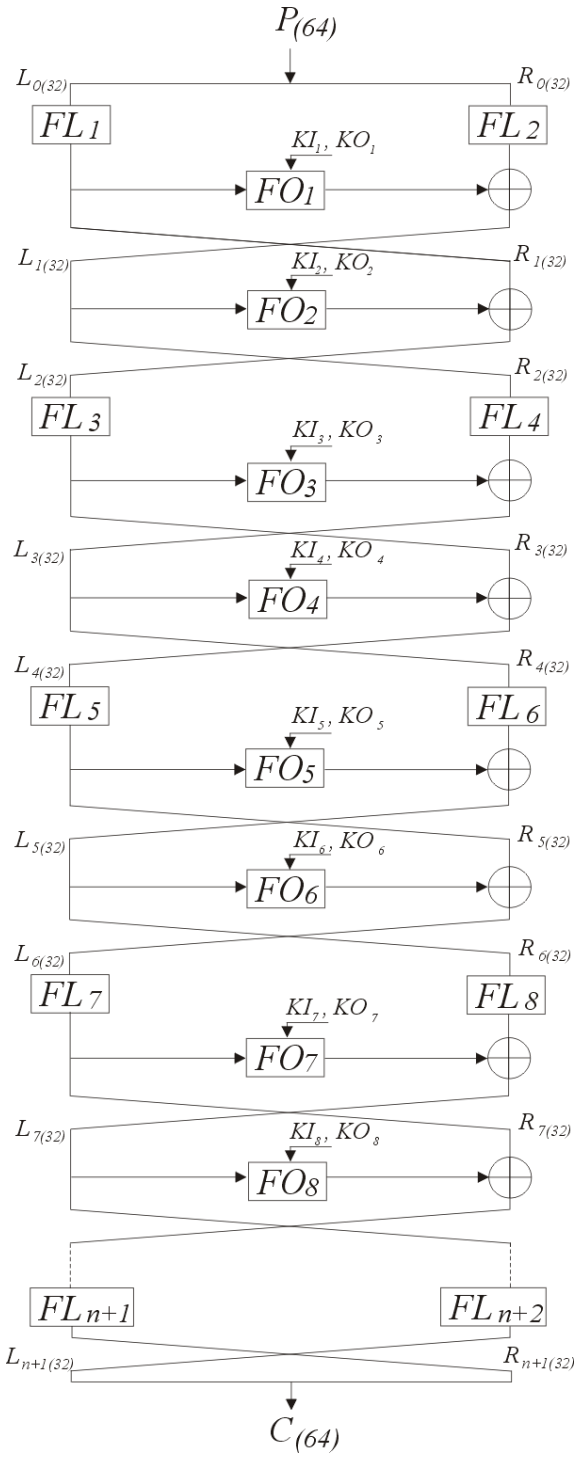


Figure 6.1. Encryption procedure for MISTY1

Data randomizing part of MISTY1 for *n=8* rounds consists of 8 identical stages (rounds) with an additional substage (subround). The rounds contain the function FO. Additionally, even-number round includes the function FL. After the final round, FL functions are used for both halves again. The description of the algorithm is in figure 6.1 and the functions FL, FO and FI are presented below in figure 6.2., 6.3., and 6.4. In the representations, $\cup$ means bitwise *or* operation, $\cap$ means bitwise *and* operation and $\oplus$ means bitwise *exclusive-or* operation.

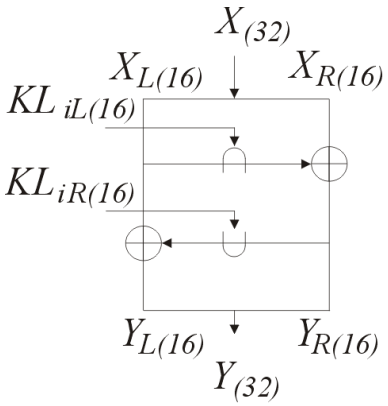


Figure 6.2. Function FL.

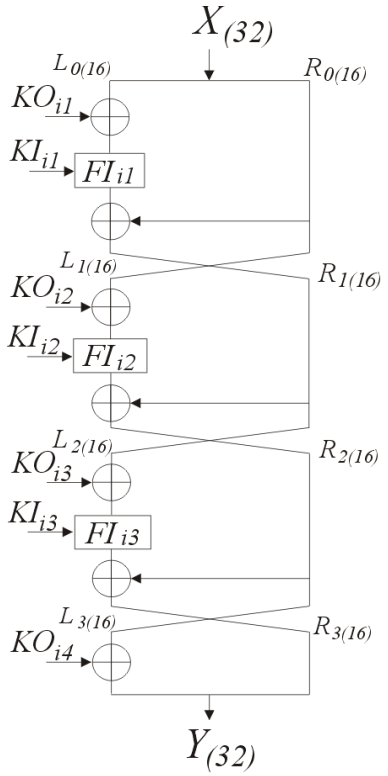


Figure 6.3. Function FO.

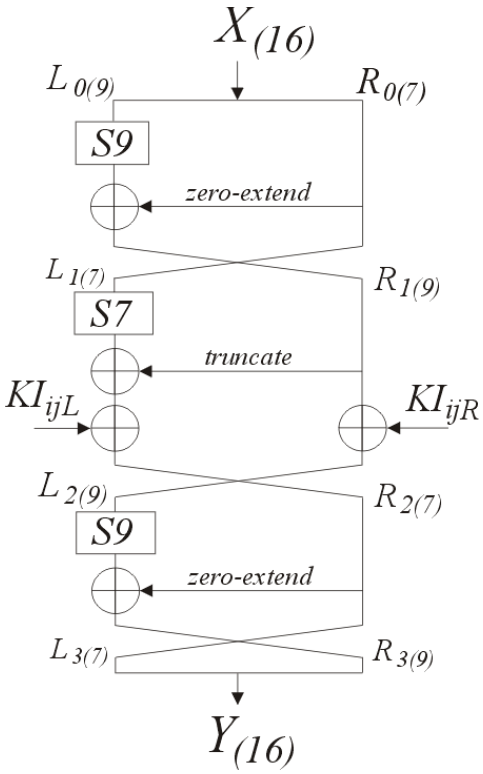


Figure 6.4. Function FI.

Function *FI* uses two S-boxes. *S7* maps 7-bit input into a 7-bit output and *S9* maps 9-bit input into a 9-bit output. Substitution tables (see Annex A) may be used and also equations to implement *S7* and *S9*. The equations describing these mappings are presented below, first for *S7* and then for *S9* (if there is no sign between two variables (e.g. z9z8) it means bitwise 'and' operation, while 1 means logic '1'):

```
y7=x7⊕x6x4⊕x7x4x3⊕x6x2⊕x7x5x2⊕x3x2⊕x7x6x1⊕x5x1⊕x7x2x1⊕x4x2x1⊕1

y6=x7x5⊕x7x3⊕x4x3⊕x6x2⊕x5x3x2⊕x1⊕x7x1⊕x4x1⊕x5x4x1⊕x6x3x1⊕x7x2x1⊕1

y5=x6x5⊕x7x5x4⊕x3⊕x6x3⊕x7x6x3⊕x7x2⊕x7x3x2⊕x4x3x2⊕x6x1⊕x4x1⊕x7x4x1⊕x3x1⊕
x5x3x1

y4=x7⊕x6⊕x7x6x5⊕x7x4⊕x5x3⊕x6x3x2⊕x5x1⊕x6x4x1⊕x7x3x1⊕x2x1⊕1

y3=x5x4⊕x7x3⊕x6x4x3⊕x2⊕x5x2⊕x6x5x2⊕x7x4x2⊕x6x1⊕x6x2x1⊕x3x2x1⊕1

y2=x7⊕x6⊕x5⊕x7x6x5⊕x7x4⊕x6x5x4⊕x6x3⊕x7x5x3⊕x7x2⊕x7x6x2⊕x4x2⊕x7x1⊕x5x2x1

y1=x7x6⊕x4⊕x7x4⊕x5x4x3⊕x7x2⊕x5x2⊕x4x2⊕x6x4x2⊕x6x1⊕x6x5x1⊕x7x4x1⊕x3x1⊕
x5x2x1
```

```
y9=z9z5⊕z9z4⊕z8z4⊕z8z3⊕z7z3⊕z7z2⊕z6z2⊕z6z1⊕z5z1⊕1

y8=z9z7⊕z6⊕z8z6⊕z7z6⊕z6z5⊕z5z4⊕z9z3⊕z7z3⊕z2⊕z9z1⊕z6z1⊕z4z1⊕1
```

```
y7=z9z8⊕z8z6⊕z5⊕z9z5⊕z7z5⊕z6z5⊕z5z4⊕z9z3⊕z4z3⊕z8z2⊕z6z2⊕z1

y6=z9⊕z8z7⊕z7z5⊕z4⊕z8z4⊕z6z4⊕z5z4⊕z4z3⊕z8z2⊕z3z2⊕z7z1⊕z5z1

y5=z8⊕z9z6⊕z7z6⊕z9z4⊕z6z4⊕z3⊕z7z3⊕z5z3⊕z4z3⊕z3z2⊕z7z1⊕z2z1

y4=z7⊕z9z6⊕z8z5⊕z6z5⊕z8z3⊕z5z3⊕z2⊕z6z2⊕z4z2⊕z3z2⊕z9z1⊕z2z1

y3=z9z8⊕z6⊕z8z5⊕z7z4⊕z5z4⊕z7z2⊕z4z2⊕z1⊕z9z1⊕z5z1⊕z3z1⊕z2z1⊕1

y2=z8⊕z9z8⊕z8z7⊕z7z6⊕z9z5⊕z4vz8z3⊕z6z3⊕z9z2⊕z5z2⊕z3z2⊕z8z1⊕1

y1=z9⊕z9z8⊕z8z7⊕z5⊕z9z4⊕z7z4⊕z6z3⊕z4z3⊕z9z2⊕z9z1⊕z6z1⊕z3z1⊕1
```

In figure 6.4 function *FI* uses two additional functions: *truncate* and *zero-extend*. The function *truncate* takes 9-bit value and converts it into a 7-bit value by dropping the two most-significant bits. The function *zero-extend* takes a 7-bit value and converts it into a 9-bit value by adding two bits to the most-significant end.

