Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

MSc THESIS

Profiling Symmetric Encryption Algorithms for Implantable Medical Devices

Di Zhu

CE-MS-2008-20

Abstract

The amount of Implantable Medical Devices (IMD) keeps booming in recent years. However, not many of them have encryption and decryption methods to protect their data communication. The purpose of this thesis work is to profile various popular symmetric encryption algorithms and select the best candidates as the benchmark for generic IMD. Moreover, this thesis gives suggestions for design digital CPU for generic IMD based on lab work simulation output.
This thesis work will focus on profiling and analyzing several metrics: average power consumption, peak power consumption and energy consumption of all candidates because those metrics are the crucial constraint of IMD. Also the background information of all candidates will be introduced and all relevant metrics such as security level, encryption rates of all candidates will be discussed in this thesis work.

Key Words: Power Consumption, Implantable Medical Devices, symmetric encryption algorithm, security level

Faculty of Electrical Engineering, Mathematics and Computer Science

Profiling Symmetric Encryption Algorithms for Implantable Medical Devices

THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Embedded Systems

by

Di Zhu
born in Beijing, China

Embedded Systems
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Profiling Symmetric Encryption Algorithms for Implantable Medical Devices

by Di Zhu

Abstract

The amount of Implantable Medical Devices (IMD) keeps booming in recent years. However, not many of them have encryption and decryption methods to protect their data communication. The purpose of this thesis work is to profile various popular symmetric encryption algorithms and select the best candidates as the benchmark for generic IMD. Moreover, this thesis gives suggestions for design digital CPU for generic IMD based on lab work simulation output.
This thesis work will focus on profiling and analyzing several metrics: average power consumption, peak power consumption and energy consumption of all candidates because those metrics are the crucial constraint of IMD. Also the background information of all candidates will be introduced and all relevant metrics such as security level, encryption rates of all candidates will be discussed in this thesis work.

Laboratory	:	Computer Engineering
Codenumber	:	CE-MS-2008-20

Committee Members :

Advisor:	Georgi Gaydadjiev
Advisor:	Christos Strydis
Chairperson:	Koen Bertels
Member:	Koen Langendoen
Member:	Henk Corporaal

Contents

List of Figures VIII

List of Tables X

1. Introduction

1.1 Background Information ..… 1
1.2 Thesis motivation …….. 2
1.3 Project Work ……. 4
1.4 Report Organization ……. 6

2. Background Concepts: Symmetric Ciphers

2.1 Taxonomy of Cryptography ……. 7
2.2 Terminology of Symmetric Cryptography Algorithms …….. 8
2.3 Analysis of Symmetric-encryption Algorithms …….13
2.3.1 DES …...14
2.3.2 BLOWFISH ……. 15
2.3.3 3-WAY ……. 17
2.3.4 GOST …... 19
2.3.5 IDEA…..21
2.3.6 LOKI91…..24
2.3.7 RC5 …...26
2.3.8 RC6 …...28
2.3.9 SKIPJACK …....30
2.3.10 XXTEA …….32
2.3.11 MISTY1 …….33
2.3.12 TWOFISH …….35
2.3.13 RIJNDAEL(AES) …….37
2.4 Symmetric Cryptography Modes…... 39
2.4.1 ECB …...39
2.4.2 CBC …..40
2.4.3 CFB ….. 41
2.4.4 OFB ….. 41

3. Profiling and Analysis

3.1 Profiling System Overview …….43
3.3.1 Profiling Metrics…….43
3.3.2 Profiling scenario …..43
3.3.3 Operation Mode Selection …….44
3.3.4 Input Datasets …….44
3.2 Experimental Setup …….45
3.2.1 The Simulator …….45

3.2.2 The Encryption Algorithm ……………………………………………………46
3.2.3 The Selected Benchmark Suite …...……………………………………… 47
3.3 Profiling Analysis ……………………………………………………………..48
3.3.1 Power Consumption ………………………………………………………. 48
3.3.2 Energy Consumption ………………………………………………………51
3.3.3 Efficiency …………………………………………………………………. 54
3.4 Cipher Selection and Result ………………………………………………….. 55

4. Conclusion
4.1 Conclusions ……………………………………………………………………60
4.2 Future Work …………………………………………………………………...60
4.3 Possible Improvements ………………………………………………………..61

Bibliography **63**

List of Figures

Figure 1.1. A Typical Implantable Medical Device………………………………………………. 3
Figure 2.1. Taxonomy of Cryptography……...………………………………………………….8
Figure 2.2. A Typical Encryption Procedure………………………………………………......9
Figure 2.3. The Encryption and Decryption Procedure of a Feistel Cipher…..……………………11
Figure 2.4. Flow Chart of DES……………………………………………………………….15
Figure 2.5. Flow Chart of Blowfish……………………………………………………... 17
Figure 2.6. Flow Chart of 3-WAY………………………………………………………… 19
Figure 2.7. Diagram of GOST ………………………………………………………… 20
Figure 2.8. Flow Chart of GOST……………………………………………………………21
Figure 2.9. Flow Chart of IDEA………………………………………………………………..23
Figure 2.10. Flow Chart of LOKI91………………………………………………………25
Figure 2.11. Diagram of LOKI91……………………………………………………..……26
Figure 2.12. Flow Chart of RC5……………………………………………………………27
Figure 2.13. Diagram of RC5……………………………………………………………...28
Figure 2.14. Flow Chart of RC6…………………………………………………….. …29
Figure 2.15. Flow Chart of SKIPJACK…………...……………………………………….31
Figure 2.16. Flow Chart of XXTEA…………………………………………………..……32
Figure 2.17. Flow Chart of MISTY1…………………………………………………...34
Figure 2.18. Flow Chart of TWOFISH……………………………………………………36
Figure 2.19. Flow Chart of RIJNDAEL…………………………………………………......38
Figure 2.20. ECB mode….………………………..40
Figure 2.21. CBC mode …………………………………………………………………40
Figure 2.22. CFB mode……………………………………….......................................41
Figure 2.23. OFB mode…………………………………………………………………42
Figure 3.1. Per-component, average power consumption (in mW) for two plaintext sizes………50
Figure 3.2. Average and peak power consumption (in mW) for two plaintext sizes……….…….51
Figure 3.3. Figure 3.3 Per-component and total encryption energy costs (in Joules)…………….53
Figure 3.4. Computation overhead of ciphers manifested as energy penalty (in Joules) when encrypting one 10-KB and ten 1-KB plaintexts…………………………………...…..54
Figure 3.5. Encryption rate (in KB/sec)……………………………………………………..55
Figure 3.6. Figure 3.6 u-op mix and frequencies for MISTY1 operating on the 1-KB BP plaintext…………………………………………………………………………..58

List of Tables

Table 3.1. Biomedical workloads used for profiling. ……………………………………………….44

Table 3.2. XScale architecture details ……………………………………………………………..46

Table 3.3. Benchmark suite of symmetric ciphers………………………………...........................47

Table 3.4. Five best-performing encryption algorithms (in descending order of performance)……………………………………………………………….……………...56

Table 3.5. the rank of best-performing encryption algorithms (according to the occurrences rate)………………………………………………………..…………………………..56

Table 3.6. Sample ARM instruction which stores registers in the stack pointed to by R13 and equivalent ARM u - OP sequence………………………………….……………….…..57

Table 3.7. MISTY1 on 01_KB……………………………………………..……………………….57

Table 3.8. u-op mix and frequencies for IDEA and RC6 operating on the 1-KB BP plaintext……59

Chapter 1

Introduction

This chapter introduces to the content of the thesis project work "Profiling Symmetric Encryption Algorithms for Implantable Medical Devices". The chapter is divided into several sections discussing the project motivation, the proposed solution, and the main challenges of the project. In addition, the organization of this project report is presented.

1.1 Background Information

An embedded system is a special-purpose computer system designed to perform one or few dedicated functions, often with real time computing constraints. It is usually embedded as part of a complex system including software, hardware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Embedded systems control many devices commonly in use today.

Embedded systems span many aspects of the modern life and there are many examples of their usage. Medical equipment is a class which advances continuously with an ever increasing number of more embedded systems for vital-signs monitoring, electronic stethoscopes for amplifying sounds, and various medical imaging (such as PET, SPET, CT and MRI) modalities for non-invasive endoscopy. One particular class of medical equipment is the microelectronics implants. Starting with the implantable pacemaker some 50 years ago, such devices have been increasingly improving over the last two decades. Implantable medical devices are made to replace and act as a missing biological structure, e.g. an implantable pacemaker. The primary purpose of a pacemaker is to maintain an adequate heart rate, either because the heart's native pacemaker of some people such as heart failure patients is not fast enough, or there is a block in the heart's electrical conduction system for some people such as myocardial infarction patients. The pacemaker contacts the heart muscles to regulate the beating of the heart. Other therapeutic implantable devices have followed, including bone-growth stimulators, drug-infusion pumps and cochlear implants that provide a semblance of hearing for the deaf. Nowadays, implantable medical devices are being widely used in many organs of the human body and their number keeps growing. In Europe alone, a total number of 299,705 implanted devices have been registered over the year 2003 (source: European Society of Cardiology [1]). According to the latest issue, the microelectronics implant market is expected to grow by at least 30% in the next 10 years.

1.2 Thesis Motivation

Typical, modern microelectronic implants comprise a central unit commonly used for data processing tasks (e.g. compression of sensor readout data) as well as for controlling all peripheral units of the device [2]. Such units usually are:

- Sensors and actuators, used to physically interface to the living tissue,
- A wireless transducer to allow (bidirectional) communication between the implant (in vivo) and an external controller (ex vivo) such as desktop computer, a palmtop, a laptop or, even, a mobile phone, and
- A power module which provides the DC power needed for the operation of the implant electronics.

Over the wireless interface of a microelectronic implant, inbound (to the implant) commands carry operational information, requests for data acquisition or any kind of effectuating on the living tissue (i.e. actuation) and so on. Outbound data streams contain implant responses to the external queries, operation-status messages and so on[2]. In figure 1 a conceptual view of a generic microelectronic implant is presented.

The Health Insurance Portability and Accountability Act (HIPPA) in the USA [3] requires that the patient is the only owner of and should fully control the disclosure policy for his Personal Health Record (PHR). Any malicious acquisition of patient personal-health records will be treated as illegal. In the European Union (EU) [4], several Directives of the European Parliament and of the Council protect the processing and free movement of personal data, including that for health care purposes.

However, very few implantable medical devices are currently designed to actively deal with security issues. Recently, some researchers from the University of Washington and the University of Massachusetts have been able to gain wireless access to an implantable heart-defibrillator and pacemaker device [5]. Once they had access, the researchers found a way to reprogram the device to deliver (undesired) electric jolts to the heart tissue.

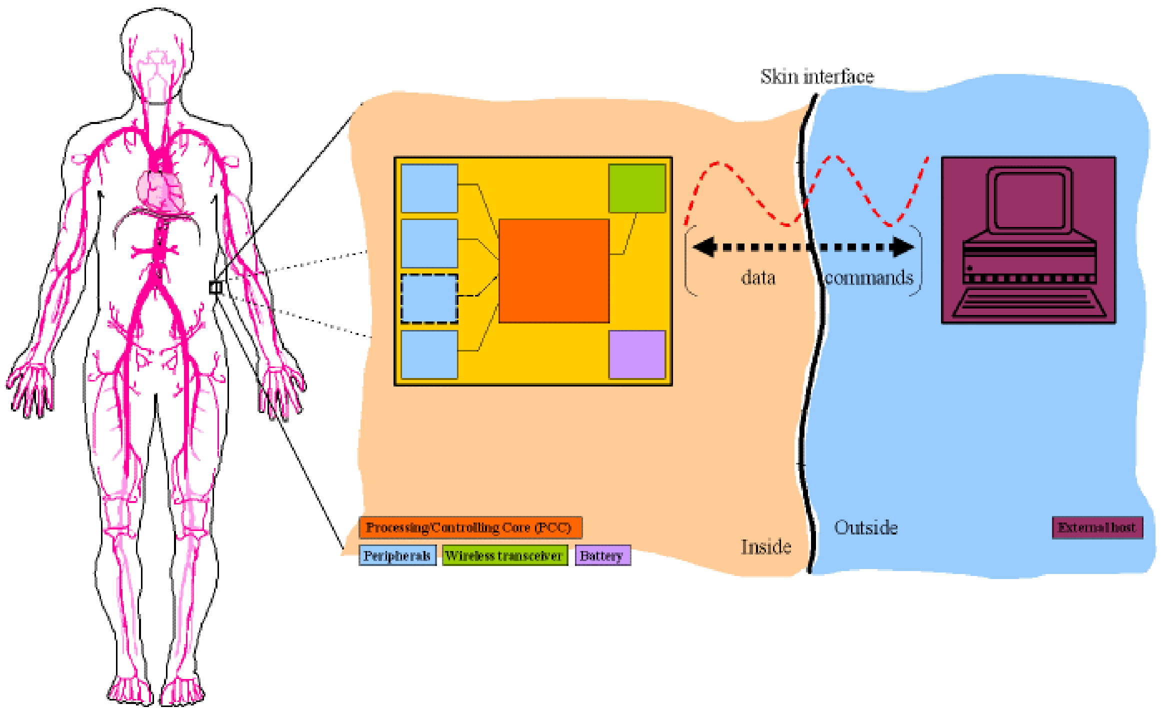

Figure 1: A Typical Implantable Medical Device [2]

While the risk for wireless attacks carried out against patients wearing such devices today remains low, because most of the radios involved only transmit over very short distances, the ability to interfere with transmission performance has been already illustrated. Furthermore, as the number of medical implants well increases in the future, and the implants devices will become more widely adopted more attention should be drawn on this issue. While newer devices are already being equipped with more powerful radios, for instance [6], patients should now, consider carefully if they would really be comfortable with implanting such a hack-prone devices in their body. Unfortunately, there are plenty of people who wouldn't necessarily have a choice.

Conclusively, protecting the privacy as well as the very health of patients is anticipated to be one of the most serious aspects of implant wireless communications in the years to come. In this context, microelectronic implants definitely require encryption of inbound and outbound traffic. Data needs to be relayed securely for protecting the personal biological information of patients. Conversely, commands to the implants need to be properly authenticated, protected and verified to be originating from legitimate operators so as to avoid unwanted, malicious or even fatal in-vivo manipulation of the devices.

1.3 Project Work

As mentioned before, implantable medical devices are a particular class of embedded systems. Designing medical implants suffers from the same challenging issues as designing other embedded systems, such as miniature size, low power consumption, constrained memory size, etc. As a matter of fact, medical implants have much stricter requirements than generic embedded systems. For instance, an implantable pacemaker is normally implanted into the human body for at least 10 years [7]. Because of the limited battery capacity, the pacemaker has to minimize its power consumption in order to continue working such a long time. Clearly, battery replacement (through a risky and costly explantation procedure) is not a viable option. Moreover, the pacemaker has to be manufactured as small as possible in order to match the tight physical constraints inside the human body. In the Computer Engineering Group, we have established the Smart Implantable Medical Systems (SiMS) project that focuses on low power consumption problem among other issues.

The SiMS project (http://ce.et.tudelft.nl/SiMS) is currently defining the architecture of a novel, minimalistic, low-power digital processor suitable for a large subset of biomedical applications. For defining this processor, various software applications have been employed as benchmarks to guide the design process. One of the examined benchmark suites, and a very important one as previously discussed, is data encryption.

In this thesis work, we study, analyze, collect and profile a large set of popular encryption algorithms in terms of power consumption, energy expenditure, encryption rate and program-code size through detailed simulations. We only choose to profile symmetric encryption algorithms for three reasons:

- Firstly, asymmetric encryption algorithms have been found to have computational and memory requirements that are prohibitively high for low-power embedded devices [8, 9]. Since we care more about memory constrained scenario in our SiMS project, we only concentrate on symmetric encryption algorithms. Nevertheless we will address the asymmetric encryption algorithms in our future work. In addition, related work in the field of embedded systems especially in wireless sensor network (WSNs) showed that carefully optimized software or hardware implementations of existing asymmetric algorithms may be viable for resource-constraint devices [10, 11].
- Secondly, our choice is based on typical application scenario for the targeted implants: data and command exchange with the implantable medical device does not happen particularly often, e.g. once a day. The reason for that primarily is the desired autonomous, unattended operation of such devices as well as the disproportionably large energy costs incurred when wirelessly transmitting data in-vivo. In effect, even if a combination of asymmetric and symmetric-key encryption is assumed for secure authentication and actual-data exchange, respectively, authentication is not expected to occur so often in an implant operation. It is, thus, not our primary concern for this study. The scope of this thesis is focusing on the most commonly executed task, i.e. the symmetrically-encrypted data exchange.

After the study, analysis and evaluation of all symmetric encryption candidates, we will selected the best performing ones for the targeted application domain and investigate their respective instruction mixes in order to gain further insight on the most suitable instructions to be included in our targeted digital processor architecture. Concisely, the main contributions of this thesis work are:

- To identify encryption algorithms which achieve the lowest average power consumption throughout their execution;
- To identify encryption algorithms with the overall lowest total energy budget for encryption various plaintext sizes;
- To identify encryption algorithms which encrypt various plaintext sizes at the highest possible rate or, at least, at a rate fast enough to satisfy the required sampling rate of the (biological) plaintext data;

To identify the instruction mixes and frequencies of the best scoring encryption algorithms for including in the architecture of the new processor currently designed for a variety of biomedical implants.

- To make the statement that encryption is a crucial, much needed and feasible provision for future microelectronic implants.

1.4 Report Organization

The background information of all candidate symmetric algorithms, profiling platform, benchmark suite and assumptions made in our thesis study are detailed in the following chapters:

Chapter 2 gives the history and the background knowledge of various widely accepted encryption algorithms. The analysis of each algorithm and our selection process will be also explained in this chapter. Chapter 3 describes experiment lab establishment and discusses our results. Chapter 4 draws the conclusion of this thesis work. The future work of the SiMS project is also discussed in this chapter.

Chapter 2

Background Concepts: Symmetric Ciphers

This chapter discusses the background concepts of the current project: "Profiling Symmetric Cryptography Algorithms for Implantable Medical Devices". Basic cryptography theory and several symmetric cryptography algorithms will be discussed and analyzed. Cipher architectures, implementation details, security levels and other information is provided. The reasons for studying only symmetric ciphers are related to the context of the SiMS project and will be explained in this chapter as well.

2.1 A taxonomy of Cryptography

Cryptography algorithms have the same old history as civilization. One famous example of using cryptography algorithms is Julius Caesar's use of an alphabetic substitution cryptographic system to send secure information to his army [12]. Another famous example is the Hagelin cryptograph. It was made by Swede Boris Hagelin in the 1930's and was used by the American army until 1950 [13]. Modern cryptography can be divided into two main subfields of study: Symmetric-key and Asymmetric-key cryptography. Symmetric-key can be divided into block ciphers and stream ciphers. Figure 2.1 depicts the taxonomy of cryptography.

Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key (or, less commonly, in which their keys are different, but related in an easily computable way). This was the only kind of encryption publicly known until June 1976 [14]. The modern study of symmetric-key ciphers relates mainly to the study of block ciphers and stream ciphers and their applications. We will discuss the terminology involved in symmetric-key cryptography in the next section.

Asymmetric-key cryptography is also known as public-key cryptography whereby two different but mathematically related keys are used — a public key and a private key [15]. A public-key system is so constructed that calculation of one key (the 'private key') from the other (the 'public key') is computationally infeasible, even though they are necessarily related. Instead, both keys are generated secretly, as an interrelated pair [16]. The public key may be freely distributed, while its paired private key must remain secret. The public key is typically used for encryption, while the private or secret key is used for decryption. The most famous applications of public-key cryptography are Elliptic-curve cryptography, PGP and the public-key infrastructure (PKI).

Elliptic Curve Cryptography (ECC) provides the highest strength-per-key-bit of any cryptography algorithm known to take. Compared with other public-key approaches, ECC not only has the higher security but also has lows computation overhead, shorter key size and narrower bandwidth. Therefore, the experts believe that ECC will become the next generation widely used public-key cryptography.

Pretty Good Privacy (PGP) is a computer program that provides cryptographic privacy and authentication. PGP is often used for signing, encrypting and decrypting e-mails to increase the security of e-mail communication.

A PKI (public key infrastructure) enables users of a basically unsecured public network such as the Internet to securely and privately exchange data and money through the use of a public and a private cryptographic key pair that is obtained and shared through a trusted authority.

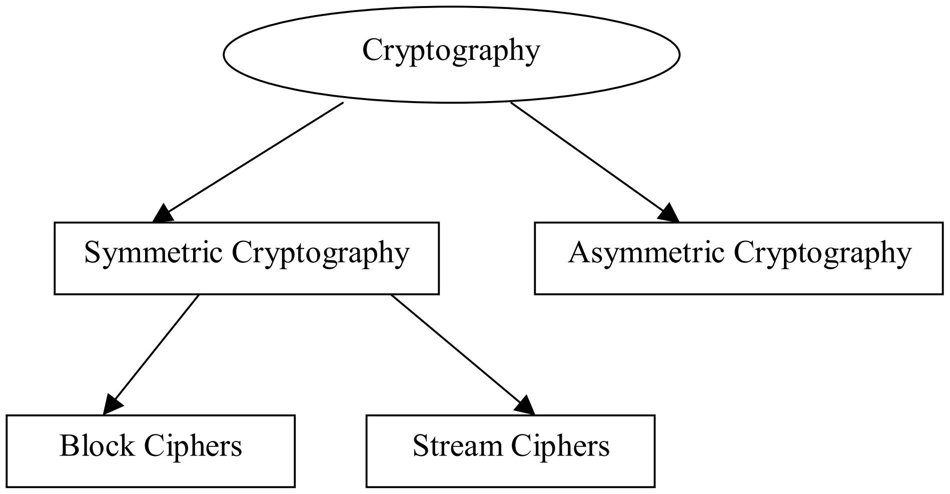

Figure 2.1: Taxonomy of Cryptography

2.2 Terminology of Symmetric Cryptography Algorithms

The terminology of symmetric cryptography algorithms mainly includes the following:

Plaintext: Plain text is the ordinary information which the sender wishes to transmit to the receiver(s).

Cipher text: The encrypted text is called Cipher text.

Encryption and Decryption: Encryption is the process of converting plain text into ciphertext. Decryption is the reverse process, moving from ciphertext back to the original plain text.

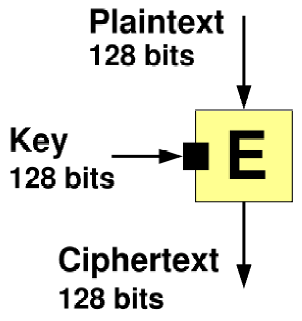

Figure 2.2: A Typical Encryption Procedure

Cipher: A cipher is a pair of algorithms which ensure the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the algorithm and by a specific key.

Key: The key is a secret parameter for encrypting or decrypting a specific message-exchange context. Keys are important, as ciphers without keys are trivially breakable and therefore less than useful for most purposes.

Block Ciphers: The block cipher is a type of symmetric-key encryption algorithm that transforms a fixed-length block of plain text data into a block of cipher text data of the same length. This transformation takes place under the action of a user-provided secret key. Decryption is performed by applying the reverse transformation to the cipher text block using the same secret key. The fixed length is called the block size and, for many block ciphers, the block size is 64 bits. In the coming years, the block size will increase to 128 bits as processors become more sophisticated. Since messages are almost always longer than a single block, some method of knitting together successive blocks is required. The different ways of knitting together blocks are known as the modes of operation and must be carefully considered when using block ciphers. Mode of operation will be further elaborated in section 4. Block ciphers can be easier to implement in software than stream ciphers, because they often avoid time-consuming bit manipulations and they operate on data in computer-sized blocks [17].

Stream Ciphers: A stream cipher is a symmetric-key cipher where plaintext bits are combined with a pseudo-random cipher bit-stream (key stream), typically by an exclusive-or (xor) operation. In a stream cipher the plaintext digits are encrypted one at a time, and the transformation of successive digits varies during the encryption. Stream ciphers typically execute at a higher speed than block ciphers and have lower hardware complexity [18]. Stream ciphers that only encrypt and decrypt data one bit at a time are not really suitable for software implementation [17]. This explains why stream ciphers can be better implemented in hardware than block ciphers.

Feistel - Cipher: A Feistel cipher is a symmetric structure used in the construction of block ciphers; it is also commonly known as a Feistel network. A large proportion of block ciphers use the scheme, including Blowfish, DES, MYSTY1, RC5, TWOFISH, XXTEA, GOST, etc. The Feistel structure has the advantage that encryption and decryption operations are very similar, even identical in some cases, requiring only a reversal of the key schedule. Therefore, the size of the code or the circuitry required to implement such a cipher is nearly halved. Feistel networks combine multiple rounds of repeated operations, such as:

- Bit-shuffling functions (often called permutation boxes or P-boxes) ;
- Simple, non-linear functions (often called substitution boxes or S-boxes);
- Linear mixing (in the sense of modular algebra) using XOR operations.

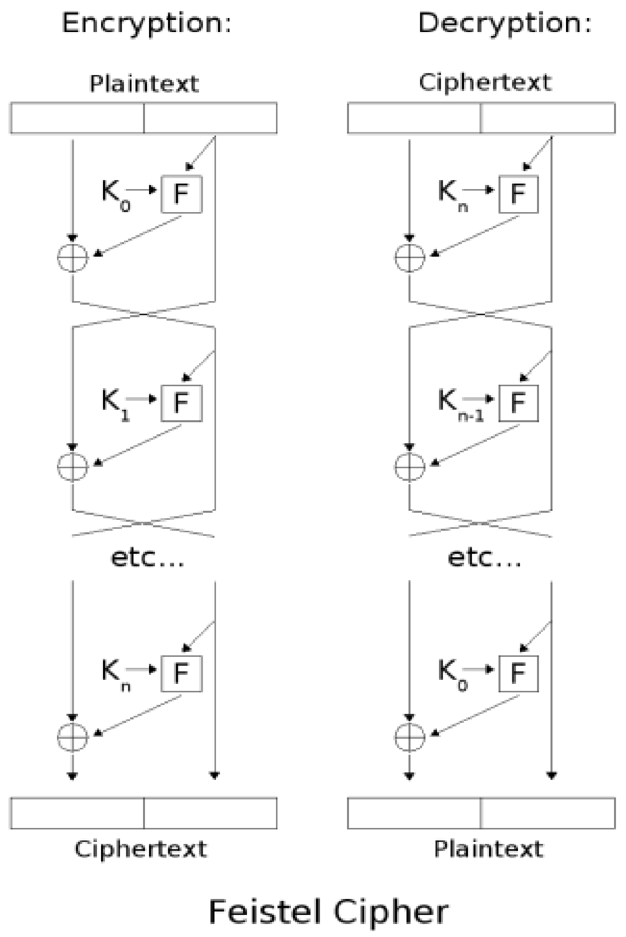

Figure 2.3: The Encryption and Decryption Procedure of a Feistel Cipher

S-Box: The substitution box (S-box) is a basic component of many symmetric key algorithms. In block ciphers, they are typically used to obscure the relationship between the plaintext and the cipher text; they rely on the Shannon's property of confusion [19]. In many cases, the S-boxes are carefully chosen to resist cryptanalysis. In general, an S-box takes some number of input bits, *m*, and transforms them into some number of output bits, *n*: an *m*×*n* S-box can be implemented as a lookup table with *2m* words of n bits each. Fixed tables are normally used, as in the Data Encryption Standards (DES), but in some ciphers the tables are generated dynamically from the key; e.g. the Blowfish and the Twofish encryption algorithms [20].

Rounds: Rounds is the number of iterations in a cipher system. From figure 3 we can have a clear view that each repeated operations stand for a round. According to the crypto analysts, the bigger the number of rounds, the more secure the algorithms will be. The downside is that the execution time of the algorithms increases enormously.

Key Size: key size or key length is the size of the key used in a given cryptographic algorithm. An algorithm's key length is distinct from its cryptographic security, which is a logarithmic measure of the fastest known computational attack on the algorithm, also measured in bits[21].

Security Margin: In order to measure the advantage and disadvantage of different cryptography algorithms, security scientists have created several solutions to define the security level. According to Lenstra and Verheul [22], a cryptosystem can be assumed to be secure only if it is considered to be sufficiently infeasible to mount a successful attack against it. Unfortunately, it is hard to quantify what precisely is meant by "sufficiently infeasible". To cope with this known issue, we adopt the widely used security margin metric, proposed also by Lenstra and Verheul, which is defined as the year until which a user was willing to trust the DES cipher. According to this definition, if an attacker could afford C_{DES} computations in 1982, sufficient to break DES, and can afford C_X computations in year y (y > 1982), sufficient to break cipher *X*, then the security of cipher *X* in year *y* is computationally equivalent to the security of DES in 1982, or in other words, the security margin of cipher *X* is *y*. Since DES was standardized in 1977 and set for review in 1982, the year 1982 is used as the baseline. If the best known attack against a cipher with key length *k* is exhaustive key search, *y* can be calculated according to:

$$y = 1982 + 30/23 * (k - 56)$$ [23]

Key schedule and subkey: A key schedule is an algorithm that, given the key, calculates the subkeys for all rounds. A subkey is a part of cipher key which is caculated with round parameter.

Linear cryptanalysis: A linear cryptanalysis is to find approximations to the action of a cipher. Attacks have been developed for block ciphers and stream ciphers. Linear cryptanalysis is one of the two most widely used attacks on block ciphers; the other is differential cryptanalysis.

Differential cryptanalysis: Differential cryptanalysis is a general form of cryptanalysis applicable primarily to block ciphers, but also to stream ciphers. In the broadest sense, it is the study of how differences in an input can affect the resultant difference at the output. In the case of a block cipher, it refers to a set of techniques for tracing differences through the network of transformations, discovering where the cipher exhibits non-random behaviour, and exploiting such properties to recover the secret key.

Brute-force attack: A Brute-force attack is a method of defeating a cryptographic scheme by trying a large number of possibilities; for example, possible keys in order to decrypt a message. In most schemes, the theoretical possibility of a brute force attack is recognized, but it is set up in such a way that it would be computationally infeasible to carry it out. Accordingly, one definition of "breaking" a cryptographic scheme is to find a method faster than the brute force attack.

Side-channel attack: a side channel attack is any attack based on additional information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms (compare cryptanalysis). For example, timing information, power consumption, electromagnetic leaks or even sound can provide an extra source of information which can be exploited to break the system. Many side-channel attacks require considerable technical knowledge of the internal operation of the system on which the cryptography is implemented.

2.3 Analysis of symmetric-encryption Algorithms

In this section several symmetric encryption algorithms will be analyzed. The history of the algorithms, their key sizes, block sizes, code sizes[1], number of rounds of algorithms and security levels will be discussed. We focus on block ciphers since they are much easier implemented in software. Implementing in software will avoid time-consuming bit manipulations as long as they operate on data in computer-sized blocks.

[1] The code size is the size of the binary program when the C-source code is compiled with the GNU ARM cross-gcc-2.7.2.3. The reasons for using a cross-platform compiler will become clear in the next chapter. The code size is largely dependent on the algorithm implementation and should, thus, be considerated.

2.3.1 DES

History of algorithm: The Data Encryption Standard (DES) is a cipher (a method for encrypting information) selected as an official Federal Information Processing Standard (FIPS) for the United States in 1976 and which has been used internationally. The algorithm was initially controversial, with classified design elements, a relatively short key length, and suspicions about a National Security Agency (NSA) backdoor. DES consequently came under intense academic scrutiny, and motivated the modern understanding of block ciphers and their cryptanalysis [24].

Key size: The key originally consists of 64 bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded. Hence the effective length is 56 bits, and it is usually used as such.

Block size: The block size is 64 bits.

Code size[1]**:** The size of DES is 14.6 KB.

Rounds: There are 16 rounds in the DES algorithms.

Security level: The security margin of DES is 1982.

Known attacks/threats: There are three attacks known that can break the full sixteen rounds of DES: differential cryptanalysis (DC), linear cryptanalysis (LC), and Davies' attack. These types of attack are sometimes termed certification weaknesses. Therefore, we can use brute force attack to attack DES algorithm, which means trying every possible key in turn.

Flow chart: Figure 2.4 depicts DES.

1. DES has 7 stages;
2. Stages 4 to 6 are repeated 16 times;
3. The DES is XOR arithmetic based algorithm.

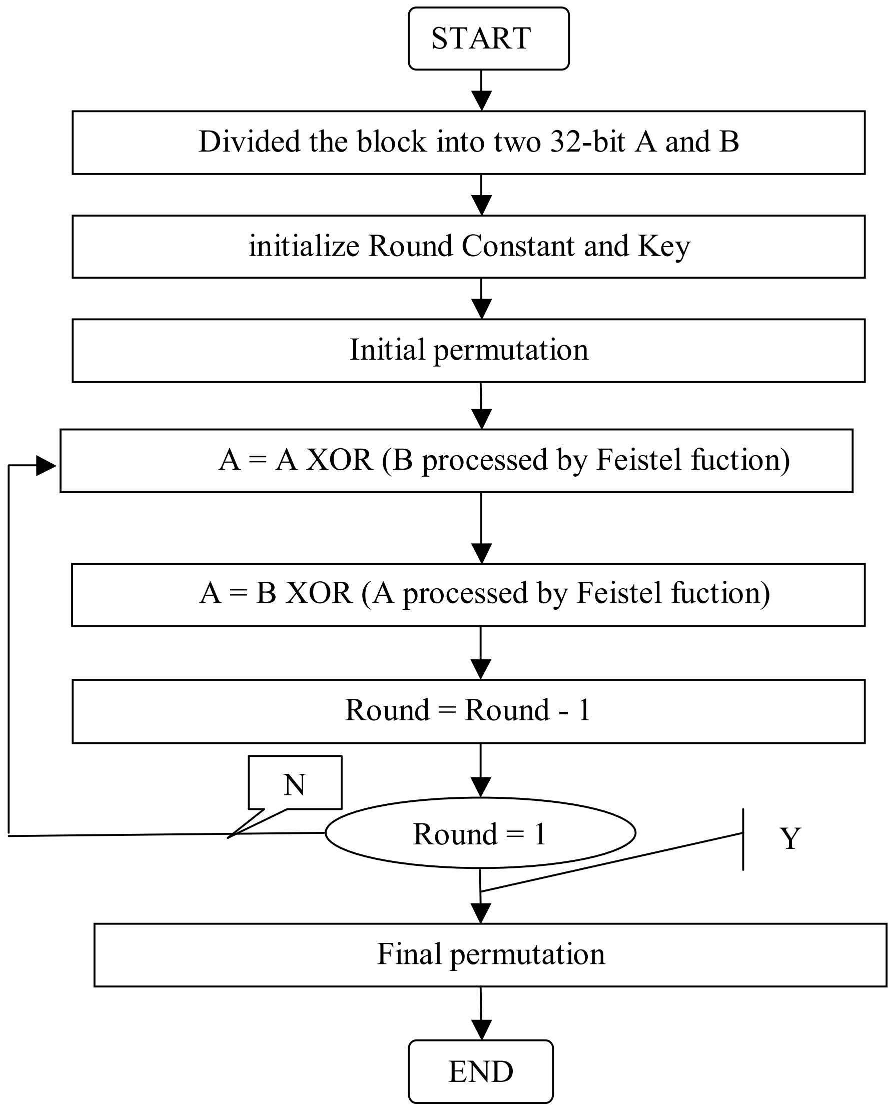
START
Divided the block into two 32-bit A and B
initialize Round Constant and Key
Initial permutation
A = A XOR (B processed by Feistel fuction)
A = B XOR (A processed by Feistel fuction)
Round = Round - 1
N
Round = 1
Y
Final permutation
END

Figure 2.4: Flow Chart of DES

Comments: It is a known fact that DES is an old-fashioned version of symmetric algorithm. It has been cryptanalyzed for many years. Therefore, the security level of DES is quite low. Triple DES is the successor of DES which has a good security level. However, the time penalty of implementing triple DES in resource-constrained platforms is high; thus we prefer implementing DES instead of triple DES in the general case.

2.3.2 BLOWFISH

History of algorithm: Blowfish has been designed in 1993 by Bruce Schneier and included in a large number of cipher suites and encryption products [25].

Key size: Blowfish features a variable key length from 0 up to 448 bits. We have selected 128 bits in our implementation experiment.

Block size: Blowfish has a 64-bit block size.

Code size: The size of the Blowfish code is 15.3 KB.

Rounds: It is a 16-round Feistel cipher and uses large key-dependent S-boxes.

Security level: The security margin of BLOWFISH is 2076.

Known attacks/threats: Blowfish provides a good encryption rate in software and no effective cryptanalysis of it has been found to date.

Flow chart: Figure 2.5 depicts Blowfish.

1. The Blowfish has 7 stages;
2. From stage 2 to stage 6 are repeated 16 times;
3. The DES is XOR arithmetic based algorithm. In the fourth step, the f-function use the first byte of the 32 bits of input to find an entry in the first S-box, the second byte to find an entry in the second S-box, and so on. The value of the f-function is ((S1(B1) + S2(B2)) XOR S3(B3)) + S4(B4) where addition is performed modulo 2^32.

Comments of Blowfish: Blowfish is a successor of DES. It uses amounts of the principles used in DES to provide the same security with greater speed and efficiency in software [26]. Blowfish uses round keys; the entire contents of all the S-boxes are created by multiple iterations of the block cipher. This enhances the security of the block cipher, since it makes exhaustive search of the key space very difficult, even for short keys [27]. It could be the most secure symmetric algorithm.

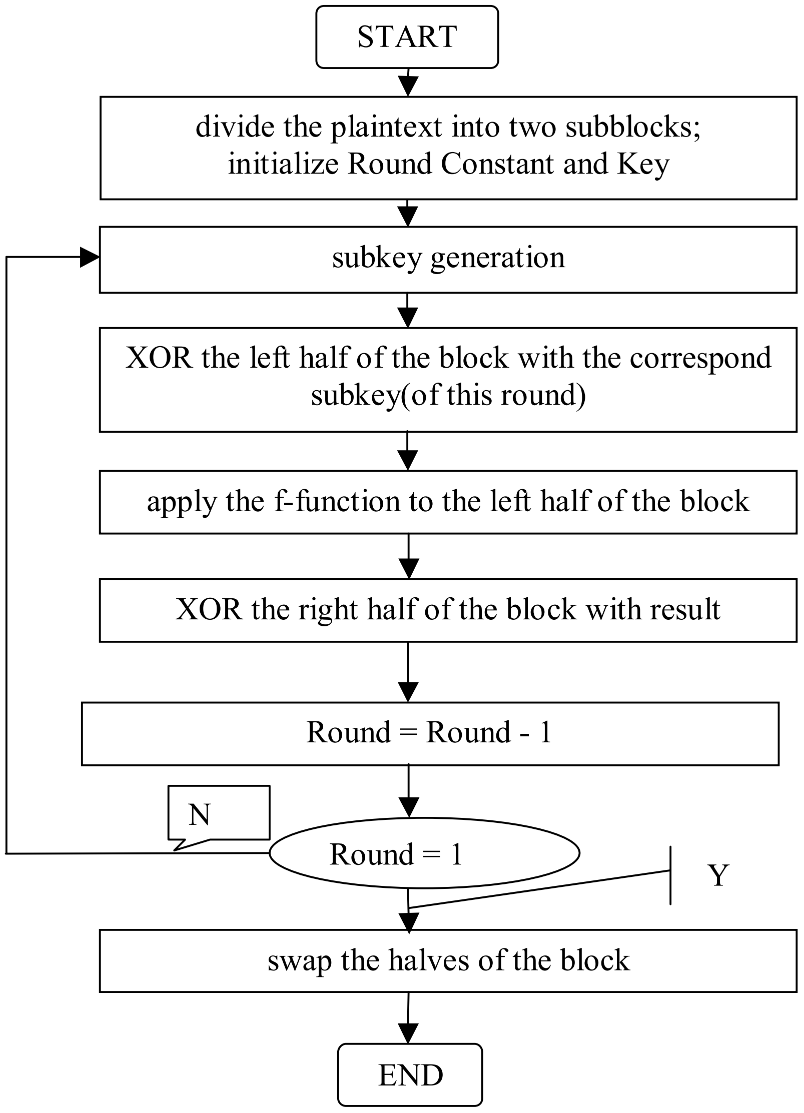
START
divide the plaintext into two subblocks; initialize Round Constant and Key
subkey generation
XOR the left half of the block with the correspond subkey(of this round)
apply the f-function to the left half of the block
XOR the right half of the block with result
Round = Round - 1
N
Round = 1
Y
swap the halves of the block
END

Figure 2.5: Flow Chart of Blowfish

2.3.3 3-WAY

History of algorithm: 3-Way is a block cipher designed in 1994 by Joan Daemen. It is designed to be very efficient in a wide range of platforms from 8-bit processors to specialized hardware, and has some elegant mathematical features which enable nearly all the decryption to be done in exactly the same circuits as did the encryption.

Key size: The key length is also 96 bits. The figure 96 arises from the use of three 32 bit words in the algorithm, from which also is derived the cipher's name.

Block size: The block size of 3-WAY is 96 bits.

Code size: The size of the 3-WAY code is 11.2 KB.

Rounds: 3-Way is an 11-round substitution-permutation network.

Security level: The security margin of 3-WAY is 2034.

Known attacks/threats: 3-Way is vulnerable to related-key cryptanalysis; Jone Kelsy, Bruce Schneier, and David Wagner show how it can be broken with one related key query and about 2^22 chosen plaintexts [28].

Flow Chart: Figure 2.6 depicts DES.

1. The DES has 7 stages;
2. Stages 2 to 7 are repeated 11 times;
3. In the first step, the plaintext is divided into three subblocks; each of them is the same size as 32bit. That is why we call this algorithm 3WAY;
4. In the third step, matrix multiplication is carried out in parallel eight times, for each of the eight bits in the block contains. The matrix multiplication has a structure that allows it to be implemented in terms of shifts and XORs of 32-bit words which is the same size as subblock size. This use of matrix multiplication is similar to Rijndael (AES);
5. In the fourth step, the first 32-bit subblock is rotated 10 bits to the right, and the third 32-bit subblock is rotated 1 bit to the left. The second subblock is not modified;
6. In the sixth step, the diffusion makes the first 32-bit subblock rotated 1 bit to the left, and the third 32-bit subblock rotated 10 bits to the right.

Comments: 3WAY algorithm is normally implemented by hardware in the industrial world.

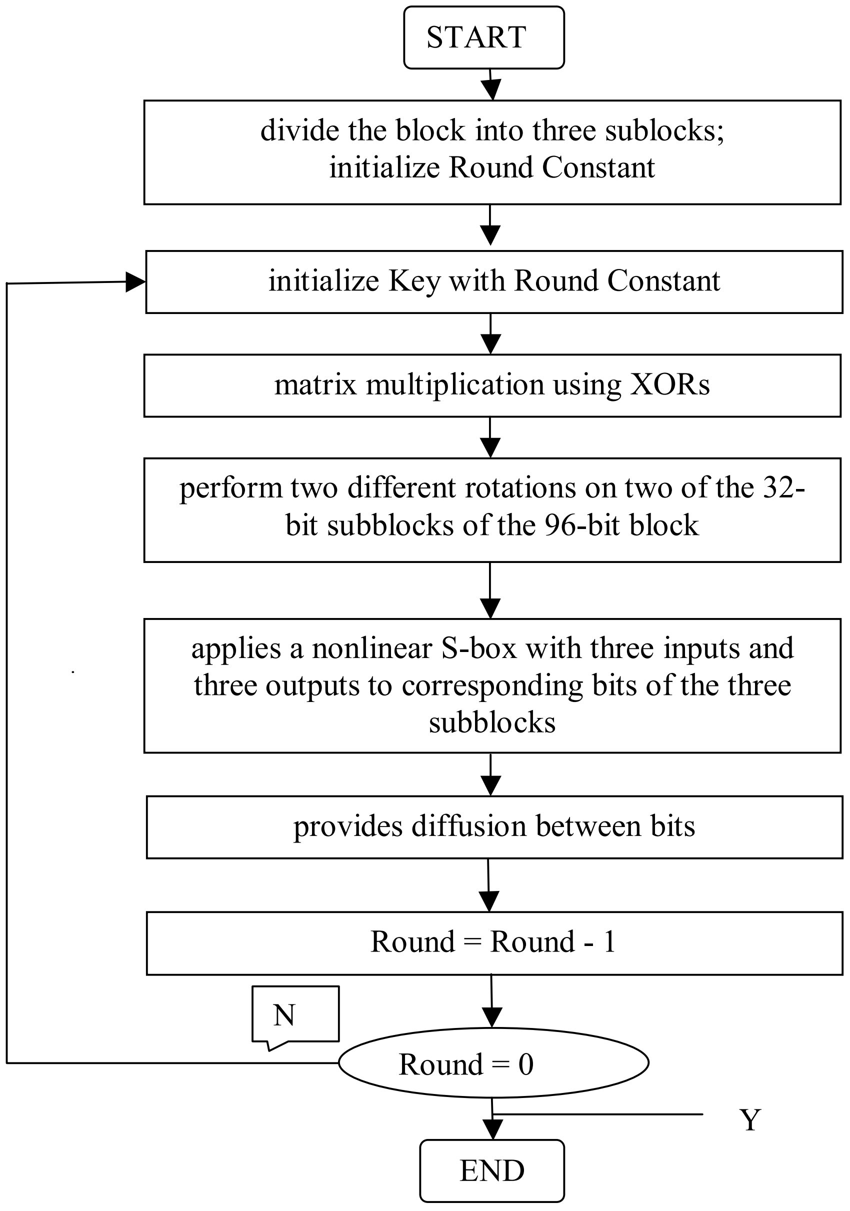
START
divide the block into three sublocks; initialize Round Constant
initialize Key with Round Constant
matrix multiplication using XORs
perform two different rotations on two of the 32-bit subblocks of the 96-bit block
applies a nonlinear S-box with three inputs and three outputs to corresponding bits of the three subblocks
provides diffusion between bits
Round = Round - 1
N
Round = 0
Y
END

Figure 2.6: Flow Chart of 3-WAY

2.3.4 GOST

History of algorithm: GOST standards were originally developed by the government of the Soviet Union as part of its national standardization strategy. The word GOST in Russian is an acronym for “gosudarstvennyy standart”, which means state standard. The first GOST standard, GOST 1 State Standardization System, was published in 1968 [29].

Key size: GOST has a key of 256 bits.

Block size: GOST has a 64-bit block size.

Code size: The code size of the GOST is 12.2KB.

Rounds: GOST is a Feistel network of 32 rounds.

Security level: The security margin of GOST is 2243.

Known attacks/threats: Not available so far.

Flow chart: Figure 2.8 depicts GOST.

1. The GOST has 6 stages;

2. Stages 2 to 6 are repeated 32 times;

3. The GOST is S-box based algorithm. In the third step, S-boxes accept a four-bit input and produce a four-bit output. The S-box substitution in the round function consists of eight 4 × 4 S-boxes.

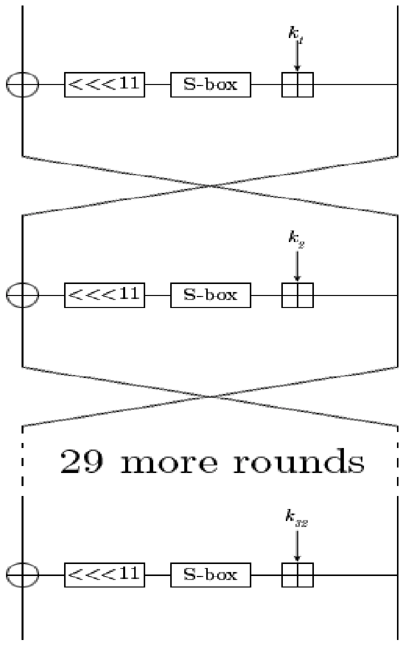

Figure 2.7: Diagram of GOST

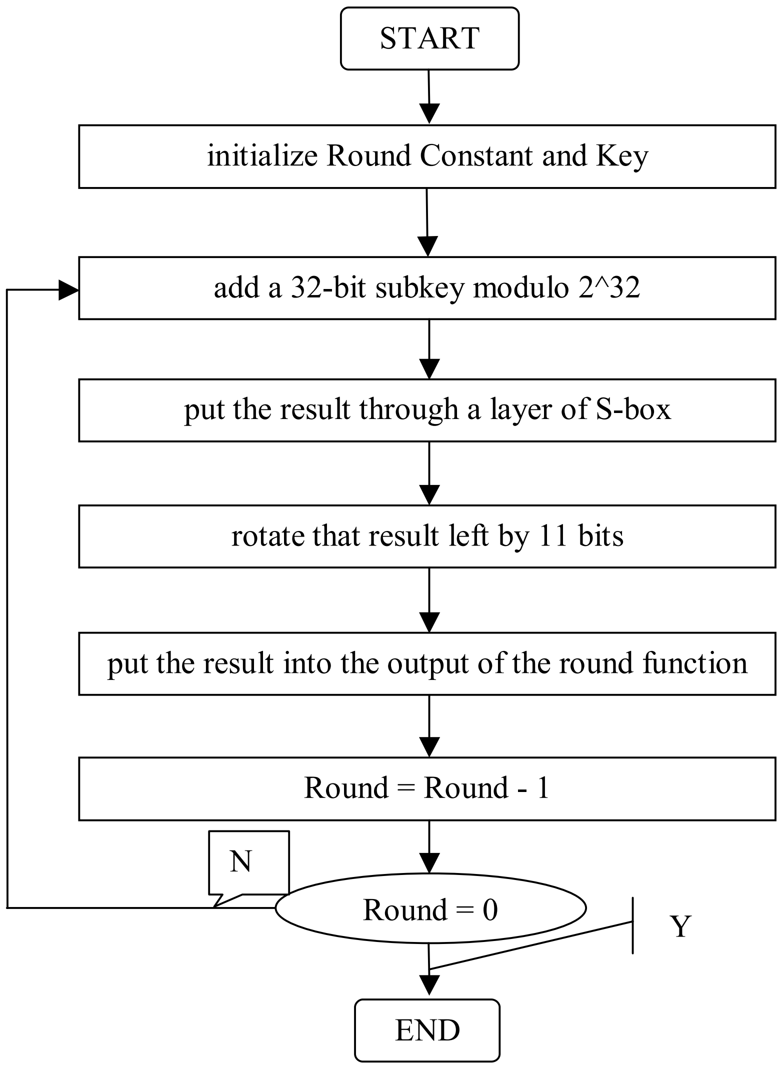
START
initialize Round Constant and Key
add a 32-bit subkey modulo 2^32
put the result through a layer of S-box
rotate that result left by 11 bits
put the result into the output of the round function
Round = Round - 1
N
Round = 0
Y
END

Figure 2.8: Flow Chart of GOST

Comments of GOST: GOST has a very simple round function comparing with DES. It can be predicted that the time consumption of GOST is smaller than DES.

2.3.5 IDEA

History of algorithm: The International Data Encryption Algorithm (IDEA) is designed by Xuejia Kai and James Massey of ETH Zurich and was first described in 1991. The algorithm was intended as a replacement for the DES. IDEA is a minor revision of an earlier cipher, PES (Proposed Encryption Standard); IDEA was originally called IPES (Improved PES). The cipher was designed under a research contract with the Hasler

Foundation, which became part of Ascom-Tech AG. The cipher is patented in a number of countries but is freely available for non-commercial use [30].

Key size: IDEA uses 52 subkeys; each of them is 16 bits long. Two are used during each round and four are used before every round and after the last round.

Block size: The plaintext block in IDEA is 64 bits long. It is divided into four quarters, each 16 bits long. Three operations are used in IDEA to combine two 16 bit values to produce a 16 bit result, addition, XOR, and multiplication.

Code size: The code size of IDEA is 13.4KB.

Rounds: There are 8 rounds in the IDEA algorithms.

Security level: The security margin of IDEA is 2243.

Known attack/threat: Not available so far.

Flow chart: Figure 2.9 depicts IDEA.

1. The IDEA has 10 stages;
2. Stages 3 to 10 are repeated 8 times;
3. The plaintext block in IDEA is divided into four quarters, each 16 bits long. Three operations are used in IDEA to combine two 16 bit values to produce a 16 bit result, addition, XOR, and multiplication.

Comments of IDEA: IDEA derives much of its security by interleaving operations from different groups — modular addition and multiplication, and bitwise exclusive OR (XOR) — which are algebraically "incompatible" in some sense.

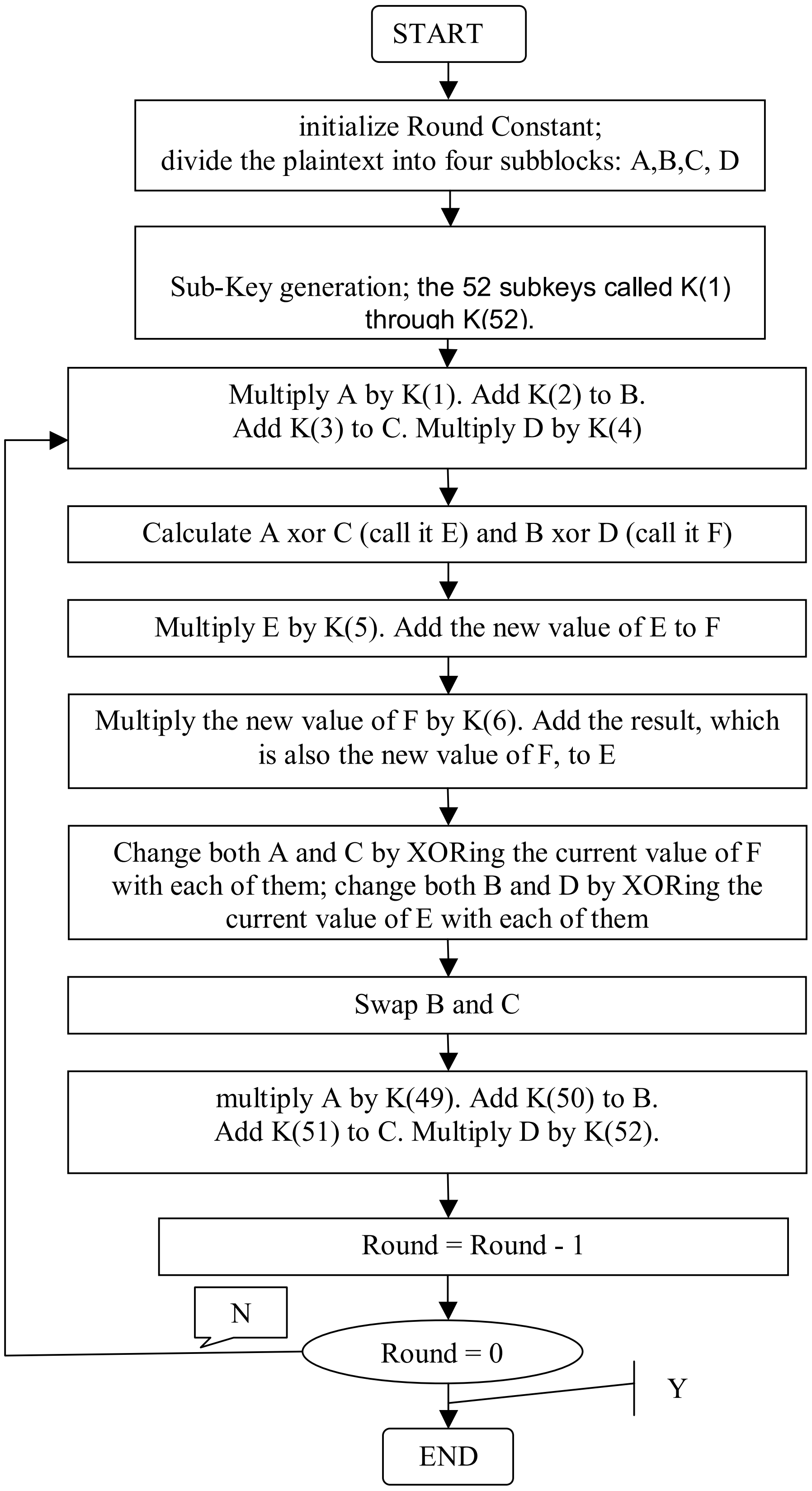
START
initialize Round Constant; divide the plaintext into four subblocks: A,B,C, D
Sub-Key generation; the 52 subkeys called K(1) through K(52).
Multiply A by K(1). Add K(2) to B. Add K(3) to C. Multiply D by K(4)
Calculate A xor C (call it E) and B xor D (call it F)
Multiply E by K(5). Add the new value of E to F
Multiply the new value of F by K(6). Add the result, which is also the new value of F, to E
Change both A and C by XORing the current value of F with each of them; change both B and D by XORing the current value of E with each of them
Swap B and C
multiply A by K(49). Add K(50) to B. Add K(51) to C. Multiply D by K(52).
Round = Round - 1
N
Round = 0
Y
END

Figure 2.9: Flow Chart of IDEA

2.3.6 LOKI91

History of algorithm: LOKI91 has also been designed as a possible replacement for DES. The cipher was developed based on a body of work analyzing DES and is very similar to DES in structure. LOKI91 was designed in response to the attacks on LOKI89 which is the ancestor of LOKI91. The changes included removing the initial and final key, new S-box, and small alterations to the key schedule [31].