The output of each round (stage) is produced by the following equations.

For the odd rounds ($i = 1, 3, \ldots, 7$) :

$$R_i = FL_i(L_{i-1}, KL_i),$$

$$L_i = FL_{i+1}(R_{i-1}, KL_{i+1}) \oplus FO_i(R_i, KO_i, KI_i),$$

For the even rounds ($i = 2, 4, \ldots, 8$) :

$$R_i = L_{i-1},$$

$$L_i = R_{i-1} \oplus FO_i(R_i, KO_i, KI_i),$$

After the last round ($i = 9$):

$$R_9 = FL_9(L_8, KL_9),$$

$$L_9 = FL_{10}(R_8, KL_{10})$$

The final 64-bit ciphertext is produced from the concatenation of $L_9$ and $R_9$.

The decryption mode operation of MISTY1 is similar to the encryption mode. The only differences are the reverse order of the sub-keys and the replacement of the function FL by the function $FL^{-1}$ (see figure 6.5 for $FL^{-1}$ function).

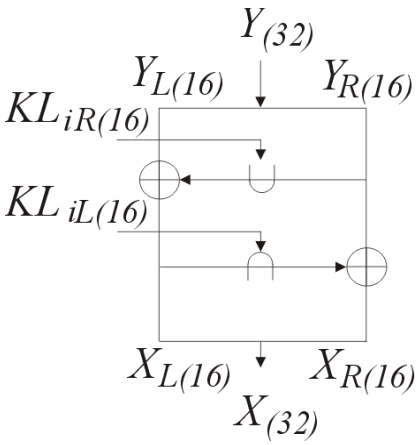


Figure 6.5. Function $FL^{-1}$.

The key scheduling part of MISTY1 takes the 128-bit secret key $K_{(128)}$ and divides it in 16-bit subkeys values $K_1, K_2, \ldots, K_8$. Based on these 16-bit values and using *FI* function other 8 subkeys $K'_1, K'_2, \ldots, K'_8$ are generated (*i* takes the value *i-8* when *i>8*):

$$K'_i = FI(K_i, K_{i+1})$$

In table 6.1. is presented the correspondence between the round subkeys $KO_{ij}$, $KI_{ij}$, $KL_{ij}$ and $K_i$ , $K'_i$ (*i* takes the value *i-8* when *i>8*).

Table 6.1. Mapping table for the subkeys.

| Round | $O_{i1}$ | $O_{i2}$ | $O_{i3}$ | $O_{i4}$ | $I_{i1}$ | $I_{i2}$ | $I_{i3}$ | $KL_{iL}$ | $KL_{iR}$ |
|---|---|---|---|---|---|---|---|---|---|
| Value | $K_i$ | $K_{i+2}$ | $K_{i+7}$ | $K_{i+4}$ | $K'_{i+5}$ | $K'_{i+1}$ | $K'_{i+3}$ | $K_{(i+1)/2}$ (odd i)<br>$K'_{i/2+2}$ (even i) | $K'_{(i+1)/2+6}$ (odd i)<br>$K_{i/2+4}$ (even i) |

### 6.1.3. Considerations regarding the security analysis of MISTY1

As their authors mention, MISTY1 algorithm was design to withstand various cryptanalytic attacks known at the design moment. MISTY1 was designed on the basis of the theory of "provable security" against differential and linear cryptanalysis [56].

Since the publication of MISTY1, due to evaluation processes such as NESSIE and CRYPTREC, many research efforts have been done for evaluating its security level; no security flaws of MISTY1 have been found for the recommended 8-round version. Some security aspects are covered below regarding MISTY1 and its strength against new attacks.

New type of attacks published after the algorithm appeared, like higher order differential cryptanalysis (which targets the S-boxes [57]), impossible differential cryptanalysis (addressing characteristic paths that never appear in Feistel cipher [58]) or slide attack (which targets the reuse of subkeys [59]) are successful up to five rounds but only if the FL-functions between the rounds are not used [10]. However due to the existence of the FL-functions these attacks are not working against MISTY1.

Even if *implementation attacks*, such as timing attacks and power analysis, are often powerful and realistic threats in smart card applications they are not covered by evaluations, as couples of countermeasures are available. Standard methods to avoid implementation weakness of block ciphers, for instance removing "jumps" or "variable-cycle instructions", can be applied to MISTY1 since the algorithm is composed of logical operations and lookup tables only. As such, proposing and developing error detection mechanisms to withstand fault analysis attacks lies in the same area of countermeasures against attacks targeting implementation.

## 6.2. Hardware simulation environment

In this section we introduce the hardware used for simulation and the synthesis/implementation tools are also mentioned. For comparison purposes we chose the same FPGA technology used in [55]. We used ModelSim XE III 6.0d for simulation of the VHDL code and Xilinx's ISE 8.1i environment targeting Virtex FPGA (device xcv1000, package bg560).

*Hardware description.* Virtex devices are composed of configurable elements *Configurable Logic Blocks (CLBs)* and *Input/Output Blocks (IOBs)* which are all interconnected by versatile routing resources. The routing resources permit the Virtex family to accommodate large and complex designs [60].

The CLBs provide the functional elements for constructing logic, while, the IOBs provide the interface between the package pins and the CLBs. CLBs interconnect through a *General Routing Matrix (GRM)*. The GRM contains an array of routing switches located at the intersections of horizontal and vertical routing channels.

Values stored in static memory cells control the configurable logic elements and interconnect resources. These values load into the memory cells on power-up, and can reload if necessary to change the function of the device.

The basic building block of the Virtex CLB is the *Logic Cell (LC)*. An LC includes a *4-input function generator*, *carry logic*, and a *storage element*. The output from the function generator in each LC drives both the CLB output and the D input of the flip-flop. Each Virtex CLB contains four LCs, organized in two similar *slices*, as shown in Figure 6.6. Figure 6.7 presents a more detailed view of a single slice.

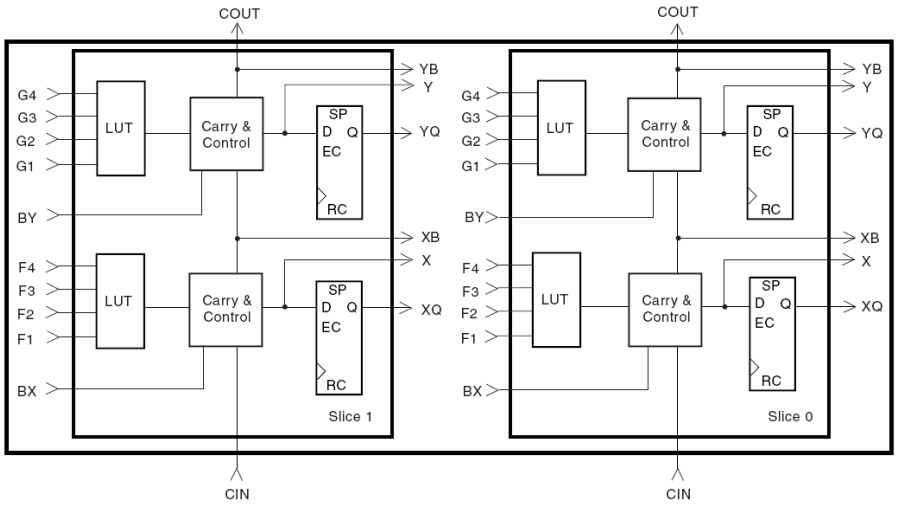


Figure 6.6. Virtex. 2-slice CLB [60]

Besides the four LCs, the CLB contains logic that combines function generators to provide functions of *five* or *six inputs*. Due to this logic, when estimating the number of system gates provided by a given device, each CLB counts as *4.5 LCs*.

*Look-Up Tables (LUTs)*. Virtex function generators are implemented as 4-input LUTs. In addition to operating as a function generator:

- each LUT can provide a 16 x 1-bit synchronous RAM;
- two LUTs within a slice can be combined to create
- a 16 x 2-bit or
- 32 x 1-bit synchronous RAM,
- or a 16x1-bit dual-port synchronous RAM [60].

A 16-bit shift register can be provided by the Virtex LUT, useful for capturing

high-speed or burst-mode data.

*Storage Elements.* The storage elements in the Virtex slice can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches. The D inputs can be driven either by the function generators within the slice or directly from slice inputs, bypassing the function generators.

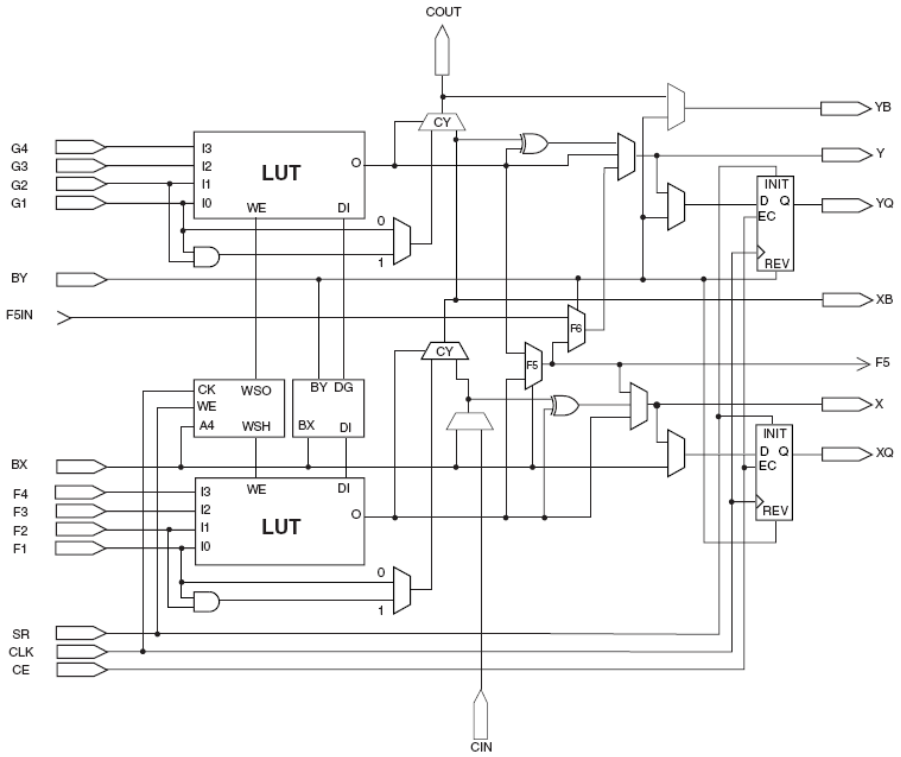


Figure 6.7. Virtex slice. One slice accommodates two 4-input LUTs [60]

In addition to Clock and Clock Enable signals, each Slice has synchronous set and reset signals (SR and BY). SR forces a storage element into the initialization state specified for it in the configuration. BY forces it into the opposite state. Alternatively, these signals can be configured to operate asynchronously. All of the control signals are independently invertible, and are shared by the two flip-flops within the slice.

*Additional Logic.* The F5 multiplexer in each slice combines the function generator outputs. This combination provides one of the following:

- a function generator that can implement any 5-input function,
- a 4:1 multiplexer,
- selected functions of up to nine inputs.

In the same way, the F6 multiplexer combines the outputs of all four function generators in the CLB by selecting one of the F5-multiplexer outputs. This permits the implementation of one of the following:

- any 6-input function,
- an 8:1 multiplexer,

- selected functions of up to 19 inputs.

Each CLB has four direct feedthrough paths, one per LC. These paths provide extra data input lines or additional local routing that does not consume logic resources.

*Arithmetic Logic*. Dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions. The Virtex CLB supports two separate carry chains, one per slice. The height of the carry chains is two bits per CLB. The arithmetic logic includes an XOR gate that allows a 1-bit full adder to be implemented within an LC. In addition, a dedicated AND gate improves the efficiency of multiplier implementation. The dedicated carry path can also be used to cascade function generators for implementing wide logic functions [60].

As we already mentioned, we chose this technology in order to allow relevant comparisons with the best-known FPGA cipher implementations. In this chapter, we compare the number of LUT's and slices used. We also evaluate the delays using maximum path delay in order to see the time overhead of the error detection mechanisms.

## 6.3. Error propagation in MISTY1 algorithm

The propagation of the errors in case of MISTY1 algorithms depends on the propagation of errors in each round and subround, respectively, of each function which describes the algorithm's round. We assume in this analysis that only a *single bit* may become faulty at any given time instant. In this section the propagation of *single bit fault* for round and subround functions of the algorithm is analyzed. The purpose of the analysis is to identify the way the single bit fault spreads to the output of each component, mainly to distinguish between two main types of propagation: first, when only one bit – or a known number of bits – is/are affected and second type when the fault spreads to all bits of the output.

In this section we analyze the propagation of errors in the two main components (see figure 6.1) of the MISTY1 algorithm: FL function – the subround function and FO function – the round function. We assume that a single bit fault is affecting the data flow input of these blocks. The purpose of this analysis is to identify the effect of a fault on the output of both components (i.e. if there is any dependency between the location of the fault and of the error(s), and if there is a 'avalanche' effect, as it should be in each round of an encryption algorithm).

*FL function and error propagation*. Based on the structure of the FL function, as can be seen in Figure 6.2 we can draw some observations, which have been confirmed by simulation tests. Depending on the location of the injected fault, the output of the function FL is affected by two errors. If a fault is injected in the left half side of the input, $X_{L(16)}$, then two errors are detected on the same position at the output, in the left half, of the function FL and on the right half. As well, if a fault is injected in the right half side of the input, $X_{R(16)}$ (on the position *i=15 downto 0*, where the maximum value of *i* is 31) then two errors are detected: one on the same position (*i*) of the right half of the output of the function FL, and the second in the same position but in the left half of the output (*i+16)*. During the simulation tests no fault injected passed undetected.

*FO function and error propagation*. In contrast with FL function, FO function does not show such dependency on location, neither on the numbers of errors generated. However, FO produces a 'avalanche' effect (the purpose of cryptographic algorithms is to 'hide' the input, as such even if there is minor difference in inputs the outputs should be different and no correlation easy to make).

Using the experimental environment described in section 6.2, we implemented a test module and simulated the behavior of the function FO in case of a single bit fault injection. We injected the fault in different locations and we compared the results between execution without fault injected and execution with fault injected. We run simulations, using couple of millions of test vectors and the results of these simulations are presented in figure 6.8.

The results shown in figure 6.8 are represented after running about eight millions of pairs of test vectors (faulty and non faulty). On the horizontal axis the number of erroneous bits (out of 32 output bits) is indicated. On the vertical axis (up) the percentage of total tests generating the same number of fault is given (ex. 13.75%, meaning almost 1100 000 test vectors of the tests detected 14 bits affected by errors). 94% of the test vectors had between 10 and 20 bits erroneous, while 98% between 9 and 22 error bits.

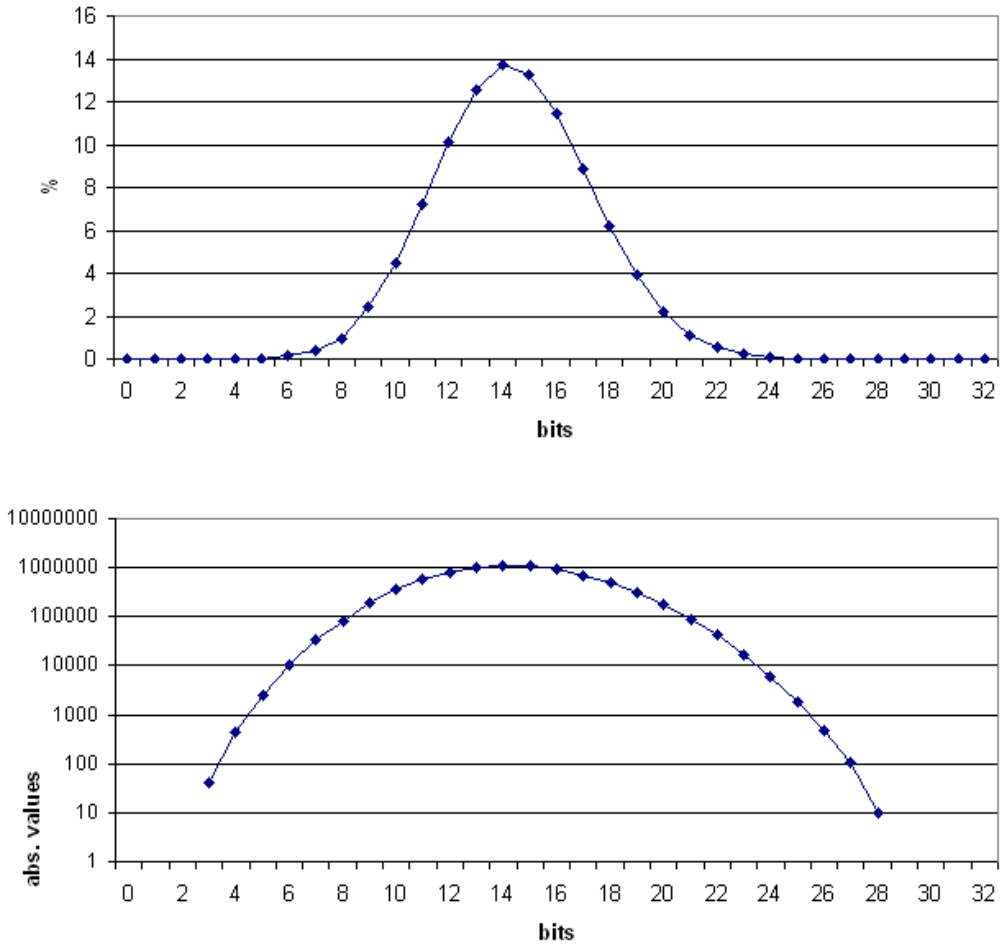


Figure 6.8. FO error propagation. Percentage distribution (up) and logarithmic distribution (down)

During the simulations no fault passed undetected, and no test vector generated less than 3 errors, or more than 29 errors. This can be noticed in figure 6.8 (down) where logarithmic representation is used for the values of vertical axis. During these simulations no correlation regarding the location of the bits affected by error has been noticed.

Analyzing the results of these simulations for error propagation some conclusions can be drawn. On one hand, using a parity bit error detection method applied for the entire output of FL (32 bits) function is not a reliable method – no injected errors can be detected. The use of a parity bit is suited only if it is applicable for each half (16 bits or each byte) and assuming a single faulty bit. On the other hand the FO function, given the 'avalanche' effect, no assumption can be made regarding the even or odd number of bits that are faulty at the output if a fault is injected. As such there is no guarantee that a parity bit method would detect all faulty outputs.