Key size: The key size of the LOKI91 is 64 bit.

Block size: The block size of the LOKI91 is 64 bit.

Code size: The code size of the LOKI91 is 11.3KB.

Round of algorithm: There are 16 rounds in the LOKI91 algorithms.

Security level: The security margin of LOKI91 is 1992.

Known attack/threat: The S-boxes functions of LOKI91 is changed to minimize the probability of seeing different inputs resulting in the same output (a kind of differential cryptanalysis attack), thus improving LOKI91's immunity to this attack. The changes to the key schedule were designed to reduce the number of "equivalent" or "related" keys, which resulted in the exhaustive search space (a kind of brute force attack) for the cipher being reduced.

Flow chart: Figure 2.10 depicts LOKI91.

1. The LOKI91 has 9 stages;
2. Stages 3 to 9 are repeated 16 times;
3. The LOKI91 is S-box and P-box based algorithms.

Comments: LOKI91 analyses key schedule to minimize generation of equivalent key, or related (key, plaintext) pairs which greatly reduce the time consumption of key generation.

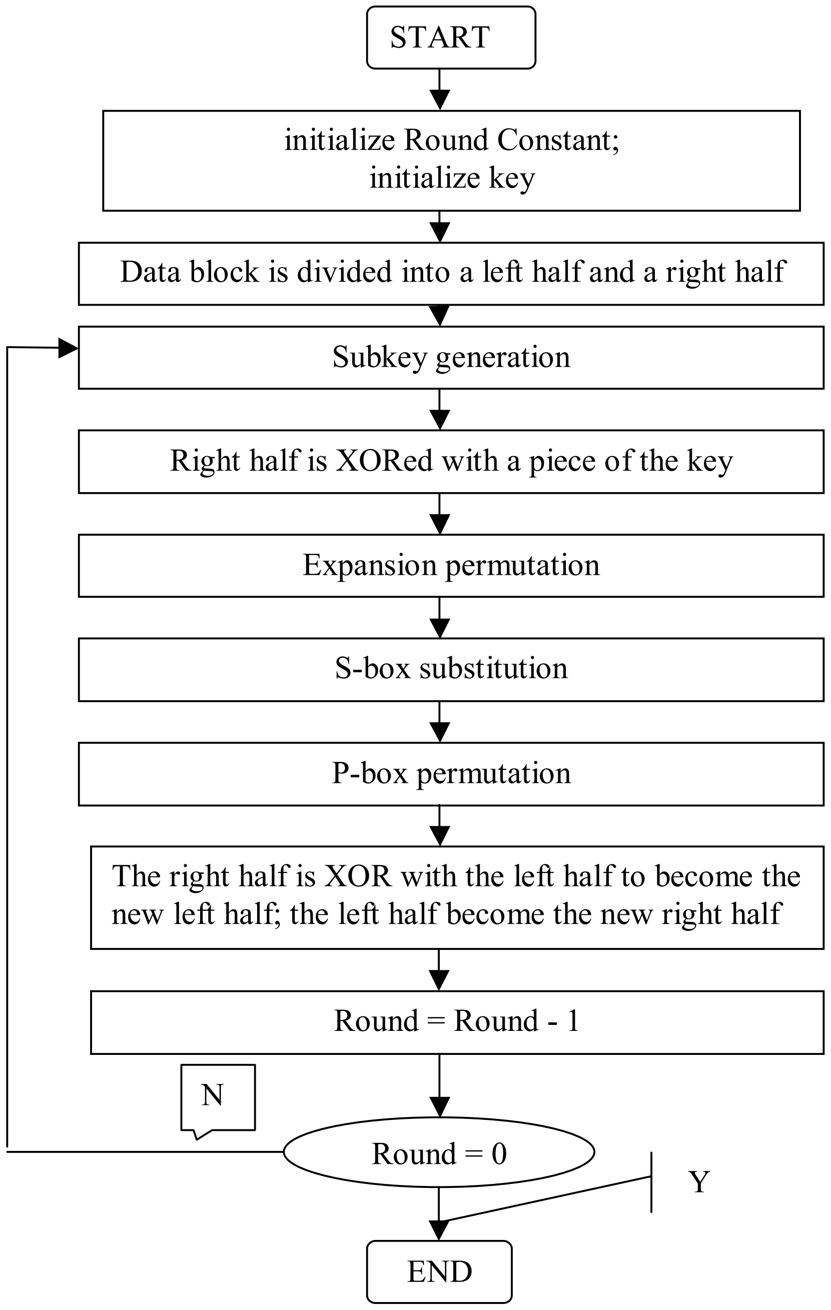
START
initialize Round Constant;
initialize key
Data block is divided into a left half and a right half
Subkey generation
Right half is XORed with a piece of the key
Expansion permutation
S-box substitution
P-box permutation
The right half is XOR with the left half to become the new left half; the left half become the new right half
Round = Round - 1
N
Round = 0
Y
END

Figure 2.10: Flow Chart of LOKI91

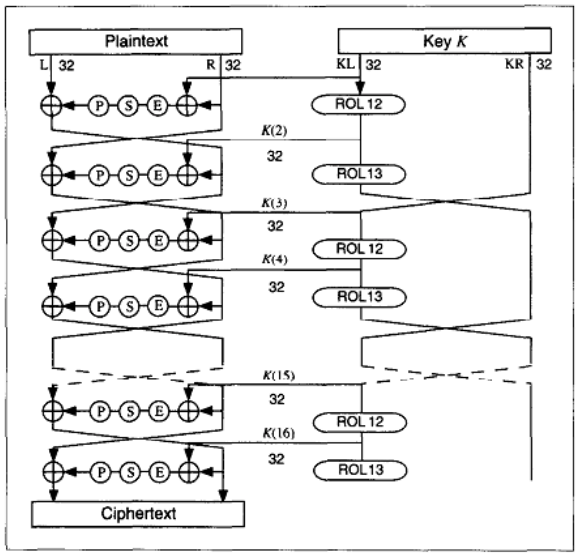

Figure 2.11: Diagram of LOKI91

2.3.7 RC5

History of algorithm: RC5 is a simple block cipher. It has been designed by Ronald Rivest in 1994 [32]. RC stands for "Rivest Cipher", or alternatively, "Ron's Code".

Key size:	The key size of RC5 is 128 bits.
Block size:	The block size of RC5 is 64 bits.
Code size:	The size of RC5 code is 11.4 KB.

Round of algorithm: There are 12 rounds in the RC5 algorithm.

Security level of algorithm: The security margin of RC5 is 2076.

Attack or hacking of algorithm: Until now, 12-round RC5 (with 64-bit blocks) is susceptible to a differential attack using 2^44 chosen plaintexts [33]. 18–20 rounds are suggested as sufficient protection.

Flow chart: Figure 2.12 depicts RC5.

1. The RC5 has 5 stages;
2. Stages 4 to 5 are repeated 12 times;

3. The RC5 is XOR arithmetic algorithm based algorithms;

4. In the fourth step, we use 2r + 2 key-dependent 32-bit words: S0, S1, S2….S2r+1, where r is the number of rounds.

Comments of RC5: RC5 is the ancestor of RC6. Both of them can be well implemented by software in embedded system.

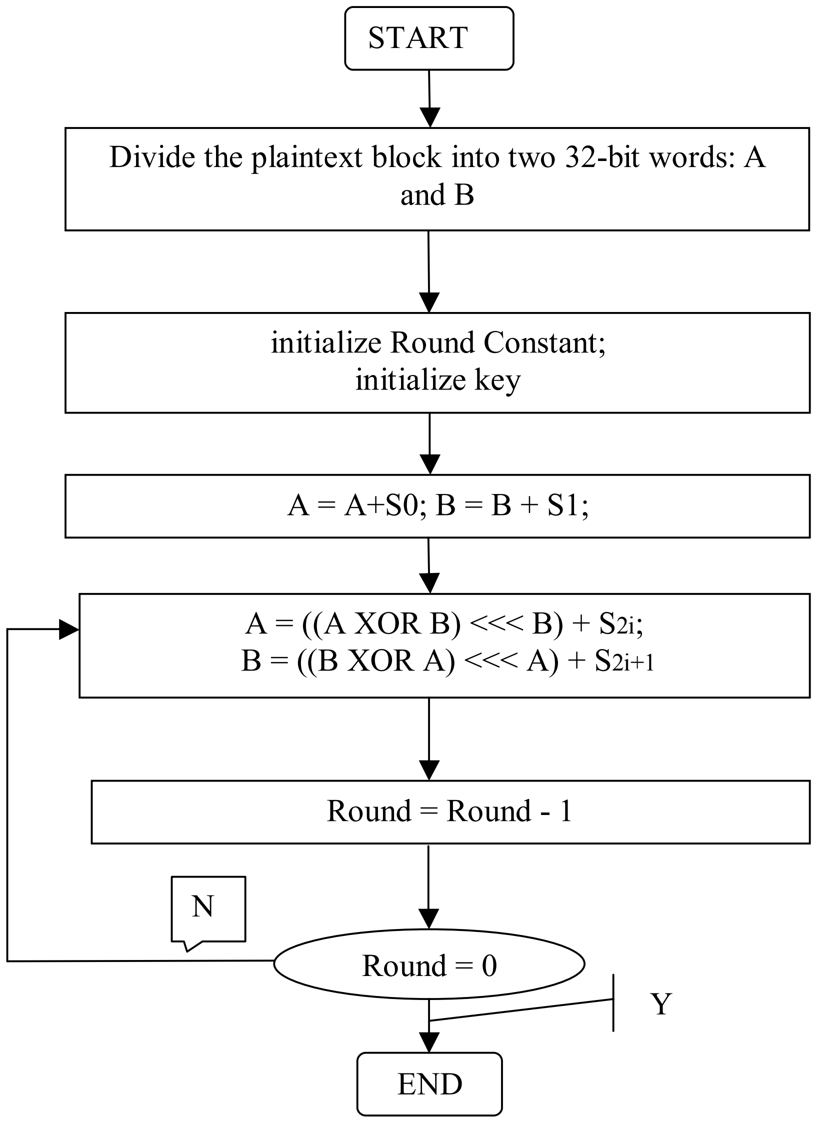
START
Divide the plaintext block into two 32-bit words: A and B
initialize Round Constant;
initialize key
A = A+S0; B = B + S1;
A = ((A XOR B) <<< B) + S2i;
B = ((B XOR A) <<< A) + S2i+1
Round = Round - 1
N
Round = 0
Y
END

Figure 2.12: Flow Chart of RC5

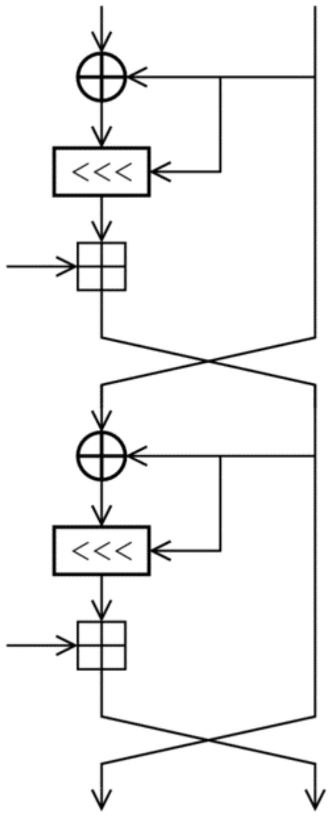

Figure 2.13: Diagram of RC5

2.3.8 RC6

History of algorithm: RC6 is derived from RC5. It was designed by Ron Rivest, Matt Robshaw, Ray Sidney, and Yiqun Lisa Yin to meet the requirements of the Advanced Encryption Standard (AES) competition. The algorithm was one of the five finalists, and was also submitted to the NESSIE and CRYPTREC projects. It is a proprietary algorithm, patented by RSA Security [34].

Key size: The key size of RC6 is 128 bits.

Block size: The block size of RC6 is 128 bits.

Code size: The size of RC6 code is 11.4 KB.

Rounds: There are 20 rounds in the RC6 algorithms.

Security level of algorithm: The security margin of RC6 is 2076.

Known attack/threat: Not available so far.

Flow chart: Figure 2.14 depicts RC6.

1. The RC6 has 8 stages;
2. Stages 4 to 8 are repeated 20 times;
3. In the third step, the first and the second subkey S0 and S1 are used. Then each round of RC6 uses two subkeys; the first one uses S2 and S3, and successive rounds use successive subkeys.

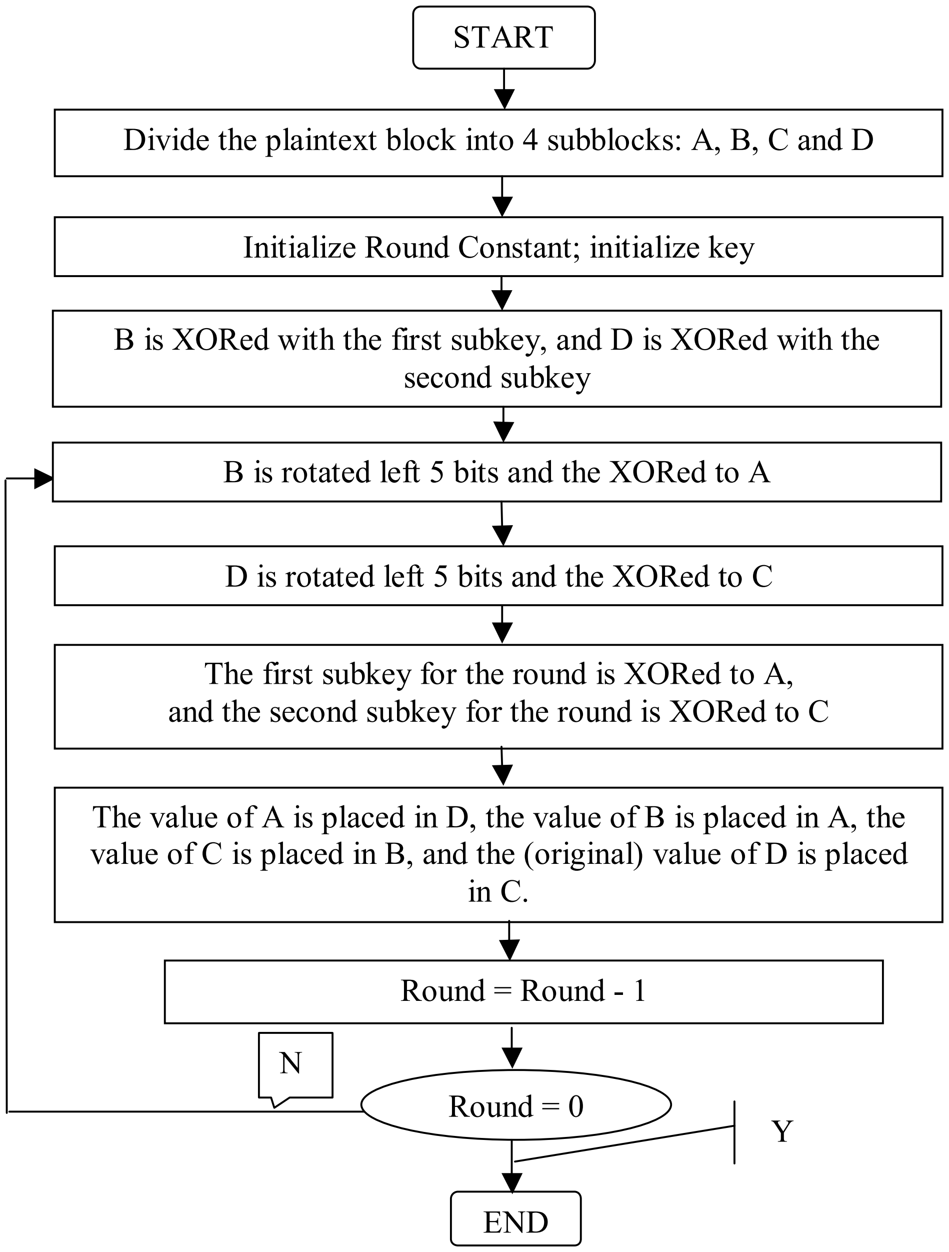
START
Divide the plaintext block into 4 subblocks: A, B, C and D
Initialize Round Constant; initialize key
B is XORed with the first subkey, and D is XORed with the second subkey
B is rotated left 5 bits and the XORed to A
D is rotated left 5 bits and the XORed to C
The first subkey for the round is XORed to A, and the second subkey for the round is XORed to C
The value of A is placed in D, the value of B is placed in A, the value of C is placed in B, and the (original) value of D is placed in C.
Round = Round - 1
N
Round = 0
Y
END

Figure 2.14: Flow Chart of RC6

Comments of RC6: RC6 can be viewed as interweaving two parallel RC5 encryption processes. However, RC6 does use an extra multiplication operation not present in RC5 in order to make the rotation dependent on every bit in a word, and not just the least significant few bits.

2.3.9 SKIPJACK

History of algorithm: SKIPJACK is representative of a family of encryption algorithms developed in 1980 as part of the NSA suite of "TYPE I" algorithms. SKIPJACK was designed using building blocks and techniques that have forty years' history. Many of the techniques are related to work that was evaluated by some of the world's most famous experts in combinatory and abstract algebra. SKIPJACK's more immediate heritage dates to around 1980, and its initial design to 1987. The specific structures included in SKIPJACK have a long evaluation history, and the cryptographic properties of those structures had many prior years of intense study before the formal process began in 1987 [35].

Key size: The key size of SKIPJACK is 80 bits.

Block size: The block size of SKIPJACK is 64 bits.

Code size: The size of SKIPJACK code is 12.2 KB.

Rounds: There are 32 rounds in the SKIPJACK algorithms.

Security level: The security margin of SKIPJACK is 2013.

Known attack/threat: Eli Biham and Adi Shamir discovered an attack against 16 of the 32 rounds within one day of declassification, and extended this to 31 of the 32 rounds within months using impossible differential cryptanalysis.

Flow chart: Figure 2.15 depicts SKIPJACK.

1. The SKIPJACK has 6 stages;
2. Stages 3 to 6 are repeated 32 times;

3. In the fourth step, permutation G is a four-round Feistel cipher, involving dividing its 16-bit input into two 8-bit halves. Like DES, the left half of the block is not changed in each round, but acts as input to the f-function, the output of which is XORed to the right half. Unlike DES, the two halves are swapped after the last round.

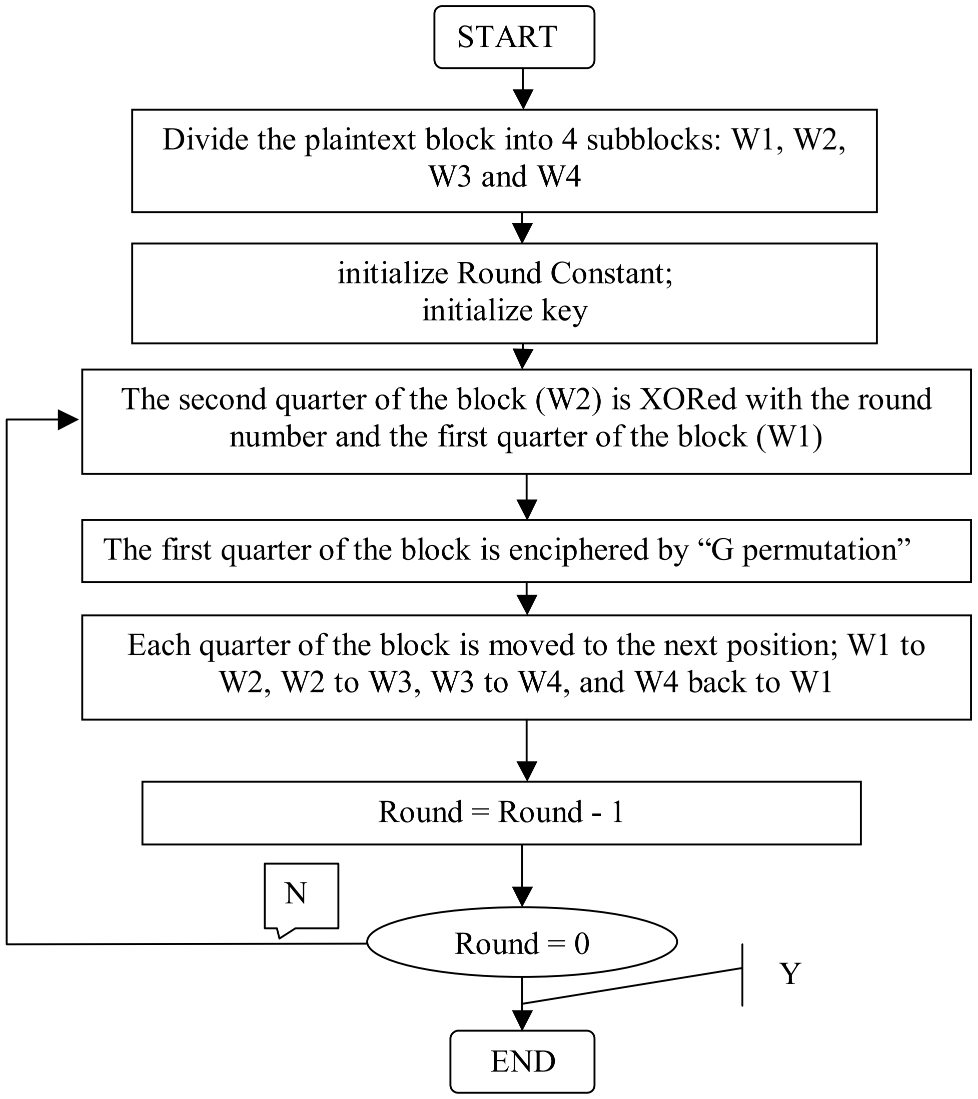
START
Divide the plaintext block into 4 subblocks: W1, W2, W3 and W4
initialize Round Constant; initialize key
The second quarter of the block (W2) is XORed with the round number and the first quarter of the block (W1)
The first quarter of the block is enciphered by “G permutation”
Each quarter of the block is moved to the next position; W1 to W2, W2 to W3, W3 to W4, and W4 back to W1
Round = Round - 1
N
Round = 0
Y
END

Figure 2.15: Flow Chart of SKIPJACK

Comments of SKIPJACK: SKIPJACK is designed to substitute DES and it is widely implemented by hardware.

2.3.10 XXTEA

History of algorithm: Corrected Block TEA (often referred to as XXTEA) is designed to correct weaknesses in the original Block TEA (Tiny Encryption Algorithm), which was first published together with a paper on TEA extensions (XTEA)[36]. The cipher's designers were Roger Needham and David Wheeler of the Cambridge Computer Laboratory, and the algorithm was presented in an unpublished technical report in October 1998 [37].

Key size: The key size of XXTEA is 128.

Block size: The block size of XXTEA is 64.

Code size: The size of XXTEA code is 11.2 KB.

Rounds: There are 32 rounds in the XXTEA algorithms.

Security level: The security margin of XXTEA is 2076.

Known attack/threat: Not available so far.

Flow chart: Figure 2.16 depicts XXTEA.

1. The XXTEA has 8 stages;
2. Stages 4 to 8 are repeated 32 times;
3. XXTEA is left shifting and right shifting arithmetic based algorithm.

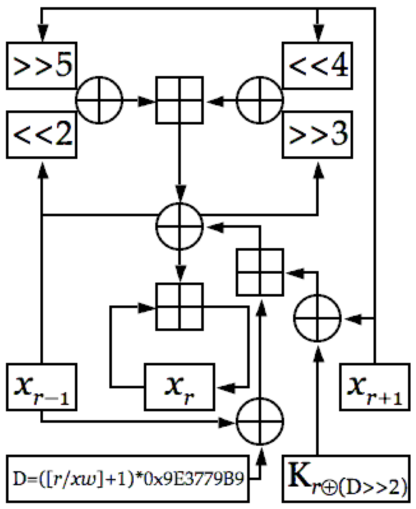

Figure 2.16: Flow Chart of XXTEA

Comments: The size of block for XXTEA can be arbitary. The unusually small size of the XXTEA algorithm would make it a viable option in situations where there are extreme constraints e.g. embedded hardware systems where the amount of available RAM is minimal [38].

2.3.11 MISTY1

History of algorithm: MISTY1 is designed in 1995 by Mitsuru Matsui and others for Mitsubishi Electric. MISTY1 is one of the selected algorithms in the European NESSIE project, and has been recommended for Japanese government use by the CRYPTREC project. "MISTY" can stand for "Mitsubishi Improved Security Technology", it is also the initials of the researchers involved in its development: Matsui Mitsuru, Ichikawa Tetsuya, Sorimachi Toru, Tokita Toshio, and Yamagishi Atsuhiro [39].

Key size: The key size of MISTY1 is 128 bits.

Block size: The block size of MISTY1 is 64 bits.

Code size: The size of MISTY1 code is 18.8 KB.

Rounds: There are 8 rounds in the MISTY1 algorithms.

Security level: The security margin of MISTY1 is 2076.

Known attack/threat: MISTY1 claims to be provably secure against linear and differential cryptanalysis.

Flow chart: Figure 2.17 depicts MISTY1.

1. The MISTY1 has 11 stages;
2. Stages 3 to 11 are repeated 8 times;
3. MYSTY1 is S-box based algorithm. It uses two kinds of function, which are called function FO and function FL. Function FO calls another function, namely FI. The key expansion part also uses function FI. Function FI uses two S-boxes, namely S7, S9.

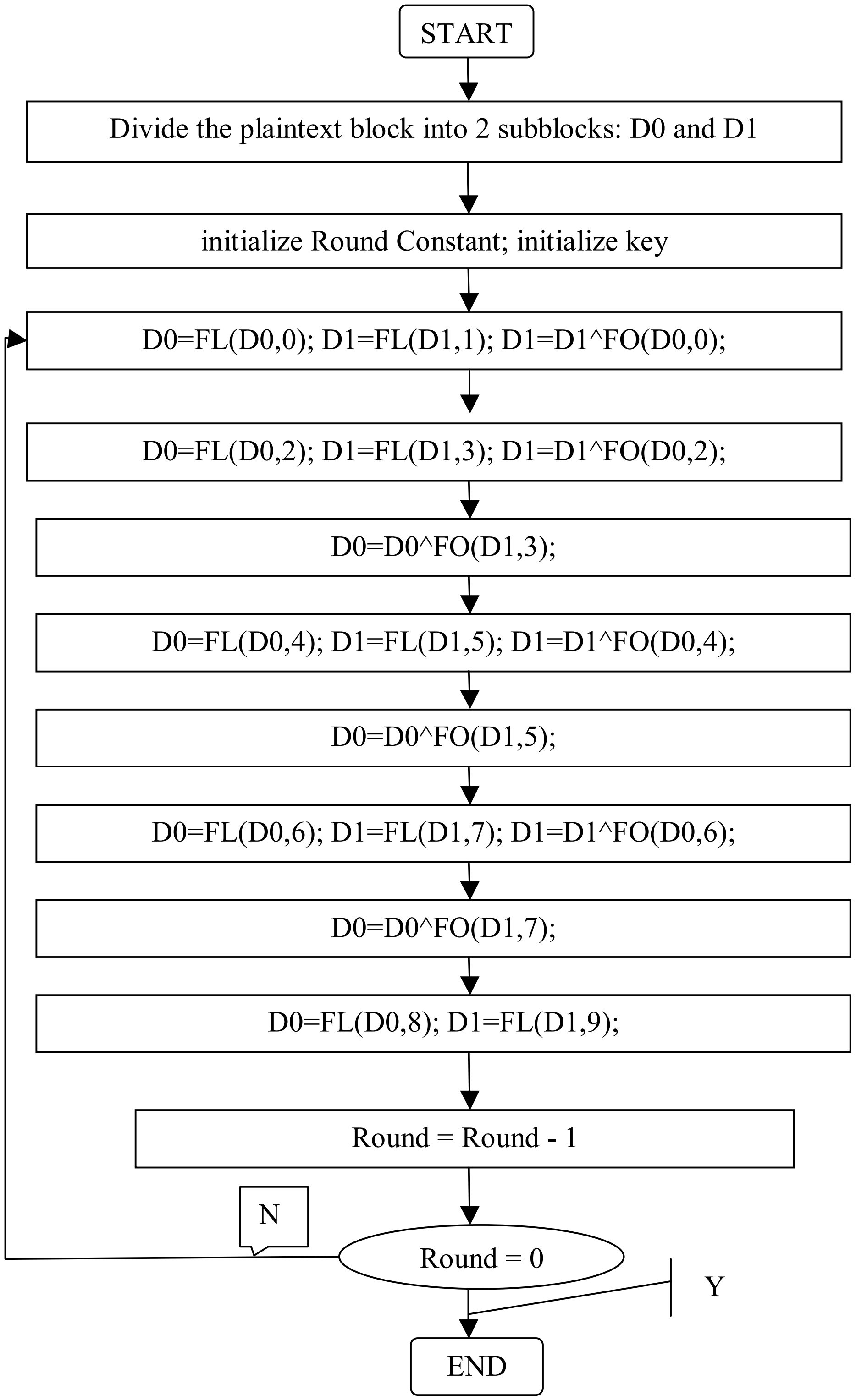
START
Divide the plaintext block into 2 subblocks: D0 and D1
initialize Round Constant; initialize key
D0=FL(D0,0); D1=FL(D1,1); D1=D1^FO(D0,0);
D0=FL(D0,2); D1=FL(D1,3); D1=D1^FO(D0,2);
D0=D0^FO(D1,3);
D0=FL(D0,4); D1=FL(D1,5); D1=D1^FO(D0,4);
D0=D0^FO(D1,5);
D0=FL(D0,6); D1=FL(D1,7); D1=D1^FO(D0,6);
D0=D0^FO(D1,7);
D0=FL(D0,8); D1=FL(D1,9);
Round = Round - 1
N
Round = 0
Y
END

Figure 2.17: Flow Chart of MISTY1

Comments of MISTY1: MISTY1 has an innovative recursive structure; the round function itself uses a 3-round Feistel network. MISTY1 claims to be provably secure against linear and differential cryptanalysis.

2.3.12 TOWFISH

History of algorithm: TWOFISH is a block cipher by Counterpane Labs, published in 1998. It was one of the five Advanced Encryption Standard (AES) finalists [40], but it was not selected for standardization. Twofish is related to the earlier block cipher Blowfish [41].

Key size: The key size of TWOFISH is 128 bits.

Block size: The block size of TWOFISH is 128 bits. It is divided into 4 parts: Q0, Q1, Q2 and Q3.

Code size: The size of TWOFISH code is 22.2 KB.

Rounds: There are 16 rounds in the TWOFISH algorithms.

Security level: The security margin of TWOFISH is 2076.

Attack or hacking of algorithm: Not available so far.

Flow chart: Figure 2.18 depicts TWOFISH.

1. The TWOFISH has 10 stages
2. From stage 3 to stage 10 are repeated 16 times
3. TWOFISH is shifting arithmetic and XOR arithmetic based algorithm

Comments of TWOFISH: On most software platforms TWOFISH is slightly slower than Rijndael (AES) for 128-bit keys, but somewhat faster for 256-bit keys [42]. TWOFISH is related to the earlier block cipher Blowfish. However, TWOFISH has seen less widespread usage than Blowfish, which has been available for a longer period of time.

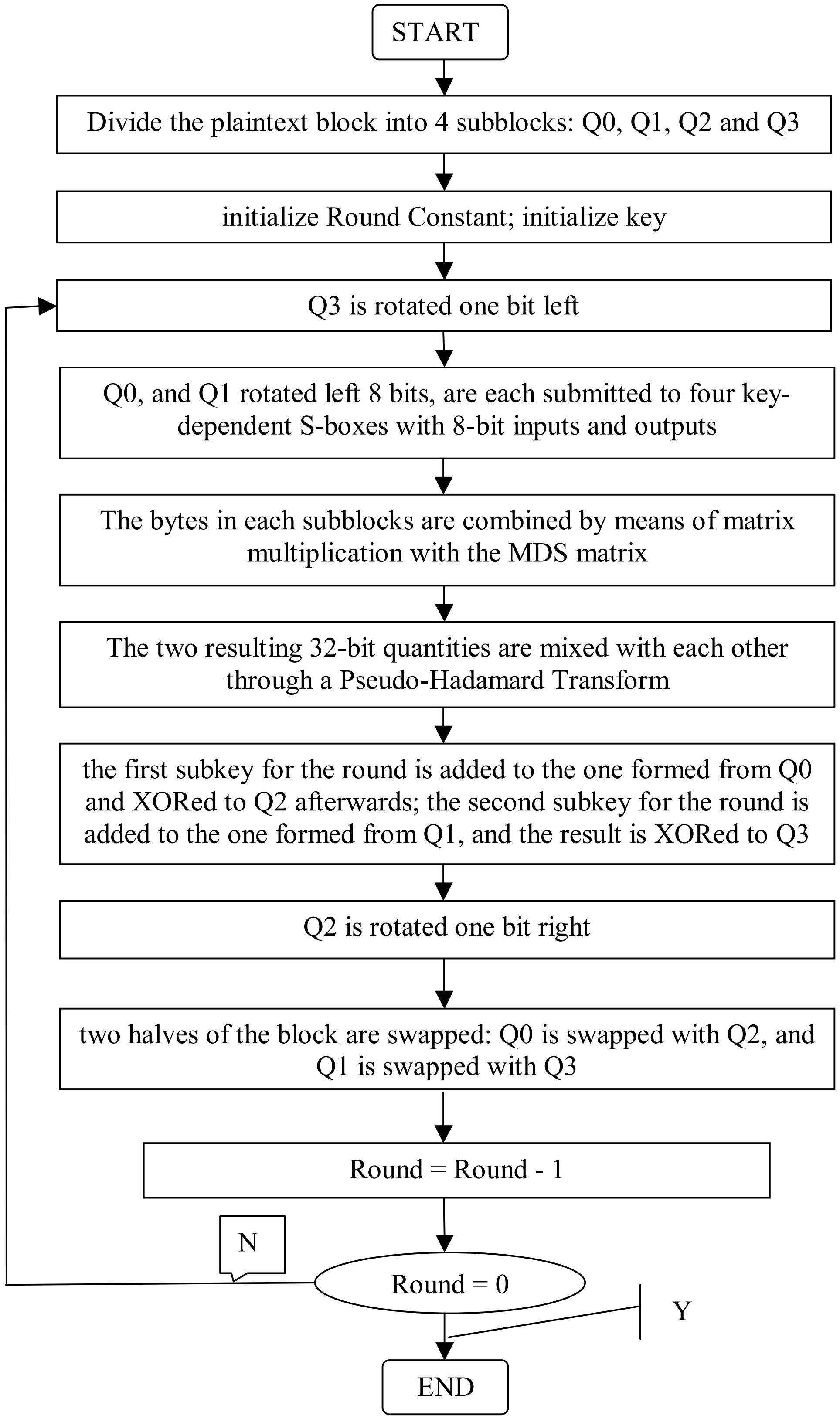
START
Divide the plaintext block into 4 subblocks: Q0, Q1, Q2 and Q3
initialize Round Constant; initialize key
Q3 is rotated one bit left
Q0, and Q1 rotated left 8 bits, are each submitted to four key-dependent S-boxes with 8-bit inputs and outputs
The bytes in each subblocks are combined by means of matrix multiplication with the MDS matrix
The two resulting 32-bit quantities are mixed with each other through a Pseudo-Hadamard Transform
the first subkey for the round is added to the one formed from Q0 and XORed to Q2 afterwards; the second subkey for the round is added to the one formed from Q1, and the result is XORed to Q3
Q2 is rotated one bit right
two halves of the block are swapped: Q0 is swapped with Q2, and Q1 is swapped with Q3
Round = Round - 1
N
Round = 0
Y
END

Figure 2.18: Flow Chart of TWOFISH

2.3.13 RIJNDAEL (AES)

History of algorithm: RIJNDAEL has been adopted as an encryption standard by the U.S. government. It has been analyzed extensively and is now used worldwide, as was the case with its predecessor [43], the Data Encryption Standard (DES). RIJNDAEL was announced by the National Institute of Standards and Technology (NIST) as U.S. FIPS PUB 197 (FIPS 197) on November 26, 2001 after a 5-year standardization process in which fifteen competing designs were presented and evaluated before Rijndael was selected as the most suitable. It became effective as a standard May 26, 2002. As of 2006, RIJNDAEL is one of the most popular algorithms used in symmetric key cryptography.

Key size: The key size of Rijndael is 128 bits.

Block size: The block size of Rijndael is 128 bits.

Code size: The size of Rijndael code is 37 KB.

Rounds: There are 12 rounds in the Rijndael algorithms.

Security level: The security margin of Rijndael is 2076.

Known attack/threat: As of 2006, the only successful attacks against AES implementations have been side-channel attacks. The National Security Agency (NSA) reviewed Rijndael, and stated that it was secure enough for US Government non-classified data [44].

Flow chart: Figure 2.19 depicts Rijndael.