Given the manner in which errors spread in FO function we consider important to investigate error detection methods applicable both at function level and algorithm level in case of MISTY1 algorithm.

## 6.4. MISTY1 and the available error detection techniques

As already mentioned in previous chapter in section 5.1.2, some specific methods are presented and applied mostly to AES selected algorithm:

- for AES finalists *redundancy-based techniques*, where the encrypted message is decrypted and compared with the input message, was proposed by Karri et al. in [45];
- in [46], Bertoni et al. describe and evaluate a scheme based on *error detection codes (i.e. parity codes)* for the specific case of AES (Rijndael);
- while Karpovsky &co., in [47], used symmetric *nonlinear (cubic) error detection codes* for a fault analysis attack resistant AES implementation.

The first method, a *general* method, has as main disadvantages the hardware and time overhead besides the security concerns regarding the storage of intermediary results (e.g. for algorithm level concurrent error detection, for instance the initial plaintext has to be *stored* till the encryption and then decryption take place to be able to compare the result with the initial plaintext and to produce or not an error signaling).

The last two methods are using *specific* functions and estimation modules (e.g. to estimate the parity) designed for certain algorithm. Such specific error detection mechanisms *depend* on the design principles and operations used by the encryption algorithm. For instance, the method proposed by Bertoni et al. in [46] takes advantage of the design of Rijndael algorithm and generates a parity matrix.

Even if the specific methods are considered to be, compared with the general ones, more advantageous from cost perspective - this is not quite true. Nevertheless if from theoretical perspective the general methods, relying on concurrent error detection, require 100% overhead in time or hardware, actually after implementation, the overhead is not so high, and can be even below the level of overhead generated by specific techniques. We are going to argument these findings in this chapter and to give values from our cost analysis evaluation of MISTY1 and from references. Besides this, all the comparisons regarding hardware

overhead were made using mostly FPGA implementations, where, as we are going to show, not always all resources are completely used and so redundant logic may use part of it without increasing the hardware overhead cost as estimated in theoretical studies.

## 6.5. Complemented duplication error detection

In this section we are proposing a new error detection technique and afterwards we are applying it to MISTY1 algorithm (described in section 6.1).

We are going to call this method *complemented duplication error detection (CD)*. The CD error detection method relying on the duplication with complementary logic technique described in [44]. In this method, relying on previously presented complementary logic, one module is designed using positive logic while the other module uses negative logic (i.e. positive logic implies that the higher of the two voltages used in a logic circuit represents a logic '1' while the lower represents '0'; negative logic uses the higher of the two voltages to represent a logic '0' while the lower represents '1').

Our method does not assume using of positive logic and negative logic, but only the idea of using a second component with complemented inputs being "processed" to generate the complemented outputs of the first component.

This technique is similar with so called operation level concurrent error detection method proposed in [45], as for each operation there is a 'complemented' operation executed concurrently. However there are major differences – the inputs are different (complemented) and there is no requirement for storage of intermediate values.

### 6.5.1. Error detection using complemented duplication

Complemented duplication error detection relies on hardware redundancy and assumes the use of each block of data (or key or intermediate values) *first* as described in algorithms and *second* with the complemented values and using complemented operations. New operations are applied to the complemented values, such that, the results of the two operational flows (direct and complemented) to be always complements (XOR-ing them to result all '1''s).

This method can be considered algorithm dependent (i.e. specific technique) because for each operator *op1* applied between *A* and *B* blocks of data, so that *A op1 B = C,* another operator *op2* should be applied to the inverse value blocks *notA* and *notB* so that *notA op2 not B = notC*, where *notX* is the complement value of *X*. However, due to its simple design and due to the lack of prediction functions CD can be considered a general error detection method.

The general representation of CD error detection technique is presented in figure 6.9.

In figure 6.9, *A*, *B*, *C* and theirs inverses *notA*, *notB, notC* are blocks of bits, and *op1* and *op2* are operands fulfilling the following conditions:

A op1 B = C,

notA op2 not B = notC

so that *notC* is the bitwise complement value of *C* and $C \oplus notC$=*"1…1"*.

For instance, if *op1* is the operator OR, for complements will be applied operator AND. Hence, if *op1* is the operator AND applied for the direct values, then, for the complements, *op2* is applied (which is OR in this case).

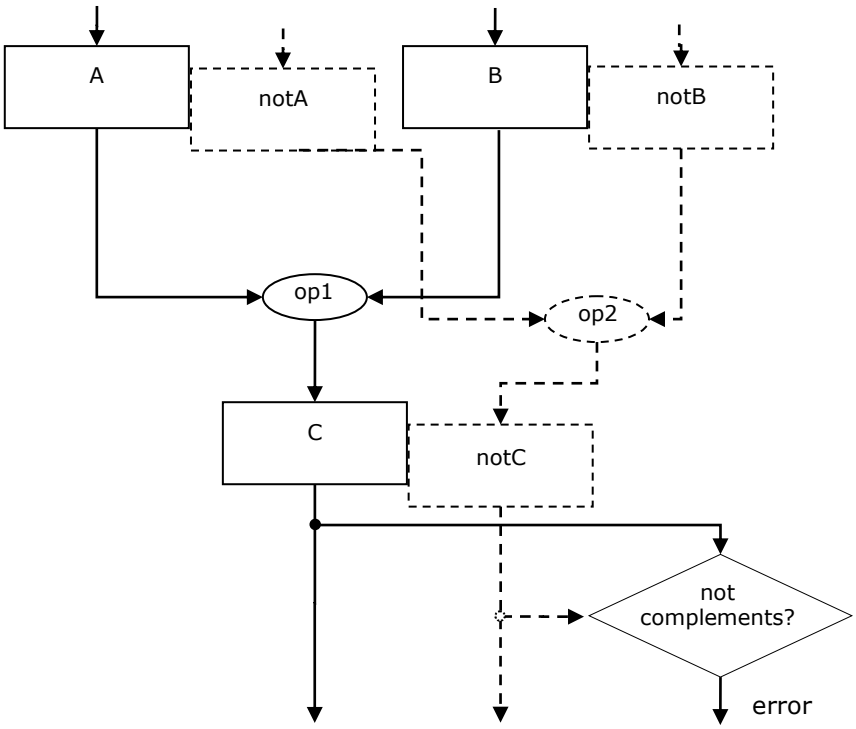


Figure 6.9. Using CD error detection. General representation.

### 6.5.2. Description of CD error detection using Boolean algebra

The CD error detection method, as well as the duplication with complementary logic technique can be also described using the Boolean algebra and dual functions [44].

*Definition.* Let *V* be a vector of n bits given by $V=(v_{n-1}, \ldots, v_1, v_0)$. For every combinational Boolean function *f(V)* there exists *a dual function* $f_d(V)$ defined by $f_d(V')= f'(V)$, where *V'* is the complement of *V* and *f'* is function *f* complemented.

*Example*. Dual functions can be obtained by replacing AND operations with OR operations, OR operations with AND operations, bits of value '1' with '0', and bits of value '0' with '1'.

The characteristics and advantages of the original duplication with complementary logic technique [44] do not apply directly to CD error detection and FPGA implementation (e.g. the different physical implementations which reduce the probabilities of same design or masking problems are reduced). However, as different logic and different functions are implemented on the same board of FPGA there are low chances that the same fault to generate same error which passes undetected and so the probability of detecting errors is higher.

One of the disadvantages of the duplication with complementary logic was the change of topology given by the hardware limitations of replacing e.g. large

number of input NOR gates with the same large number of input NAND gates [44].

However, this is not a disadvantage for FPGA boards, and for CD error detection method, as all the functions including the gates functionalities are implemented using look-up-tables (LUT's) as can be seen in figure 6.7.

Another disadvantage of the duplication with complementary logic mentioned in [44] is due to the difficulty to convert a complex function in the complementary logic. For CD error detection in case of FPGA implementation this depends on the complexity of the implemented function as well.

### 6.5.3. Applying CD error detection for MISTY1 functions

In this section we are going to analyze the suitability of CD error detection in case of MISTY1 algorithm, to apply the error detection for MISTY1 functions and evaluate the costs of error detection mechanism.

As we introduced in section 6.1, data randomizing part of MISTY1 uses for all round a function FO, and for all odd rounds, and after the last one, a subround with FL function.

FO function includes FI functions. FL uses bitwise AND, OR and XOR operations. FO function relies on bitwise XOR operation and sub-functions FI. FI uses XOR bitwise operation and two substitution boxes: S7 and S9. For implementation purposes S7 and S9 can be defined as tables or implemented using logic expressions using bitwise XOR and bitwise AND.

The operations and sub-functions used in MISTY1 are summarized in the following table (Table 6.2).

Table 6.2. Functions and operations used in MISTY1

| Function | Operations and/or functions |
|---|---|
| **FL** | AND, OR, XOR |
| **S7** | AND, XOR |
| **S9** | AND, XOR |
| **FI** | XOR, S7, S9, zero-extend, truncate |
| **FO** | XOR, FI |

As presented in section 6.4.1, for all these operations another operation on the complemented inputs is used in case of CD error detection. In the case of FI dual function, *zero-extend* is replaced by *one-extend* (extending 2 bits of one, from a vector of seven bits to a vector of nine bits) and *truncate* is used for both FI function and FI dual function. As such, for each function in MISTY1 a complement function can be defined, as all the functions can be described using AND and OR functions (see table 6.3). S-boxes, as seen in section 6.1.2 are composed of AND and XOR operations.

Let $A$ and $B$ be two vectors of $n$ bits given by $A=(a_{n-1}, \ldots, a_1, a_0)$, and $B=(b_{n-1}, \ldots, b_1, b_0)$. For every function/operation $f(A)$, $f(A,B)$ of MISTY1 we are defining the *dual function* $f_d(A)$, $f_d(A, B)$. Dual functions applied to complemented inputs $f_d(A')$ and $f_d(A', B')$ are presented in table 6.3, where $A'$ is the complement of $A$. These are the complement operations for the operations described by $f(A)$, $f(A,B)$.