1. The Rijndael has 10 stages;
2. Stages 2 to 7 are repeated 12 times;
3. Rijndael is S-box based algorithms.

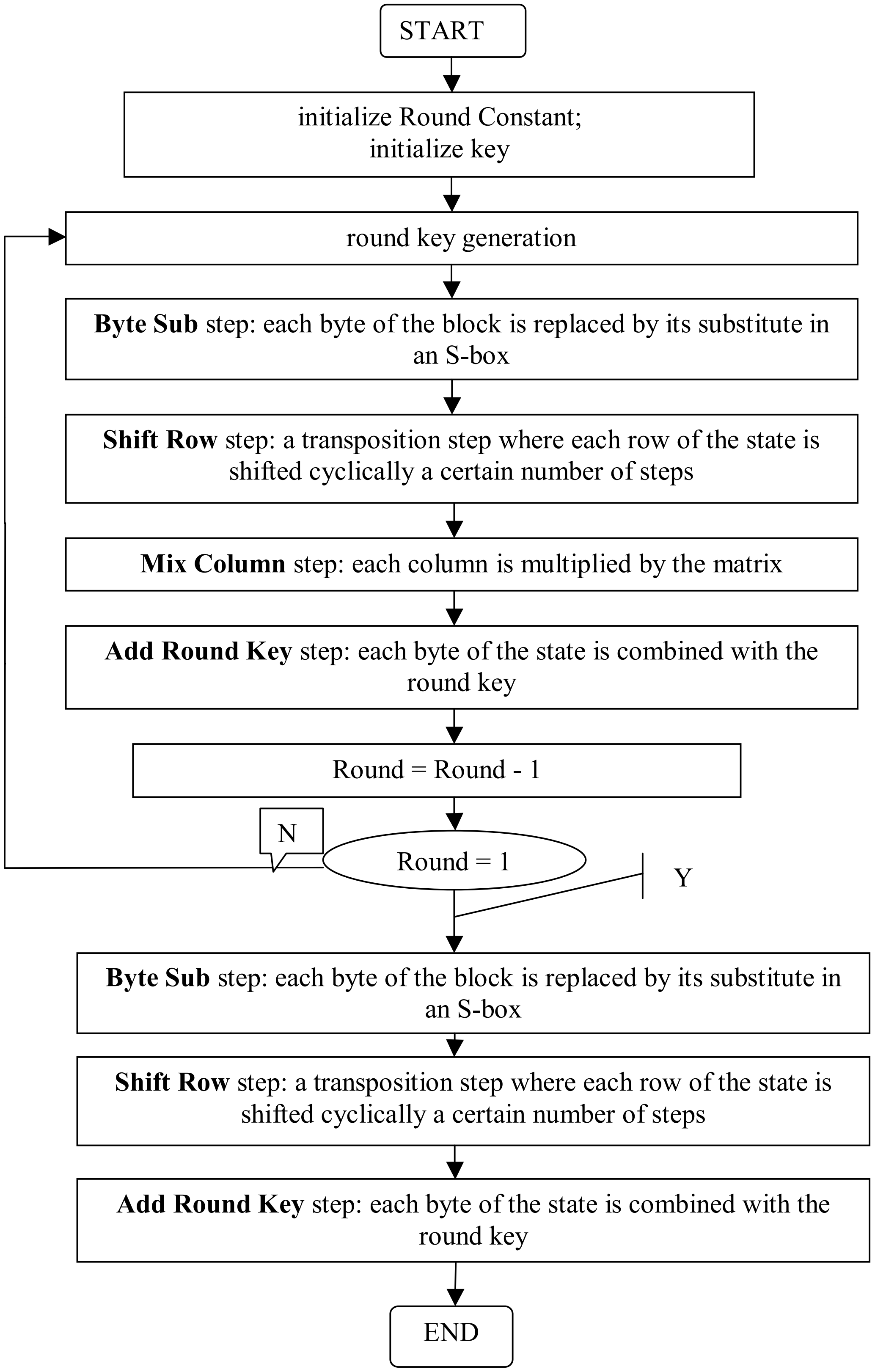
START
initialize Round Constant;
initialize key
round key generation
Byte Sub step: each byte of the block is replaced by its substitute in an S-box
Shift Row step: a transposition step where each row of the state is shifted cyclically a certain number of steps
Mix Column step: each column is multiplied by the matrix
Add Round Key step: each byte of the state is combined with the round key
Round = Round - 1
N
Round = 1
Y
Byte Sub step: each byte of the block is replaced by its substitute in an S-box
Shift Row step: a transposition step where each row of the state is shifted cyclically a certain number of steps
Add Round Key step: each byte of the state is combined with the round key
END

Figure 2.19: Flow Chart of RIJNDAEL

Comments of RIJNDAEL: If the embedded systems support 32-bit or larger words, it is possible to speed up execution of this cipher by combining “SubBytes” and “ShiftRows” with “MixColumns”, and transforming them into a sequence of tables. This requires four 256-entry 32-bit tables, which utilizes a total of four kilobytes (4096 bytes) of memory—one kilobyte for each table. A round can now be done with 16 tables and 12 32-bit exclusive-or (XOR) operations, followed by four 32-bit exclusive-or operations in the “AddRoundKey” step. If the resulting four kilobyte table size is too large for a given target platform, the table operation can be performed with a single 256-entry 32-bit table by the use of circular rotates. Using a byte-oriented approach it is possible to combine the “SubBytes”, “ShiftRows”, and “MixColumns” steps into a single round operation.

2.4 Symmetric cryptography Modes

There are several different modes of symmetric cryptography. Different modes have different considerations: some are more concerned about security level, while others may be concerned more about efficiency or fault tolerance. In the following sectors we will discuss the most frequently used modes:

2.4.1 ECB

Electronic codebook (ECB) mode is the most obvious way to use a block cipher: A block of plaintext encrypts into a block of cipher text. Since the same block of plain text always encrypts to the same block of cipher text, it is theoretically possible to create a code book of plaintexts and corresponding cipher texts.

This is the easiest mode to work with. Each plaintext block is encrypted independently. We don’t have to encrypt a file linearly; we can encrypt the 10 blocks in the middle first, then the blocks at the end, and finally the blocks in the beginning. This is important for encrypted files that are accessed randomly, like a database. If a database is encrypted with ECB mode, then any record can be added, deleted, encrypted, or decrypted independently of any other record-assuming that a record consists of a discrete number of encryption blocks. This mode is very suitable for parallel processing.

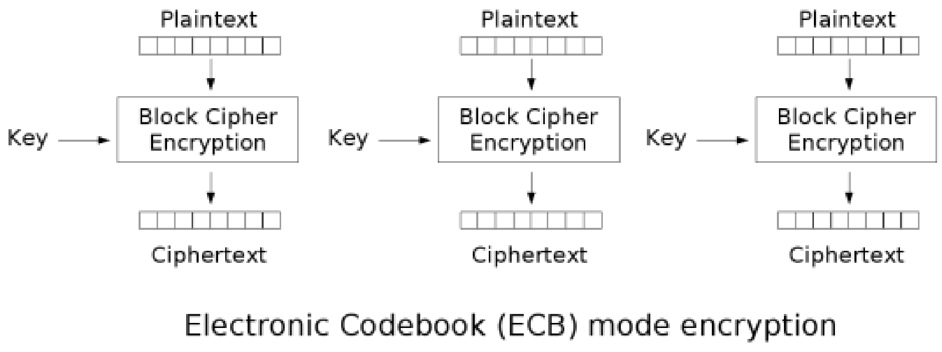

Figure 2.20: ECB mode

2.4.2 CBC

In cipher block chaining (CBC) mode, the plaintext is XORed with the previous cipher text block before it is encrypted. After a plaintext block is encrypted, the resulting cipher text is also stored in a feedback register. Before the next plaintext block is encrypted, it is XORed with the feedback register to become the next input to the encrypting routine. The resulting cipher text is again stored in the feedback register, to be XORed with the next plaintext block, and so on until the end of the message. The encryption of each block depends on all the previous blocks.

CBC mode can be characterized as feedback of the cipher text at the encryption end and feed forward of the cipher text at the decryption end. This has implications having to do with errors. A single bit error in a plaintext block will affect that cipher text block and all subsequent cipher text blocks. This isn't significant because decryption will reverse that effect, and the recovered plaintext will have the same single bit error. CBC is very suitable for high reliable error correcting environment.

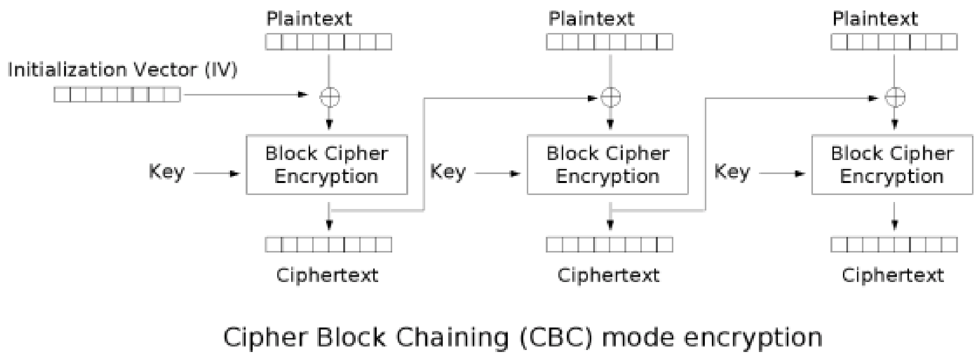
Plaintext
Initialization Vector (IV)
Key
Block Cipher Encryption
Ciphertext
Cipher Block Chaining (CBC) mode encryption

Figure 2.21: CBC mode

2.4.3 CFB

Block ciphers can also be implemented as a self-synchronizing stream cipher; this is called cipher-feedback (CFB) mode. With CBC mode, encryption cannot begin until a complete block of data is received. This is a problem in some network applications. In a secure network environment, for example, a terminal must be able to transmit each character to the host as it is entered. When data has to be processed in byte-sized chunks, CBC mode just won't do. In CFB mode, data can be encrypted in units smaller than the block size.

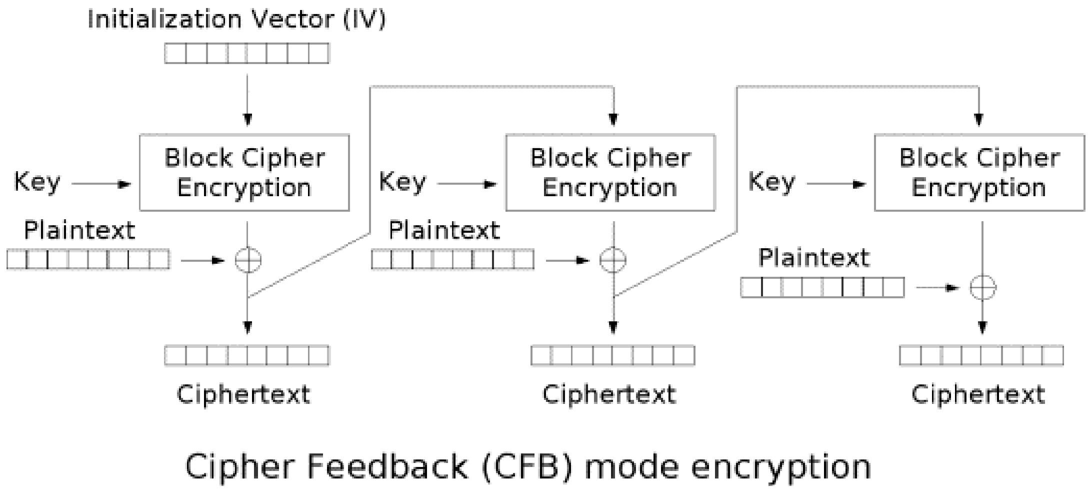

Figure 2.22: CFB mode

2.4.4 OFB

Output-feedback (OFB) mode is a method of running a block cipher as a synchronous stream cipher. It is similar to CFB mode, except that n bits of the previous output block are moved into the right-most positions of the queue. Decryption is the reverse of this process. This is called n-bit OFB. On both the encryption and the decryption sides, the block algorithm is used in its encryption mode. This is sometimes called internal feedback, because the feedback mechanism is independent of both the plaintext and the cipher text streams.

One nice feature of OFB mode is that most of the work can be offline, before the plaintext message even exists. When the message finally arrives, it can be XORed with the output of the algorithm to produce the cipher text.

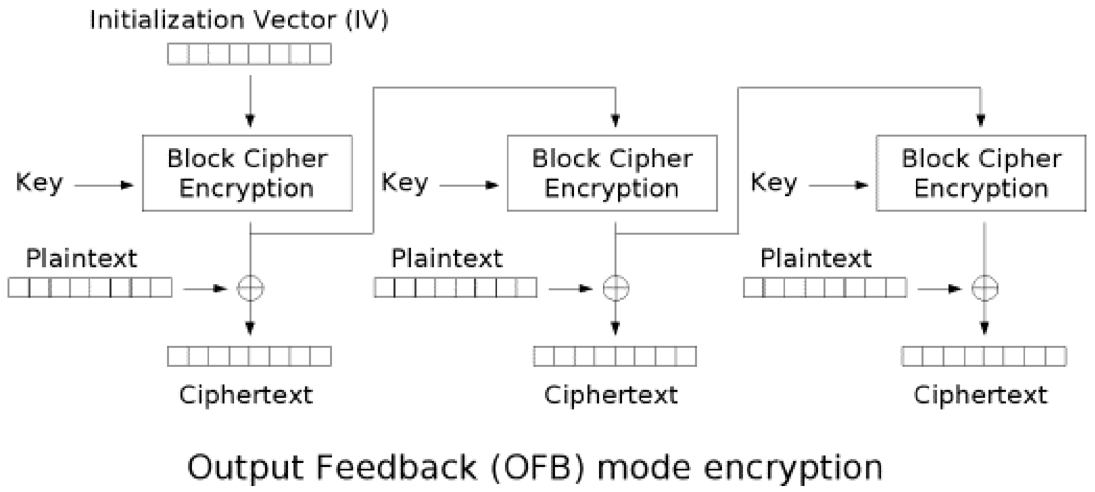

Figure 2.23: OFB mode

Chapter 3

Profiling and Analysis

In the previous chapter we have discuss the background concept of symmetric ciphers. This chapter we will profile all symmetric ciphers which have been discussed in chapter 2. In order to give good guidance of designing a novel, minimalistic, low-power processor suitable for a large subset of biomedical applicationsctives, we will analysis the average and peak power consumption, total energy budget, encryption rate and efficiency of all candidates and select one cipher as our benchmark.

3.1 Profiling System Overview

The profiling system overview comprises four parts: profiling metrics, profiling scenario, operation mode selection and input datasets.

3.1.1 Profiling Metrics

The implementation of cryptographic systems presents several requirements and challenges for embedded systems. First, the performance of the algorithms is often crucial. One needs encryption algorithms to run at the transmission rates of the communication links. Slow running cryptographic algorithms translate into consumer dissatisfaction and inconvenience. On the other hand, fast running encryption might mean high product costs since higher speeds were achieved through custom hardware devices. Moreover, the biomedical-implant field calls for further particular design requirements and constraints. Therefore, it is crucial to analyze average & peak power consumption, the total energy cost and the encryption rate as our profiling metrics.

3.1.2 Profiling Scenario

In the context of a typical application scenario, as discussed previously, (outbound) data telemetry takes place a lot more often than (inbound) command reception in implants. In effect, we are focused here on the encryption part of the profiled algorithms. Furthermore, due to their symmetric nature, most of these algorithms have the same computational requirements for both encryption and decryption.

3.1.3 Operation Mode Selection

Operation mode is the way for encrypting a message longer than the block size of an algorithm which has been discussed in chapter 2. In this thesis work we only consider the Electronic Codebook (ECB) mode. It has been shown that different operation modes (e.g. CBC, CFB and OFB) incur different fault-tolerance levels with regard to information loss due to transmitted-packet loss but incur the same energy penalty. Since in this work we are not investigating the efficiency of different modes in terms of information integrity but, rather, profile ciphers based on their power performance, ECB is sufficient for our thesis work.

3.1.4 Input Datasets

The nature of the input datasets (i.e. plaintexts) does not affect the behavior of the studied encryption algorithms except for their size. For other algorithms, such as biological data compression, the nature of the input datasets does impact the performance. However, in the interest of completeness, we have tested the algorithms against one biological workload provided from the BIOPAC (R) Student Lab PRO v3.7 software [45].

The workload (BP) represents continuous blood-pressure readouts. Our prior extensive study [46] on biomedical implants has revealed that typical data-memory sizes inside the implants range from 1 KB to 10 KB. Therefore, BP workloads of both sizes (1 KB and 10 KB, roughly) have been profiled. The specifics can be seen in Table 3.1.

test name	Size (B)	Samples (#)	Duration (sec)	Sample rate (sml/sec)	Sample rate (KB/sec)
Blood Pressure (BP) 1KB	1404	141	0.282	500	4.86
Blood Pressure (BP) 10KB	12798	1198	2.396	500	5.22

Table 3.1 Biomedical workloads used for profiling.

3.2 Experimental Setup

The experimental setup comprises three parts: the simulator setup, encryption algorithms setup and the selected benchmark suite.

3.2.1 The Simulator

The profiling of symmetric ciphers has been based on XTREM [47], a modified version of SimpleScalar [48, 49]. The XTREM simulator is a cycle-accurate, microarchitectural, power- and performance- functional simulator for the Intel XScale core. It models the effective switching node capacitance of various functional units inside the core, following a similar modeling methodology to the one found in Wattch [50]. XTREM has been selected for its straight-forward functionality but mostly for its high precision in modeling the performance and power of the Intel XScale core [51]. More precisely, it exhibits an average performance error of 6.5% and an even smaller average power error of 4% [52].

The main XScale characteristics are summarized in Table 3.2. Many of the architectural features have been integrated into XTREM. Thumb instructions and special memory-page attributes are not supported but they do not affect simulation results since they are not used by our benchmarked applications. XTREM allows monitoring of 14 different functional units of the Intel XScale core: Instruction Decoder (DEC), Branch-Target Buffer (BTB), Fill Buffer (FB),Write Buffer (WB), Pend Buffer (PB), Register File (REG), In-struction Cache (I$), Data Cache (D$), Arithmetic-Logic Unit (ALU), Shift Unit (SHF),Multiplier Accumulator (MAC), Internal Memory Bus (MEM), Memory Manager (MM) and Clock (CLK).

Although XScale (and, thus, XTREM) is a low-power processor with aggressive power-management features, we are well-aware that it is not suitable for biomedical implants in terms of power consumption. However, our selection has been based on availability and on the crucial fact that XTREM models actual hardware with very high accuracy. Moreover, in this study we focus on the relative behavior and performance of different encryption algorithms rather than on absolute figures. We are interested in the differences observed across the various ciphers and resulting trends are highly probable to stay the same in our envisioned biomedical processor.

Last, clock frequency has been lowered to closer resemble realistic implantable systems. Other parameters have not been tampered with since it is not certain that the simulator will scale properly. For instance, instruction and data TLBs have not been disabled; the operating voltage or the memory latencies have not been altered.

feature	value
ISA	32-bit ARMv5TE-compatibility,8 DSP instructions
Pipeline depth	7/8-stage, super-pipelined
Datapath width	32-bit
RF size	16 registers
Issue policy	in-order
Instr. Window	single-instruction
I-Cache	32KB 32-way set-assoc. (1-cc hit/170-cc miss lat.)
D-Cache	32KB 32-way set-assoc. (1-cc hit/170-cc miss lat.)
TLB	32-entry fully-assoc.
BTB	128-entry direct-mapped
Branch Pred.	2-bit Bimodal
Write Buffer	8-entry
Fill Buffer	8-entry
Mem. bus width	4-byte
INT/FP ALUs	4 / 4
DSP co-proc.	40-bit, low-power, variable-lat. MAC
Clock freq.	2 MHz (typ. 200 MHz)
Oper. Voltage	1.5 Volt
Implem. Tech.	0.18 um

Table 3.2 XScale architecture details.