Using Boolean algebra the expressions $f_d(A', B')= f'(A, B)$ and $f_d(A')= f'(A)$ can be verified. Hence, the output of the dual function $f_d$ (function applied to the complemented inputs) is the complement of the output of function $f$.

Table 6.3. MISTY1 operations and their complements

| **Operation (*f(A), f(A,B)*)** | **Complement operation ($f_d(A')$ and $f_d(A', B')$)** |
|---|---|
| A AND B | A' OR B' |
| A OR B | A' AND B' |
| A XOR B | (A' XOR B')' or A' XOR B or A XOR B' |
| A truncate | A' truncate |
| A zero-extend | A' one-extend |

In order to analyze the cost of error detection a competitive implementation of MISTY1 is required. As no free source code implementing MISTY1 using VHDL was found we developed our implementation for each function. We compare in table 6.4 our implementation results with the ones published based on NESSIE's MISTY1 implementation in VHDL [55] and targeting Virtex FPGA device xcv1000, package bg560. No other valuable implementations for this environment (and showing detailed implementation results) were found.

Table 6.4. Implementation results

| **Component** | **Implementation [55] # of 4 LUT's** | **Implementation (this work) # of 4 LUT's** | **Implementations comparison** |
|---|---|---|---|
| **FL** | 32 | 32 | 0.00 |
| **S7** | 44 | 48 | 9.09 |
| **S9** | 44 | 53 | 20.45 |
| **FI** | 172 | 170 | -1.16 |
| **FO** | 655 | 602 | -8.09 |

If the results of our implementation are compared with the reference results, we can notice that for S boxes the results are disadvantageous. For S7 and S9 our implementation is less advantageous. However, functions FI and FO incorporate S7 and S9 and are compensating this disadvantage, as they have better simulation results. FL function has the same number of 4 input look-up-tables used (32 of 4 LUT's).

Based on this comparison, we can conclude that our implementation is comparable with the reference one and can serve as reference for analyzing the cost of error detection.

The results of our implementation (without and with complemented error detection mechanisms included) are listed in table 6.5. We introduced in this table the number of slices as well. This is relevant in order to notice that not all resources reported as used are actually mapped i.e. not all slices have both 4 LUT's tables used. In case of FL function, 18 slices contain a number of 36 4 LUT's, however only 32 are used – in this case 4 LUT's with 4 inputs each are unused.

For the CD error detection mechanism a *self-checking checker* was deployed [61]. For our implementation the output of the checker uses a distance-two error detecting code *1-out-of-2*, where {(0,1), (1,0)} are valid code words, first for correct output, second for 'error' output (see figure 6.9) and {(0,0), (1,1)} are invalid code words. As such unidirectional errors are detected.

Table 6.5. Analysis of overhead for CD error detections

| Component | # of 4 LUT's | # of slices | # of 4 LUT's (CD error detect.) | # of slices (CD error detect.) | area overhead (in # of 4 LUT's) | area overhead (in # of slices) |
|---|---|---|---|---|---|---|
| **FL** | 32 | 18 | 32 | 18 | 0 % | 0 % |
| **S7** | 48 | 27 | 48 | 27 | 0 % | 0 % |
| **S9** | 53 | 30 | 55 | 31 | 3.77 % | 3.33% |
| **FI** | 170 | 97 | 187 | 106 | 10.00 % | 9.28% |
| **FO** | 602 | 345 | 651 | 371 | 8.14 % | 7.54% |

Regarding the cost of implementation, from table 6.5 it can be noticed that the largest hardware overhead was generated in case of FI function i.e. 10%, for the number of LUT's. In the same time, as every slice contains two 4-input LUT's next to other logic, it can be noticed that even in case of complemented duplication not all 4 LUT's from the FPGA slices are used. Time related values were not included due to small differences: the maximum combinational path delay had an increase with 3.75% (from 45.059ns for the unprotected implementation to 46.751ns with complemented duplication), in case of FO function in our simulations.

### 6.5.4. Analysis of overhead costs

Due to such low overhead we felt the need to validate these results. We analyze in this section the FL function and the logic used to implement the component and we notice that the same logic can be used, in the same time, to implement the dual function. As such no extra hardware is required.

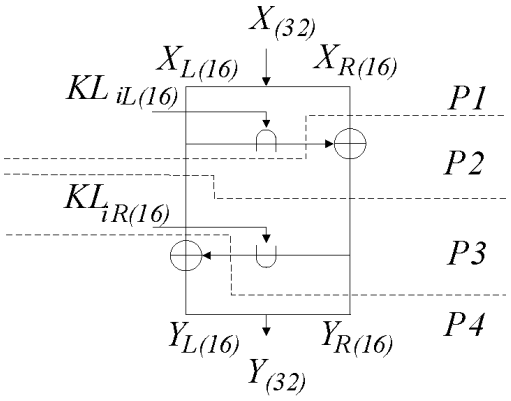


Figure 6.10. FL function broken down in steps.

As seen in figure 6.2 the following steps are carried out during FL function execution:

P1: on 2 vectors of 16 bits AND2 logic

P2: on 2 vectors of 16 bits XOR2 logic

P3: on 2 vectors of 16 bits OR2 logic

P4: on 2 vectors of 16 bits XOR2 logic

Given the properties of dual functions, P3 is the dual function of P1. If after P4 invertors are placed we obtain the dual function of P2. Hence the logic used for the steps P3 and P4 may be used simultaneous with the steps P1 and P2 to produce the output of the first half of FL dual function. Then during P3 and P4 are executed, P1 and P2 corresponding logic may be used to execute the second half of the dual function of FL. Logic for inversion is required as well as multiplexer to set the flow. These logic control functionalities are easily implemented using the rest of the logic from the FPGA board where a CLB may be execute any of: any 6-input function, an 8:1 multiplexer, any selected functions of up to 19 inputs.

No extra time is required to execute these steps – as they are executed in parallel; as such no time overhead is generated.

The use of a self-checking checker did not add cost (extra hardware CLB, LUT's or supplementary delay in maximum combinational path delay) compared with the simulations using a simple checker.

Based on this analysis, the 0% hardware overhead does not surprise anymore. Further analysis could be carried out for the other functions, where the stages of these functions cannot be separated such as for the case of FL function. However, it is clear that the software used to map the VHDL description, on the logic of the FPGA board, performs optimization. Furthermore, the results of implementation/simulation on FPGA (where optimization can (re-)allocate the supplementary hardware to the unused resources of already counted slices/LUT's) cannot be compared with theoretical ones which are not considering optimization (e.g. the results of overhead presented at the end of Chapter 5).

## 6.6. Using parity prediction for error detection

Error detection mechanism based on *error detection codes (i.e. parity codes)* applied for AES (Rijndael) algorithm is presented by Bertoni et al. in [46]. Only the parity prediction functions for Rijndael transformations are given in [46], which are algorithm dependent.

We propose in this section the general mathematical description and parity prediction for operations used in MISTY1 algorithm (table 6.2).

*Definition*. Let *A* and *B* be two vectors of *n* bits given by $A=(a_{n-1}, \ldots, a_1, a_0)$, and $B=(b_{n-1}, \ldots, b_1, b_0)$. For every function/operation *f(A), f(A,B)* of MISTY1 we define the *parity prediction functions p(f(A)), p(f(A, B))*, where *f* is function used in MISTY1 algorithm, *p(A)* is the parity of vector *A* and *p(B)* is the parity of vector *B*. For any *A* vector the following expression holds: $p(A)= a_{n-1}\ XOR\ \ldots XOR\ a_1\ XOR\ a_0$.

We are introducing in table 6.6 a parity prediction function for each operation of MISTY1 algorithm.

Table 6.6. Parity prediction for MISTY1 operations

| **Operation (parity of… is →)** | **Parity prediction** |
|---|---|
| A AND B | p(A) XOR p(B) XOR p(A OR B) |
| A OR B | p(A) XOR p(B) XOR p(A AND B) |
| A XOR B | p(A) XOR p(B) |
| A truncate | p(A) XOR p(truncated bits) |
| A zero-extend | p(A) |

Using Boolean algebra or simulation software the expressions from table 6.6

can be verified. In Table 6.6, for the parity prediction *p(A AND B)* and *p (A OR B)* respectively, it is not sufficient to compute the parity of the operands A and B, but also the parity of the result of the other operation (A OR B) respectively (A AND B) are required.

We analyze FL function. Starting from figure 6.11 we evaluate the resources required to generate parity bits, and to predict the output parity.

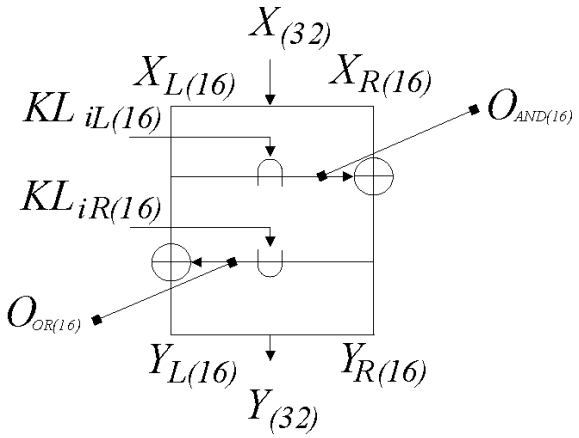


Figure 6.11. FL function and parity points.

Based on the observations made in Section 6.3 addressing error propagation, there has to be at least a parity bit per each half (16 bits) of the output in order to detect single-bit faults (as we saw, any faulty bit in position *i, 0≤i≤15,* in $X_{R(16)}$ or $X_{L(16)}$ is propagating in both halves of the output: $Y_{L(16)}$ and $Y_{R(16)}$ on the same *i* position generating 2 errors). As such, only one bit parity for 32 bits input/output will not be able to detect all single-bit errors. The following option can be considered – parity generated for 16 bits or for 8 bits.

A parity generator on 16 bits has between disadvantages the large number of XOR gates required to generate the parity (8+4+2+1) and the delay given by $4*t_{XOR}$ to detect the error (where $t_{XOR}$ is the time required to generate the output for a XOR gate).

A parity generator on 8 bits has less disadvantages: the number of XOR gates required to generate the parity (4+2+1) and the delay given by $3*t_{XOR}$ to detect the error are smaller. However, more parity generators are required and more parity bits are stored in this case.

Analysing figure 6.11, considering the parity prediction functions from table 6.6, the following points need to be evaluated (parity bits to be generated):

- for the 4 inputs - plaintext $X_{L(16)}$ and $X_{R(16)}$ and subkeys $KL_{iR(16)}$ and $KL_{iL(16)}$,
- for 4 intermediate results on lines $O_{AND(16)}$, and $O_{OR(16)}$ and, on the same place, the results of the OR function where AND function is in the algorithm, and the results of the AND function where OR is used (see parity prediction function for OR respectively AND operations in table 6.6),
- for 2 outputs $Y_{L(16)}$ and $Y_{R(16)}$.

Hence there are 10 places where parity checkers need to be included in order to generate the parity prediction. As such, this parity prediction method is expensive both in hardware (number of supplementary XOR gates) and in time overhead (due to parity checkers).

Nevertheless in section 6.3 we showed that the 'avalanche' effect does not allow only one parity bit for FO function. Almost 50% of the faults are generating an even number of errors at the output of FO function. However, further analysis could be carried out to see which is the hardware and time overhead of this parity prediction error detection method for the other functions of MISTY1 algorithm.

Based on the above analysis for FL and the error propagation in FO we consider that this parity prediction method is on the one hand expressive (both from hardware and time overhead above 100% for FL) and on the other hand does not provide good error detection.

Table 6.7. Analysis of overhead for parity prediction error detection in case of FL function

| Com-ponent | # of 4 LUT's | # of slices | Max. comb. path delay | # of 4 LUT's (parity predict-tion) | # of slices (parity predi-ction) | Max. comb. path delay (parity prediction) | Area overhead (in # of 4 LUT's) | Area overhead (in # of slices) | Time overhead (max. path delay) |
|---|---|---|---|---|---|---|---|---|---|
| FL | 32 | 18 | 8.369 ns | 109 | 57 | 15.260 ns | 240% | 216% | 82% |

Based on simulation (see table 6.7 where a bit parity was used for 16 bits of data), the overhead of the parity prediction method in case of FL function of MISTY1 algorithm is above 100% area overhead and in the same time, the time overhead is 82%. As such, the conclusion is that this method is expensive for MISTY1 algorithm.

## 6.7. Analysis of overhead in related work. Comparison

Based on the observations from the previous sections we analyze and compare the cost of different error detection methods. We make distinction between the theoretical assumptions of hardware and time overhead and the results after simulation/ implementation on boards using optimizing software.

Redundancy based techniques, namely concurrent error detection methods, used in [45], theoretically assume 100% overhead; however, in practice, in FPGA implementations the results are different. For the case of Rijndael, the overhead varies between about 19% in case of round level CED to 21% for algorithm level CED and 38% for operation level CED. Besides this, the performance degradation due to time overhead varies also in case of Rijndael from about 24% in case of operation level CED to 27% for round level CED and 61% in case of algorithm level CED [45]. (The implementation overhead is in about the same range for the other evaluated algorithms: RC6 and Twofish and in some cases a few percent lower for Serpent). However, the method assumes storage of plaintext/intermediate results/ciphertext and so is not considered a secure method.

Error detection codes based on parity codes, considered in [46], is approximated to generate between 10 to 20% overhead for Rijndael. Our similar method introduced in section 6.6, when simulating the implementation with parity prediction for MISTY1's FL function, produced a simulation report with large area

overhead (more than 200%) and time overhead 82%. In such situation, the approximations from [46] seem contradictory. However this can be due to the specificity of the algorithm. In case of MISTY1's FL function, parity prediction function has to be applied after each operation and this generates high overhead. However, this method detects only single-bit faults or odd number of bits faults.

Symmetric nonlinear (cubic) error detection codes used in [47] requires 75% hardware overhead for Rijndael protection. This method seems more secure and has a better error coverage compared to parity based techniques. We did not apply this specific error detection technique to MISTY1 algorithm.

Comparing the theoretical cost assumption of CED methods with FPGA implementations cost analysis does not seems to be appropriate. As can be seen from the paragraph on CED techniques, even if CED theoretically assumes 100% overhead, in practice, even for an algorithm such as Rijndael which is not so efficient for hardware implementation as MISTY1, the implementation results are much better. As such, 100% overhead is not the reference value for error detection techniques.

However, comparing with the above methods, our new method called *complemented duplication error detection* introduced in section 6.5 is on the one hand less expensive (compared with all of them, however probably depending on the algorithm) and on the other hand its security level is at least as the level of other specific methods, but better than CED's security.

## 6.8. Conclusions

In this chapter we propose a new error detection technique *complemented duplication error detection* and compare its cost with previous proposed techniques for error detection. We use an algorithm designed to have a good hardware performance to evaluate our technique: MISTY1 algorithm. MISTY1 was evaluated and selected by both NESSIE (in Europe) and CRYPTREC (in Japan) evaluation projects as secure and efficient 64-bit block algorithm.

We discussed in this chapter error detection mechanisms using both general and specific techniques already presented in Chapter 5.

The main contributions of this chapter are:

- error propagation analysis of single-bit fault for MISTY1 main functions;
- analysis of the results of error propagation;
- a new error detection technique based on complemented duplication
- the general description of the technique;
- state of corresponding 'complement' operations for MISTY1 operations;
- cost analysis of the new method compared with the implementation of MISTY1 without error detection mechanisms;
- parity prediction technique applied to MISTY1 algorithm
    - stating the functions for generating parity bits for each operation of MISTY1 algorithm;
    - analysis of the coverage of the parity prediction technique in case of MISTY1;
    - evaluation of parity prediction technique for one of the MISTY1 functions;
- overhead analysis for previous research results and our technique;
- comparison of error detection techniques (from related work and ours)

from several of perspectives (e.g. theoretical overhead costs, implementation overhead costs, security, error detection coverage).

A fault tolerant implementation of MISTY1 algorithm, designed using VHDL language and simulated using ModelSim XE III 6.0d and Xilinx's ISE 8.1i environment targeting Virtex (VIRTEX1000bg560-6) device, is used for evaluation. The results are compared, for evaluation purposes, with other implementations – which are not considering error detection mechanisms.

The method proposed in this Chapter, using complemented duplication has a low overhead cost and is more secure as it does not require storage of plaintext, intermediate results or ciphertext. However, these findings may be further evaluated in the case of other cryptographic algorithms.

# 7. CONCLUSIONS

## 7.1. Findings of this dissertation

This dissertation addresses the need for dependability and security of computing systems. From the broad research field on dependability the focus is on how security can improve dependability. As we know from reliability, the reliability of a system relies on each component's reliability, and this is true also for dependability. For a system which includes security mechanisms, if there are dependability requirements, then all components need to be dependable, including the security mechanisms. In this work we addressed the requirements for dependability of security mechanisms, namely the fault tolerance of security implementations.

This relatively new research topic on fault tolerant security was determined by the development of implementations related attacks such as fault analysis attacks. This work addresses this research topic. It motivates the need for research in this field, identifies solutions and proposes mechanisms to improve the dependability of security mechanisms, contributing this way to the dependability of systems.

This chapter contains a section listing the personal contributions and relevant personal papers published on these topics. The last section proposes future research paths inspired from this dissertation.

## 7.2. Personal contributions

Here are the personal contributions of this dissertation

- analysis of the trend towards adaptable systems, where dependability needs to adapt its requirements and means to the environmental changes; the need of security in this context;

  Publications: [13], [62], [63], [64], [65].

- analysis of relationship between security and dependability; identifying dependencies between security and fault tolerance;

  Publications: [16], [17], [66].

- state-of-the-art investigation of
  - security and cryptographic algorithms evaluated in selection processes in USA (NIST selection process for AES), Europe (NESSIE research project) and Japan (CRYPTREC selection process for algorithms to be use din eGovernment infrastructure);
  - intrusion tolerance approaches, addressing research results from MAFTIA and ITUA projects;
  - mechanisms to detect/tolerate errors in cryptographic implementations;

  Publications: [16], [49].

- focusing on faults and their impact on operation modes
  - identifying vulnerabilities for operation modes; proposing solutions to avoid vulnerabilities in operations modes;

    Publication: [43].

- new error detection techniques applied for block ciphers

  - *Last round error detection*. Proposing and applying new error detection technique, using last round error detection CED, for cryptographic implementations targeting permanent faults in Triple-DES algorithm; cost analysis of this error detection technique;

  - *Error detection relying on complementary property*. Proposing an error detection technique relying on the complementary property of DES algorithm;
    - Cost analysis of this technique for Triple-DES algorithm. Applicability analysis for other algorithms;
    - error propagation analysis of single-bit fault for MISTY1 main functions; analysis of the results of error propagation;

  - *Complemented duplication error detection*. Introducing and applying a new error detection technique based on complemented duplication for MISTY1 algorithm
    - the general description of the technique;
    - state of corresponding 'complement' operations for MISTY1 operations;
    - cost analysis of the new method compared with the implementation of MISTY1 without error detection mechanisms;

      Publications: [49], [50], [67].