3.2.2 The Encryption Algorithm

When putting together our benchmark suite of ciphers, we have made an effort to include sources adhering to the following characteristics:

- Large range of symmetric encryption techniques and styles, from high-performing to compact flavors;

- Mature, optimized, well-documented implementation code base;
- Various algorithm complexities;
- Suitability: the XTREM simulator can only handle C and Java sources. Furthermore, in its current version it does not support simulating an OS on top of the simulated hardware, thus prohibiting the use of encryption sources - such as the excellent bzip2 algorithm that require multithreading support or other high-level features;
- Availability: all collected benchmarks comprise utterly free, published or free under the GNU General Public License sources, readily available to the research community.

3.2.3 The Selected Benchmark Suite

The implementation of a given encryption algorithm plays crucial role for the performance and behavior of the algorithm as its underlying structure. While adhering to the above characteristics, in order to offer the best possible fairness in our selection process, we have attempted to include algorithms built with the same implementation philosophy (e.g. algorithm suite implemented by the same author(s)) and/or algorithms being top representatives in their category. Table 3.3 summarizes our selected benchmark suite. An implementation of the original DES algorithm, although considered not secure any longer, has been included in our benchmark suite as a reference point for the rest of the considered ciphers.

encryption algorithm	**key size (bits)**	**Security margin**[2]
GOST	256	2243
BLOWFISH	128	2076
IDEA	128	2076
RC5	128	2076
XXTEA	128	2076
MISTY1	128	2076
RC6	128	2076
TWOFISH	128	2076
RIJNDAEL	128	2076
3WAY	96	2034
SKIPJACK	80	2013
LOKI91	64	1992
DES	56	1982

Table 3.3 Benchmark suite of symmetric ciphers

[2] The background information of security margin can be obtained from chapter 2.2 terminology of symmetric cryptography algorithms.

3.3 Profiling Analysis

The profiling analysis comprises three parts: power consumption, energy consumption and efficiency.

3.3.1 Power consumption

We start our profiling study by firstly examining how the selected ciphers perform in terms of power consumption since this is a crucial attribute of energy-constrained devices as implants. Overall and per-component average power consumption is depicted in Fig. 3.1 for all 13 ciphers and for the two BP plaintext sizes 1KB and 10 KB.

Across all ciphers we can readily see in the figure that the memory-manager unit (MM) is the most power-hungry component of the processor with a rough 69% fraction of overall power consumed. The MM unit is activated each time the core is stalled because of a main-memory instruction or data access. Next follow the ALU consuming roughly 18%, the clock structure (CLK) consuming 5% and the instruction-cache (I$) consuming 3:5% of the overall power, on aver-age. Compared with other types of workloads, e.g. data compression, encryption is more computationally intensive (i.e. many arithmetic and logic operations), thus the high consumption of the ALU is not surprising. Further, encryp-tion is typically data- rather than control-dominated, with few instruction branches, placing high demands on linear in-struction fetch. That is why the instruction-cache consumes on average more power than other memory units, e.g. the data-cache or the BTB. Last, the clock structure is known throughout digital systems to be a significant component of power consumption, which is also the case here. If we operated the processor at a higher frequency, power consumption would increase considerably. In terms of plaintext sizes, overall average power consumption increases insignificantly (about 3%) with input size. Essentially, in the range from 1KB to 10KB of plaintext size which is of interest for our case, power consumption does not seem to be affected. This agrees also with the findings of Law et al. [23]. In accordance to the same work as well as our own measurements, a significant difference in consumed power would be observed in plaintext sizes comparable to the block size of the ciphers, i.e. 10 to 30 Bytes. In this range, key-initialization tasks place a computational overhead comparable to the actual encryption process. This indicates that encryption becomes more power-efficient with larger plaintext sizes.

A final observation from Figure 3.1 is that the power-behavior of the ciphers does not change with increasing plaintext size, at least in the range of interest. There is one exception: 3-WAY and XXTEA switch places when moving to the larger plaintext but this is of minimal significance since they both score the poorest in terms of average power consumption. The best performing ciphers on this metric are IDEA, LOKI91, SKIPJACK, MISTY1 and RIJNDAEL. Although DES is included in the study as a reference point, it cannot be selected as a winning candidate in the profiling due to its compromised status. It is interesting, however, to observe that it features one of the lowest power profiles even though it is one of the oldest encryption algorithms.

Except for average power consumption, another interesting metric is peak power consumption. This is especially important for battery-powered systems such as implants are. A battery able to support a cipher with a given average power consumption may be unable to deliver the required output at a given point in time if the cipher sporadically presents peak power values which are largely deviating from its average power needs. To address this aspect of the profiled ciphers, we have plotted Figure. 3.2. The ciphers are depicted in order of increasing peak-power profiles. The bar series denoted as average power consumption is the aggregated equivalent of the bars seen previously, in Figure. 3.1.

It is interesting to see that ciphers scoring high in the previous test, such as IDEA and LOKI91, display a large difference of roughly 35 mW between average and peak power, which can potentially throw the implant designer off track. This difference has to be taken into serious account if such ciphers are to be employed in an implantable device. IDEA, MISTY1, LOKI91 and RIJNDAEL still occupy the first positions. However, TWOFISH in now inside the top-scoring ciphers and, what is more, it displays the most consistent profile between average and peak power. In terms of our chosen plaintext sizes, and similarly to average power, peak-power profiles present no differences.

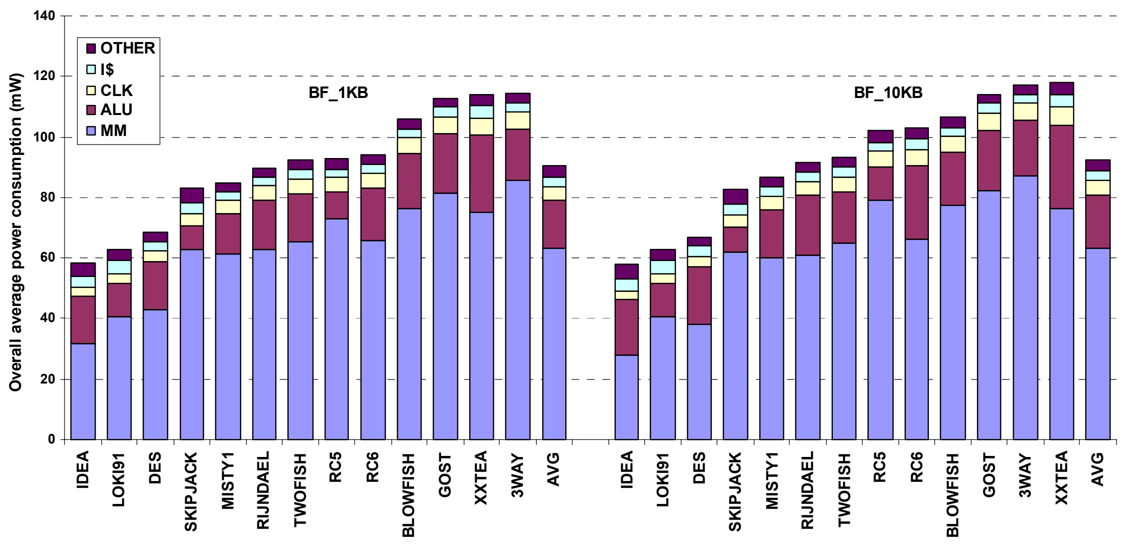

Figure 3.1 Per-component, average power consumption (in mW) for two plaintext sizes

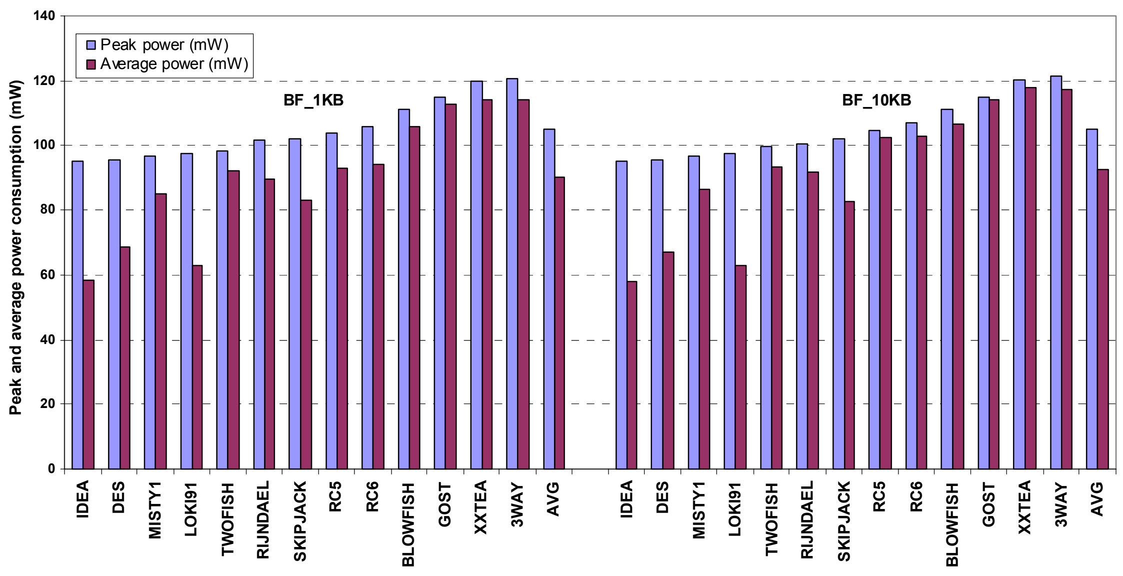

Figure 3.2 Average and peak power consumption (in mW) for two plaintext sizes

3.3.2 Energy consumption

Apart from the rate at which a cipher consumes energy, i.e. its power consumption, it is important to also investigate the total energy costs incurred for executing the whole cipher. This metric is the total energy expenditure of a cipher and our findings are summarized in Figure. 3.3, for both plain-text sizes. SKIPJACK and LOKI91 have been omitted from the plots since they display excessively large energy needs (an order of magnitude larger for LOKI91 than the rest of the algorithms). However, average values include these two algorithms in their calculation to give a complete view.

Knowing the overall energy budget needed for completing a single encryption task is especially important for implantable systems. It directly tells us how much stored energy the given task needs in order to execute and, in effect, what energy amount will be deduced from the battery. It also tells us if e.g. a scheduled encryption and transmission of physiological readouts can take place or not. Given the mission-critical tasks implants performance, it might be preferable at some point to not engage in transmission of (encrypted) data. For instance, it is more important for a pacemaker running low on battery to keep working for an extra couple of days (to allow time for recharging or servicing) than to transmit ECG readouts to its inquiring host once and then power down.

In terms of energy distribution in the various processor components, we can again see that the MM, ALU, CLK and I$ are the most demanding ones. However, Fig. 3 tells a completely different story for the energy sparing-ness of the profiled ciphers. RC6 and RC5 have climbed in the first positions of the ranking, becoming the most energy-efficient ciphers. IDEA and MISTY1 follow with RIJNDAEL and BLOWFISH contesting for the fifth position across the two different plaintexts. Clearly, MISTY1 and RIJNDAEL perform better when smaller plaintext sizes are considered. Conversely, BLOWFISH favors larger sizes. Further, it is surprising that XXTEA is not among the best-scoring ciphers since it is considered a relatively light-weight algorithm.

A last observation in this subsection is that energy budget does not scale linearly with plaintext size for most of the ciphers. The cost of encrypting a 10-KB workload as opposed to that of successively encrypting 10 1-KB workloads is 14% smaller, in an overall. The reason for that difference again is the overhead penalty paid during initialization of the encryption algorithms (e.g. key setup). As our simulations have revealed, other factors also contributing to this penalty are the increased fetch- and data-stalls that are reduced over the execution time of a cipher as cache entries get filled, etc. However, this penalty is not similar across the various ciphers. In Figure.3.4, the energy budgets for encrypting one 10-KB workload and 10 consecutive 1-KB workloads are plotted. The ciphers are ranked in order of increasing difference between the two budgets, i.e. in order of increasing penalty. RC6, IDEA, RC5, MISTY1 and RIJNDAEL are still in the first positions, incurring small penalties but TWOFISH has fallen near the bottom of the ranking, due to introducing a significant energy penalty. This secondary metric of energy is interesting because it indirectly gives a measure of computational efficiency of the various ciphers.

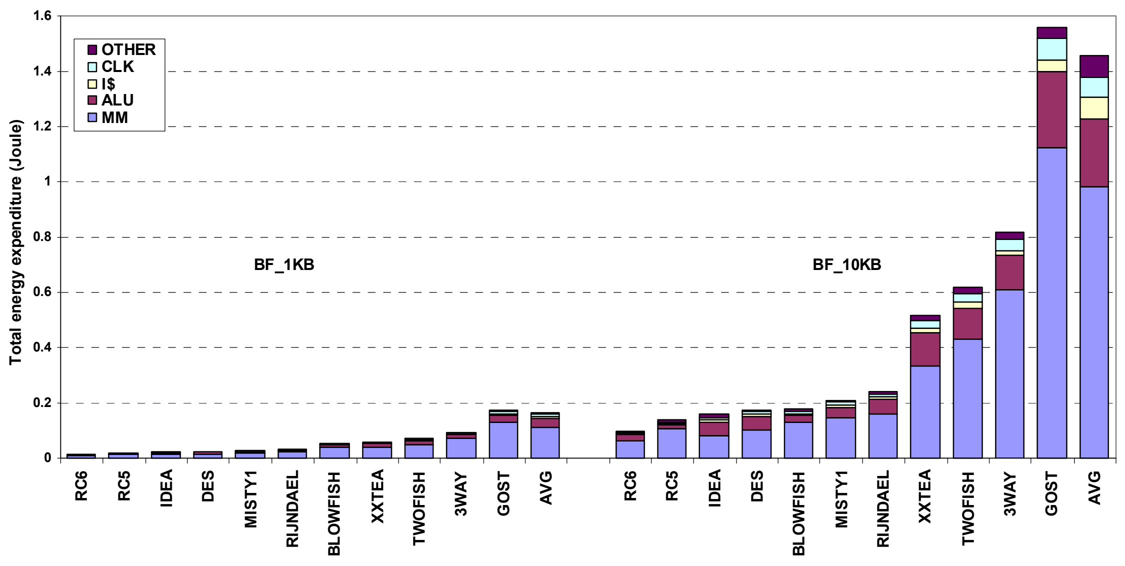

Figure 3.3 Per-component and total encryption energy costs (in Joules).

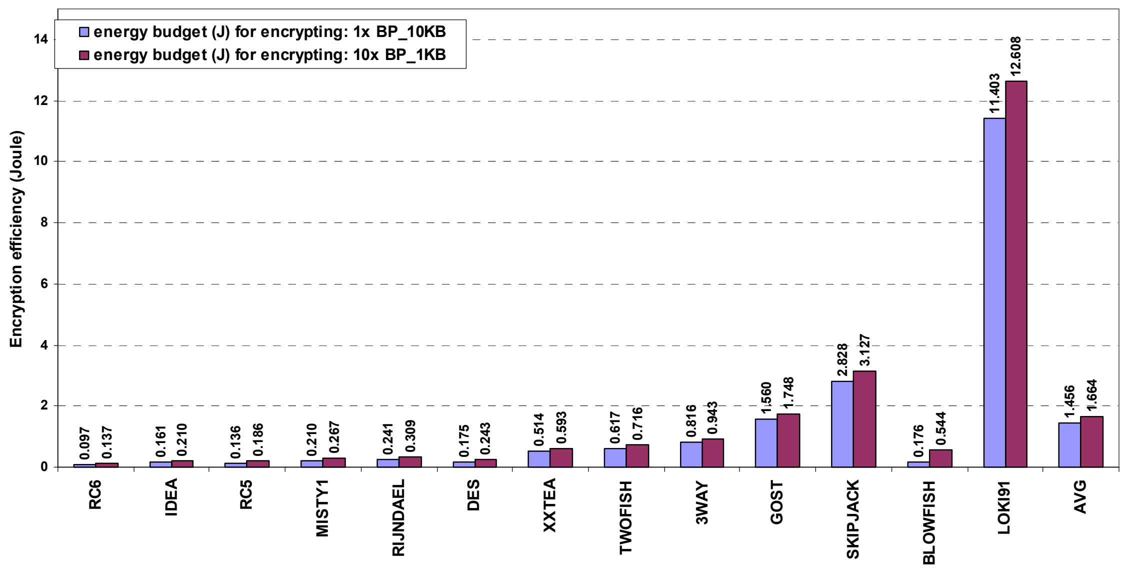

Figure 3.4 Computation overhead of ciphers manifested as energy penalty (in Joules) when encrypting one 10-KB and ten 1-KB plaintexts.

3.3.3 Efficiency

Another metric we use in our profiling work of block ciphers is their encryption rate. In Figure 3.5 encryption rates in KB/sec are reported for 1-KB and 10-KB plaintexts. RC6, RC5, MISTY1, RIJNDAEL and BLOWFISH score the highest on this metric, with RC6 and RC5 being by far the fastest ciphers. In fact, and contrary to the rest of the ciphers, RC6 and RC5 achieve impressive encryption-rate improvements with increasing plaintext size. The rate of BLOWFISH appears also to benefit largely from a larger plaintext size. In an overall, all ciphers seem to benefit from larger plaintexts: from 3:34 KB/sec for the 1-KB data, the average rate boosts to 4.52 KB/sec. The reasons for this are the same as the ones previously mentioned concerning the energy penalty. They are related to the cipher-key initialization phase as well as the cold start of the processor itself.

For the targeted implant applications, our primary concern is to preserve power consumption at low levels. This means that we are not seeking the fastest performing cipher but, rather, one which is fast enough to cover our needs. As can be seen in Table 3.1, the biological data we used as plaintext features a relatively high (in this context) sampling rate of 4.86 KB/sec for the 1-KB and 5.22 KB/sec for the 10-KB workload. In our 2-MHz simulated processor, only ciphers RC6 and RC5 manage to sustain the required sampling rate. The cost paid is that both ciphers display a relatively high power profile (93 mW to 100 mW) as seen in subsection 3.3.1.