- evaluation of error detection techniques vs. related work techniques
  - parity prediction technique applied to MISTY1 algorithm
  - stating the functions for generating parity bits for each operation of MISTY1 algorithm;
  - analysis of the coverage of the parity prediction technique in case of MISTY1; implementation evaluation of parity prediction technique for one of the MISTY1 functions;
  - comparison of error detection techniques (from related work and ours) from several perspectives (e.g. theoretical overhead costs, implementation overhead costs, security, error detection coverage).

    Publications: [67] [68].

## 7.3. Possible further research topics

Further research may address several topics not (completely) covered in this work

- analysis of (inter-) dependencies between security and dependability;
- targeting security implementations
  - evaluation of new proposed error detection techniques from other

side-channel attacks perspectives e.g. power analysis attack. Such an evaluation would clarify if these methods are or not generating other types of vulnerabilities;

- targeting MISTY1 algorithm
  - evaluation of other error detection techniques (e.g. nonlinear (cubic) error detection codes) for MISTY1 algorithm;
  - evaluation of error propagation for the control flow (key scheduling) of the algorithm; applying error detection mechanisms for control flow;
- targeting complemented duplication error detection technique
  - cost evaluation of this techniques for other cryptographic algorithms;
  - addressing other cryptographic algorithms (e.g. stream ciphers) from the perspective of fault analysis attack and evaluation of possible error detection techniques.

# ANNEX A. MISTY1 ALGORITHM, S-BOXES

S7[128] = {
27, 50, 51, 90, 59, 16, 23, 84, 91, 26,114,115,107, 44,102, 73,
31, 36, 19,108, 55, 46, 63, 74, 93, 15, 64, 86, 37, 81, 28, 4,
11, 70, 32, 13,123, 53, 68, 66, 43, 30, 65, 20, 75,121, 21,111,
14, 85, 9, 54,116, 12,103, 83, 40, 10,126, 56, 2, 7, 96, 41,
25, 18,101, 47, 48, 57, 8,104, 95,120, 42, 76,100, 69,117, 61,
89, 72, 3, 87,124, 79, 98, 60, 29, 33, 94, 39,106,112, 77, 58,
1,109,110, 99, 24,119, 35, 5, 38,118, 0, 49, 45,122,127, 97,
80, 34, 17, 6, 71, 22, 82, 78,113, 62,105, 67, 52, 92, 88,125};

S9[512] = {
451,203,339,415,483,233,251, 53,385,185,279,491,307, 9, 45,211,
199,330, 55,126,235,356,403,472,163,286, 85, 44, 29,418,355,280,
331,338,466, 15, 43, 48,314,229,273,312,398, 99,227,200,500, 27,
1,157,248,416,365,499, 28,326,125,209,130,490,387,301,244,414,
467,221,482,296,480,236, 89,145, 17,303, 38,220,176,396,271,503,
231,364,182,249,216,337,257,332,259,184,340,299,430, 23,113, 12,
71, 88,127,420,308,297,132,349,413,434,419, 72,124, 81,458, 35,
317,423,357, 59, 66,218,402,206,193,107,159,497,300,388,250,406,
481,361,381, 49,384,266,148,474,390,318,284, 96,373,463,103,281,
101,104,153,336, 8, 7,380,183, 36, 25,222,295,219,228,425, 82,
265,144,412,449, 40,435,309,362,374,223,485,392,197,366,478,433,
195,479, 54,238,494,240,147, 73,154,438,105,129,293, 11, 94,180,
329,455,372, 62,315,439,142,454,174, 16,149,495, 78,242,509,133,
253,246,160,367,131,138,342,155,316,263,359,152,464,489, 3,510,
189,290,137,210,399, 18, 51,106,322,237,368,283,226,335,344,305,
327, 93,275,461,121,353,421,377,158,436,204, 34,306, 26,232, 4,
391,493,407, 57,447,471, 39,395,198,156,208,334,108, 52,498,110,

202, 37,186,401,254, 19,262, 47,429,370,475,192,267,470,245,492,
269,118,276,427,117,268,484,345, 84,287, 75,196,446,247, 41,164,
14,496,119, 77,378,134,139,179,369,191,270,260,151,347,352,360,
215,187,102,462,252,146,453,111, 22, 74,161,313,175,241,400, 10,
426,323,379, 86,397,358,212,507,333,404,410,135,504,291,167,440,
321, 60,505,320, 42,341,282,417,408,213,294,431, 97,302,343,476,
114,394,170,150,277,239, 69,123,141,325, 83, 95,376,178, 46, 32,
469, 63,457,487,428, 68, 56, 20,177,363,171,181, 90,386,456,468,
24,375,100,207,109,256,409,304,346, 5,288,443,445,224, 79,214,
319,452,298, 21, 6,255,411,166, 67,136, 80,351,488,289,115,382,
188,194,201,371,393,501,116,460,486,424,405, 31, 65, 13,442, 50,
61,465,128,168, 87,441,354,328,217,261, 98,122, 33,511,274,264,
448,169,285,432,422,205,243, 92,258, 91,473,324,502,173,165, 58,
459,310,383, 70,225, 30,477,230,311,506,389,140,143, 64,437,190,
120, 0,172,272,350,292, 2,444,162,234,112,508,278,348, 76,450 };

# REFERENCES


[1] L.J. Hoffman, K. Lowson-Jenkins, J. Blum, "Trust beyond security: An expended trust model", *Communications of the ACM,* Vol. 94, No. 7, July, 2006, pp. 95-101.

[2] M. Barbacci, M. Klein, T. Longstaff, C. Weinstock, "Quality Attributes," CMU/SEI-95-TR-021 report, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University, 1995, available at: www.sei.cmu.edu/pub/documents/95.reports/pdf/tr021.95.pdf, last visited November 2005.

[3] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996, available at: http://www.cacr. math.uwaterloo.ca/hac/, last visited November 2006.

[4] C.H. Hauser, D.E. Bakken, A. Bose, "A Failure to Communicate," *IEEE Power & Energy Magazine*, Vol. 3, No. 2, March, 2005, pp. 47-55.

[5] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of dependable and secure computing," *IEEE Trans. On Dependable and Secure Computing*, Vol. 1, No.1, January-March, 2004, pp. 11-33.

[6] D.K. Pradhan, *Fault-tolerant Computer System Design*, Computer Science Press, Prentice Hall, second print, 2003.

[7] B. Schneier, *Applied Cryptography*, John Wiley & Sons, New York, 1996.

[8] RSA Security web site: http://www.rsasecurity.com/

[9] E. Oswald, B. Preneel, "A theoretical evaluation of some NESSIE candidates regarding their susceptibility towards power analysis attacks", NESSIE report, 2002.

[10] NESSIE consortium, NESSIE security report, Version 2.0, February 19, 2003, available at: https://www.cosic.esat.kuleuven.ac.be/ nessie/deliverables/D20-v2.pdf, last visited November 2006.

[11] P. Verissimo, L. Rodrigues, *Distributed Systems for System Architects*, Kluwer Academic Publishers, 2001.

[12] M. W. Maier, "Architecting principles for systems-of-systems," Systems Engineering, Vol. 1, No. 4, 1998, pp. 267-284.

[13] G. Deconinck, T. Rigole, K. Vanthournout, **R. Tirtea** et al., "Embedded automation for energy applications and its interdependence with the info'structure," *2006 IEEE International Conference on Systems, Man, and Cybernetics (SMC'06)*, Taipei, Taiwan, October 8 - 11, 2006, pp. 575-579.

[14] **R. Tirtea**, G. Deconinck, R. Belmans, "Overview of Monitoring Mechanisms Used in the Grid," *Proc. of 8th Int. Conf. on Engineering of Modern Electric Systems (EMES)*, Oradea, Romania, May, 2005, 6 pages.

[15] **R. Tirtea**, "Integration at middleware level of fault tolerance for distributed systems," PhD thesis, K.U.Leuven, ISBN 90-5682-650-6, December, 2005.

[16] **R. Tirtea**, G. Deconinck, "On Dependencies between Fault Tolerance and Security in Distributed Systems," *Proc. of First CRIS Int. Workshop on Critical Information Infrastructures (CIIW'05)*, Linköping, Sweden, May, 2005, pp. 27-33.

[17] **R. Tirtea**, G. Deconinck, R. Belmans, "Fault tolerance adaptation requirements vs. quality-of-service, real-time and security in dynamic distributed systems," *Proc. of Annual Reliability & Maintainability Sym. (RAMS)*, Newport Beach, CA, USA, January, 2006, 7 pages.

[18] MAFTIA project homepage, available at: http://www.MAFTIA.org/, last visited November 2006.

[19] R. Stroud, I. Welch, J. Warne, P. Ryan, "A Qualitative Analysis of the Intrusion-Tolerance Capabilities of the MAFTIA Architecture," *Proc. of Int. Conf. on Dependable Systems and Networks (DSN'04)*, Florence, Italy, June - July, 2004, pp. 453-464.

[20] M. Cukier, T. Courtney, J. Lyons, H. V. Ramasamy, W. H. Sanders, M. Seri, M. Atighetchi, P. Rubel, C. Jones, F. Webber, P. Pal. R. Watro, and J. Gossett, "Providing Intrusion Tolerance With ITUA", *Suppl. Vol. Of Int. Conf. on Dependable Systems and Networks (DSN'02)*, Washington, DC, June, 2002, pp. C-5-1 to C-5-3.

[21] National Institute of Standards and Technology, NIST Report, September 12, 1997 (Volume 62, Number 177). Docket No. 970725180-7180-01.Pages 48051-48058.

[22] NIST Report on the Development of the Advanced Encryption Standard (AES), James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, and co., October 2, 2000.

[23] T. Messerges, "Securing the AES Finalists Against Power Analysis Attacks," *Preproc. Fast Software Encryption Workshop,* April 10-12, 2000.

[24] NESSIE, Call for Cryptographic Primitives, Version 2.2, 8th March 2000, available at: https://www.cosic.esat.kuleuven.ac.be/ nessie/call/, last visited November 2006.

[25] NESSIE consortium, NESSIE project announces final selection of crypto algorithms, February 27, 2003, available at: https://www.cosic.esat.kuleuven.ac.be/nessie/deliverables/ press_release_feb27.pdf, last visited November 2006.

[26] B. Preneel, "Strengths and weaknesses of information security standards", course materials, *Graduate School in Electronics and Communications*, Louvain-la-Neuve, May 11-13, 2004.

[27] CRYPTEC site, Evaluation of Cryptographic Techniques, available at: http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html

[28] Y.B. Zhou, D.G. Feng, "Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing", Cryptology ePrint Archive, Report 2005/388, 2005, available at: http://eprint.iacr.org/2005/388.

[29] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks", Cryptology ePrint Archive, Report 2004/100, 2004, av. at: http://eprint.iacr.org/2004/100.pdf.

[30] D. Boneh, R. DeMillo, R. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Eurocrypt '97*, *LNCS*, Vol. 1233, Springer-Verlag, 1997, pp. 37-51.

[31] D. Boneh, R. DeMillo, R. Lipton, "On the importance of checking cryptographic protocols for faults," *Journal of Cryptology*, Springer-Verlag, Vol. 14, No. 2, 2001, pp. 101-119.

[32] E. Biham, A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *Advances in Cryptology - CRYPTO '97, LNCS*, Vol. 1294, Springer-Verlag, 1997, pp. 513-525.

[33] "Advanced Encryption Standard (AES)", Federal Information Processing Standard (FIPS), Publication 197, National Bureau of Standards, US Department of Commerce, Washington D.C., December 2001.

[34] "DES Modes of Operation", Federal Information Processing Standard (FIPS), Publication 81, National Bureau of Standards, US Department of Commerce, Washington D.C., December 1980.

[35] ISO/IEC 10116, "Information technology - Security techniques - Modes of operation of an n-bit block cipher algorithm," IS 10116, 1991.

[36] IETF, IPsec Working Group, "Using AES Counter Mode With IPsec ESP," Request for Comments 3686, 2004, available at: http://rfc3686.x42.com/, last visited November 2006.

[37] The ATM Forum Technical Committee, ATM Security Specification Version 1.1, af-sec-0100.002, March 2001, available. at ftp://ftp.atmforum.com/pub/approved-specs/af-sec-0100.002.pdf, last visited November 2005.

[38] American National Standards Institute, American National Standard X3.106-1983 (R1996), Data Encryption Algorithm, Modes of Operations for DES, 1983.

[39] NIST Special Publication 800-38A 2001 Edition, Recommendation for Block Cipher Modes of Operation, Methods and Techniques, available at: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf, last visited November 2006.

[40] W. Diffie, M. Hellman, "Privacy and Authentication: An Introduction to Cryptography," *Proceedings of the IEEE*, no.67, 1979, pp. 397–427.

[41] H. Lipmaa, P. Rogaway, D. Wagner, "Comments to NIST concerning AES Modes of Operation: CTR-Mode Encryption," *Symmetric Key Block Cipher Modes of Operation Workshop*, 2000, available at: www.tcs.hut.fi/~helger/papers/lrw00/html/

[42] H. Heys, "Analysis of the Statistical Cipher Feedback Mode of Block Ciphers," *IEEE Tran. On Computers,* Vol. 52, No. 1, Jan. 2003, pp. 77-92.

[43] **R. Tirtea**, G. Deconinck, "Specifications overview for counter mode of operation. Security aspects in case of faults," *Proc. of 12th IEEE Mediterranean Electrotechnical Conference (MELECON 2004)*, Dubrovnik, Croatia, May, 2004, 5 pages.

[44] Barry W. Johnson, *Design & analysis of fault tolerant digital systems*, Addison-Wesley Longman Publishing Co., Inc., 1989.

[45] R. Karri, K. Wu, P. Mishra, Y. Kim, "Concurrent Error Detection of Fault Based Side-Channel Cryptanalysis of 128-Bit Symmetric Block Ciphers," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 21, No. 12, December, 2002, pp. 1509-1517.

[46] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri, "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard," *IEEE Trans. on Computers*, Vol. 52, No. 4, April, 2003, pp. 492-505.

[47] M. Karpovsky, K. J. Kulikowski, A. Taubin, "Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard," *Proc. of Int. Conf. on Dependable Systems and Networks (DSN'04)*, Florence, Italy, June - July, 2004, pp. 93-102.

[48] NIST, Announcing Proposed Withdrawal of Federal Information Processing Standard (FIPS) for the Data Encryption Standard (DES) and Request for Comments, 2004, available at: http://edocket. access.gpo.gov/2004/04-16894.htm

[49] **R. Tirtea**, G. Deconinck, "Fault Detection Mechanisms for Fault Analysis Attacks Resistant Cryptographic Architecture," *Proc. of Third IEEE Int. Conf. on Systems, Signals & Devices SSD'05,* March, 2005, Sousse, Tunisia, 6 pages.

[50] **R. Tirtea**, "Using Cryptographic Algorithms Properties for Fault Tolerance Purposes," *Proc. of 6th Int. Conf. of Renewable Sources and Environmental Electro-Technologies (RSEE)*, 8 - 10 June 2006, Oradea, Romania, 4 pages.

[51] M. Matsui, "New Block Encryption Algorithm MISTY", *Fast Software Encryption Workshop FSE'97*, Lecture Notes in Computer Science no. 1267, Springer-Verlag, 1997.

[52] NESSIE, List of accepted NESSIE submissions, available at: https://www.cosic.esat.kuleuven.ac.be/nessie/workshop/submissions.html, last visited November 2006.

[53] MISTY1 website: http://www.mitsubishi.com/ghp_japan/misty/licensee.htm

[54] NESSIE consortium Portfolio of recommended cryptographic primitives, February 27, 2003 available at: https://www.cosic.esat.kuleuven.ac.be/nessie/deliverables/decision-final.pdf, last visited November 2006.

[55] Rouvroy G, Standaert F-X, Quisquater J-J, Legat J-D. "Efficient FPGA Implementation of Block Cipher MISTY1", *Proc. of the 10th Reconfigurable Architectures Workshop (RAW)*, Nice, France, 2003, 7 pages.

[56] M. Matsui, Supporting Document of MISTY1, available from: https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions.htm, last visited May 2006.

[57] H. Tanaka, K. Hisamatsu, T. Kaneko, "On the strength of MISTY1 without FL functions against higher order differential attack," *Proc. Of AAECC'99*, no. 1719 in Lecture Notes in Computer Science, Springer-Verlag, 1999, pp. 221-230.

[58] E. Biham, A. Biryukov and A. Shamir, "Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials," *Proc. of Eurocrypt'99*, no. 1592 in Lecture Notes in Computer Science, Springer-Verlag, 1999, pp. 12-23.

[59] A. Biryukov and D. Wagner, "Slide attacks," in *Proc. of Fast Software Encryption FSE'99,* no. 1636 in Lecture Notes in Computer Science, Springer-Verlag, 1999, pp. 245-259.

[60] Virtex™ Field Programmable Gate Arrays, Product Specification DS003-1 (v2.5), 2001, available at: http://direct.xilinx.com/bvdocs/publications/ds003.pdf, last visited September 2006.

[61] T. R. N. Rao, E. Fujiwara, *Error-Control Coding for Computer Systems*, Prentice-Hall, 1989.

[62] **R. Tirtea**, G. Deconinck, V. De Florio, R. Belmans: "QoS monitoring at middleware level for dependable distributed automation systems," *Proc. 13th Int. Symp. on Software Reliability Engineering*, Annapolis, Maryland, USA, November, 2002, pp. 217-218.

[63] **R. Tirtea**, "Integration of adaptable fault management techniques into a dependable middleware architecture," *Suppl. Vol. of Int. Conf. on dependable systems and networks (DSN-2004)*, Florence, Italy, June-July, 2004, pp. 184-186.

[64] **R. Tirtea**, G. Deconinck, R. Belmans, "Cost analysis of adaptive fault management," *Proc. Of Annual Reliability & Maintainability Symp. (RAMS)*, Alexandria, Virginia, USA, January, 2005, 7 pages.

[65] **R. Tirtea**, G. Deconinck, V. De Florio, "Le logiciel d'intermédiation: Un panorama critique et réutilisation dans les systèmes d'automatisation enfouis à sûreté de fonctionnement," *Genie logiciel*, Paris, No. 61, June, 2002, pp. 2-8.

[66] **R. Tirtea**, G. Deconinck, V. De Florio, R. Belmans, "Using resource monitoring to select recovery strategies," *Proc. Ann. Reliability & Maintainability Symp.,* Los Angeles, USA, January, 2004, pp. 266-271.

[67] **R. Tirtea**, M. Vladutiu, "Protection of Cryptographic Implementations using Error Detection Techniques", accepted for publication in *Proc. Of Int. Conf. on Digital Communications and Computer Applications (DCCA2007)*, Jordan, March, 19-22, 2007, 6 pages.

[68] **R. Tirtea**, M. Vladutiu, G. Deconinck, "Low Cost Self-Testing Implementation for MISTY1 Cryptographic Algorithm", under review for *5th IEEE International Conference on Industrial Informatics (INDIN)*, Vienna, Austria, July 23-26, 2007, 6 pages.