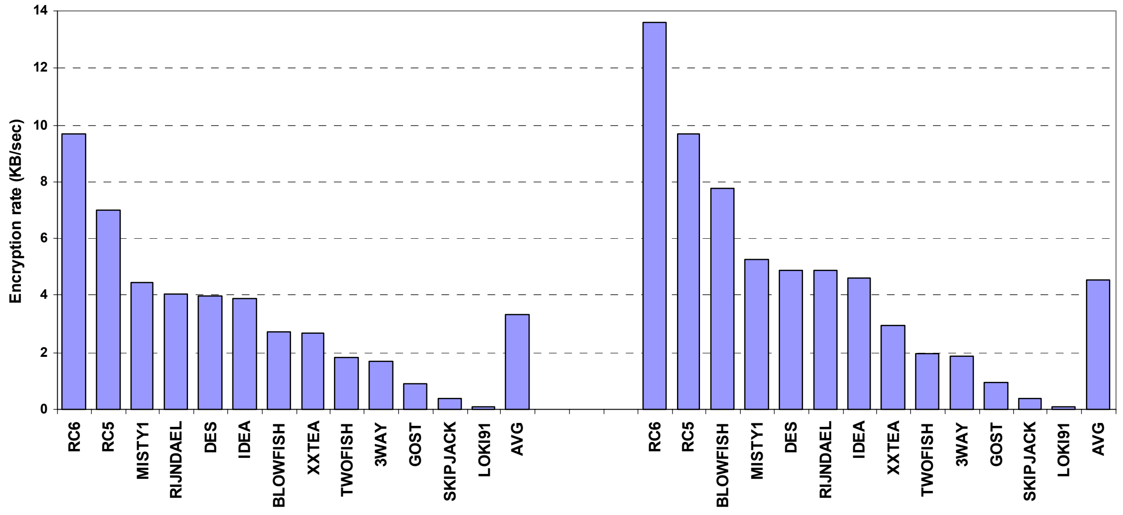

Figure 3.5 Encryption rate (in KB/sec).

3.4 Cipher selection and result

To summarize our analysis results, we present in Table 3.4 the 5 best-performing algorithms in each one of our profiled metrics (except for the security-margin metric). According to table 3.5, MISTY1 appears in all 5 metrics of the above table. IDEA, RIJNDAEL, RC6 and RC5 follow with each with 4 occurrences in the table. However, IDEA performs consistently better than RIJNDAEL with the exception of encryption rate. Besides, RIJNDAEL scores almost always last in the ranking among ciphers with 4 occurrences. Last, RC6 scores always better than RC5. LOKI91 has 3 occurrences in the table but is, in any case, dismissed due to its now insecure nature.

average power consumption	**peak power consumption**	**total energy cost**	**encryption efficiency**	**encryption rate**	**program-code size**
IDEA	IDEA	RC6	RC6	RC6	XXTEA
LOKI91	MISTY1	RC5	IDEA	RC5	3WAY
SKIPJACK	LOKI91	IDEA	RC5	MISTY1	LOKI91
MISTY1	TWOFISH	MISTY1	MISTY1	RIJNDAEL	RC6
RIJNDAEL	RIJNDAEL	BLOWFISH	RIJNDAEL	BLOWFISH	RC5

Table 3.4 Five best-performing encryption algorithms
(In descending order of performance)

algorithm	occurrences	comment
MISTY1	5	-
IDEA	4	except for encr. rate, always better than RIJNDAEL
RIJNDAEL	4	among ciphers with 4 occurences, it scores almost always last in the ranking
RC6	4	always better than RC5
RC5	4	-
LOKI91	3	
BLOWFISH	2	
SKIPJACK	1	
TWOFISH	1	
3WAY	1	
XXTEA	1	

Table 3.5 the rank of best-performing encryption algorithms
(According to the occurrences rate)

Conclusively, from the above findings, MISTY1 is the most promising cipher according to our imposed metrics; thus, we take a closer look at its underlying instruction mix. Table 3.6 illustrates the type and frequency of instructions executed for encrypting the 1-KB BP plaintext with the MISTY1 cipher. XTREM, which is based on SimpleScalar, implements ARM instructions through microcode (referred to as u - ops hereon). We included u - op statistics rather than ARM instruction because they can better capture the workings of the underlying architecture. For instance, a single ARM command to store multiple registers to the stack pointed to by R13, breaks down to a number of more elementary u - ops (see Table 3.6).

ARM instruction	Equivalent ARM microcode
stmdb r13!,{r4-r8,r10-r15}	agen tmp1,r13,0
	agen tmp0,tmp1,-16
	stp r11,[tmp0]
	agen r13,r13,-16
	agen tmp0,tmp1,-12
	stp r12,[tmp0]
	agen tmp0,tmp1,-8
	stp r14,[tmp0]
	agen tmp0,tmp1,-4
	stp r15,[tmp0]

Table 3.6 Sample ARM instruction which stores registers in the stack pointed to by R13 and equivalent ARM u - OP sequence

Going back to Fig. 3.6 and Table 3.7, we readily observe that execution is heavily dominated by load/store (stp, ldp) operations, logic operations (eor, and, orr) and register-to-register copy operations (mov). Such a mix also explains the dominance of the MM and ALU components in the power-consumption plots, as previously discussed. This mix motivates us to-wards effcient implementation of loads/stores, moves and logic operations in terms of power consumption and speed. Note that "agen" is always used to calculate an address before load/store-related operations and, thus, is not considered as a stand-alone instruction but as part of those operations.

u-op	count	percentage
agen (address generation)	155134	30%
ldp (load from private memory)	109861	21%
eor (logical xor)	71909	14%
mov (move)	63862	12%
stp (store to private memory)	36527	7%
and (logical and)	23523	5%
orr (logical or)	19073	4%
other	40405	8%
TOTAL	**520294**	

Table 3.7 MISTY1 on 01_KB

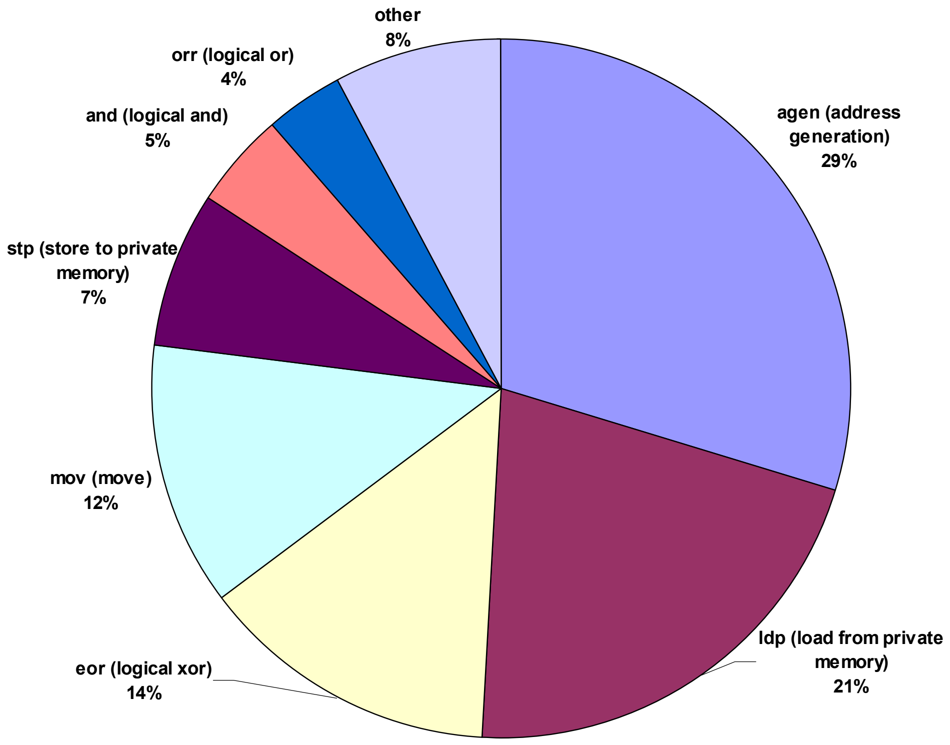

Figure 3.6 u-op mix and frequencies for MISTY1 operating on the 1-KB BP plaintext.

By investigating also the second and the third best ciphers, i.e. IDEA and RC6, we accumulate the following statistics, seen in Table 3.8. The mixes in this case favor load/store and move operations highly but logic operations to a smaller extent, compared to the MISTY1 case. However, they both display high percentages of arithmetic (add, sub, rsb, cmp) and branch (b) operations, contrary toMISTY1. Given that MISTY1 scores high in most metrics of our profiling study, optimizing our architecture for the more focused MISTY1 instruction mix alone is considered the best option.

IDEA		RC6	
u-op	percentage	u-op	percentage
mov (move)	29%	agen (address generation)	25%
agen (address generation)	18%	mov (move)	15%
ldp (load from private memory)	12%	ldp (load from private memory)	15%
b (unconditional branch)	9%	stp (store to private memory)	8%
add (add)	7%	add (add)	8%
cmp (compare)	6%	b (unconditional branch)	5%
stp (store to private memory)	5%	eor (logical xor)	4%
orr (logical or)	4%	sub (subtract)	4%
other	10%	rsb (reverse subtract)	4%
		other	11%

Table 3.8 u-op mix and frequencies for IDEA and RC6 operating on the 1-KB BP plaintext

Chapter 4

Conclusions and Future Works

This chapter discusses the conclusions from the current thesis project. Future work, possible improvements and acknowledgements on the project are also discussed in this chapter.

4.1 Conclusions

The objective of the thesis project was to profile symmetric-encryption algorithms in order to give suggestion for designing low-power digital computer architecture for the SiMS project. In this perspective, the project has completed the following:

1 Described and analyzed encryption algorithms in depth and in detail.
2 Did survey 13 block ciphers in terms of underlying structure, advantage and disadvantage, security level, and code size.
3 Did comment on all 13 block algorithms regarding unique traits and specific which are useful for profiling and analyzing for the SiMS digital architectures.
4 Implemented various symmetric encryption algorithms by using XTREM simulator.
5 Examined numbers of metrics of all symmetric-encryption algorithms on a workload suite of recorded biological signals.
6 Compared the relative behavior of ciphers operating on different workload sizes. Qualitative results regarding the trade-offs among different metrics such as encryption rate and average power consumption, peak power consumption, energy consumption, etc, have been offered.
7 After performing the above tasks, MISTY1 was selected as the winner of the profiling study and identify the most common instructions executed by MISTY1.

4.2 Future Work

This project has defined the benchmark for encryption algorithms suitable for biomedical devices. However, our thesis project concentrated on symmetric encryption algorithms and did not take into consideration of any asymmetric algorithms. In the future work, we will:

- Investigate the asymmetric encryption and decryption algorithms and implement several algorithms with different workloads to measure the metrics of them.

- Extend our benchmark suite with different application classes such as losses data-compression which profile compact storage and over-the-air transmitted data.
- Investigate potential problems and benefits when merging different applications together such as merge of encryption algorithms and compression algorithms.

4.3 Possible Improvements

1. *We need to optimize the structure of algorithms to enhance the performance of algorithms.*

   Many symmetric encryption algorithms have redundancy structure which keeps the algorithm secure. Some crypto analyzers already optimized parts of the redundancy structure of some encryption algorithms and still kept the algorithms secure. But there is still no evident that optimized algorithms are suitable for Implantable Medical Device. More lab experiments are needed to be implemented to convince the suitability of optimized encryption algorithms.

2. *We can reduce the rounds of algorithms but keep the reasonable secure level:*

   The more rounds the algorithm has, the more secure the algorithm will be. However, more rounds will take additional CPU resources and increase the power consumption. To reduce the round of algorithms but keep it still secure is an efficient method to reduce the power consumption and other metrics. Crypto analyzers already found the least rounds margin of many encryption algorithms. But whether this least rounds margin are suitable for IMD is still requested to testified. More lab experiments are required to test the least round margin of encryption for Implantable medical devices.

Bibliography

[1] M. Abramovici, et. Al., "A Reconfigurable Design for-Debug Infrastrcuture for SoCs", Proceedings of the 43rd annual conference on Design automation, 2006
Ector, H., and Vardas, P. Current use of pacemakers, implantable cardioverter defibrillators, and resynchronization devices: data from the registry of the european heart rhythm association. *European Heart Journal Supplements 9* (August 1988), p. 144-149.

[2] http://ce.et.tudelft.nl/SiMS/main_background.php

[3] The US Department of Health and Human Services. Summary of the HIPAA Privacy Rule, 2003.

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

[5] Daniel Halperin, Thomas S. Heydt-Benjamin, Kevin Fu, Tadayoshi Kohno, and William H. Maisel, "Security and Privacy for Implantable Medical Devices", Vol. 7, No. 1 January–March 2008

[6] www.secure-medicine.org

[7] Venkateswara Sarma Mallela, V Ilankumaran, and N.Srinivasa Rao, "Trends in Cardiac Pacemaker Batteries"

[8] Du, W., Deng, J., Han, Y., Varshney, P., Katz, J., and Khalili, A. A pairwise key predistribution scheme for wireless sensor networks. ACM Transactions on Information and System Security (TISSEC) 8 (May 2005), p. 228-258

[9] Jeong, Y.-S., and Lee, S.-H. Hybrid key establishment protocol based on ecc for wireless sensor network. Lecture Notes in Computer Science 4611 (August 2007), p. 1233-1242.

[10] Gaubatz, G., Kaps, J.-P., and Sunar, B. Public key cryptography in sensor networks - revisited. In Lecture Notes in Computer Science (2005), p. 2-18.

[11] Wander, A., Gura, N., Eberle, H., Gupta, V., and Shantz, S. Energy analysis of public-key cryptography for wireless sensor networks. In 3rd IEEE International Conference on Pervasive Computing and Communications (2005), p. 324-328.

[12] R. F. Churchhouse, “Codes and Ciphers: Julius Caesar, the Enigma, and the Internet”, Cambridge University Press; 1 edition

[13] Bengt Beckman, “Codebreakers: Arne Beurling and the Swedish Crypto Program during World War II”, ISBN 0-8218-2889-4

[14] Http://en.wikipedia.org/wiki/Symmetric-key_algorithm

[15] Http://en.wikipedia.org/wiki/Public-key_cryptography

[16] Anoop MS, “Public Key Cryptography: Applications Algorithms and Mathematical Explanations”, Tata Elxsi Ltd, India

[17] Bruce Schneier, “Applied Cryptography - Second Edition”, p 210-211, John Wiley & Sons, 1996, ISBN 0-471-11709-9

[18] Bruce Schneier, “Applied Cryptography - Second Edition”, p 223-225, John Wiley & Sons, 1996, ISBN 0-471-11709-9

[19] Claude Shannon, “A Mathematical Theory of Cryptography, Memorandum MM 45-110-02”, Sept. 1, 1945, Bell Laboratories.

[20] S. Mister and C. Adams (1996). "Practical S-Box Design". Workshop on Selected Areas in Cryptography (SAC '96)

[21] Arjen K. Lenstra, Eric R. Verheul, “ Selecting Cryptographic Key Sizes ” , J. Cryptology 14(4): 255-293 (2001)

[22] Lenstra, A., and Verheul, E. Selecting cryptographic key sizes. Journal of Cryptology: the journal of the International Association for Cryptologic Research 14, 4 (2001), p. 255-293.

[23] Law, Y., Dourmen, J., and Hartel, P. Survey and benchmark of block ciphers for wireless sensor networks. ACM Transactions on Sensor Networks 2 (February 2006), p. 65-93.

[24] National Bureau of Standards, Data Encryption Standard, FIPS-Pub.46. National Bureau of Standards, U.S. Department of Commerce, Washington D.C., January 1977.

[25] Bruce Schneier, “Applied Cryptography - Second Edition”, p 336-339, John Wiley & Sons, 1996, ISBN 0-471-11709-9

[26] Bruce Schneier, Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). Fast Software Encryption 1993

[27] Serge Vaudenay, "On the weak keys of Blowfish," Fast Software Encryption (FSE'96), LNCS 1039, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 27--32.

[28] Http://en.wikipedia.org/wiki/3-Way

[29] Bruce Schneier, “Applied Cryptography - Second Edition”, p 331, John Wiley & Sons, 1996, ISBN 0-471-11709-9

[30] Xuejia Lai and James L. Massey and S. Murphy, Markov ciphers and differential cryptanalysis, Advances in Cryptology — Eurocrypt '91, Springer-Verlag (1992), pp17–38.

[31] Http://www.unsw.adfa.edu.au/~lpb/research/loki91/

[32] Rivest, R. L. (1994). "The RC5 Encryption Algorithms", Proceedings of the Second International Workshop on Fast Software Encryption (FSE) 1994e: 86–96.

[33] Biryukov A. and Kushilevitz E. (1998). Improved Cryptanalysis of RC5. EUROCRYPT 1998.

[34] R.L. Rivest, M.J.B. Robshaw, R.Sidney, and Y.L. “Yin. The RC6 Block Cipher v1.1”, August 1998.

[35] Http://en.wikipedia.org/wiki/Skipjack_(cipher)

[36] Matthew D. Russell (2004-02-27). "Tinyness: An Overview of TEA and Related Ciphers", Archived from the original on 2007-08-12.

[37] Roger M. Needham and David J. Wheeler (October 1997), "Tea extensions", Computer Laboratory, Cambridge University, England, Retrieved on 2008-07-04.

[38] Http://en.wikipedia.org/wiki/XXTEA

[39] M. Matsui, “Block encryption algorithm MISTY”, In Fast Software Encryption, 4th International Workshop, FSE '97, LNCS 1267, pages 64–74.Springer-Verlag, 1997.

[40] Bruce Schneier, Doug Whiting, "A Performance Comparison of the Five AES Finalists", Retrieved on 2006-08-13.

[41] Http://en.wikipedia.org/wiki/Twofish

[42] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, “The Twofish Encryption Algorithm: A 128-Bit Block Cipher”, John Wiley & Sons. ISBN 0-471-35381-7

[43] “NIST reports measurable success of Advanced Encryption Standard”, Journal of Research of the National Institute of Standards and Technology, May-June, 2002

[44] Joan Daemen and Vincent Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard." Springer-Verlag, 2002. ISBN 3-540-42580-2.

[45] Http://www.lintoninst.co.uk/bsl_pro_software.htm

[46] Strydis, C., Gaydadjiev, G., and Vassiliadis, S. Implantable microelectronic devices:A comprehensive review. CE-TR-2006-01, Computer Engineering, Delft University of Technology, December 2006.

[47] Contreras, G., Martonosi, M., Peng, J., Ju, R.,and Lueh, G.-Y. XTREM: A Power Simulator for the Intel XScale Core. In LCTES'04 (2004), p. 115-125.

[48] Austin, T., Larson, E., and Ernst, D, “SimpleScalar: an infrastructure for computer system modeling”, IEEE Computer 35, 2 (February 2002), 59-67.

[49] Burger, D., and Austin, T. “The SimpleScalar tool set, version 2.0”, ACM SIGARCH Computer Architecture News 25, 3 (June 1997), p. 13-25.

[50] Brooks, D., Tiwari, V., and Martonosi, M. Wattch: “A Framework for Architectural-Level Power Analysis and Optimizations”, In ISCA'00 (2000), p. 83-94.

[51] Intel Corporation. Intel XScale Micro architecture for the PXA255 Processor: User's Manual, March 2003.

[52] Contreras, G., and Martonosi, M. The XTREM Power and Performance Simulator for the Intel XScale Core: Design and Experiences. ACM Transactions on Embedded Computing Systems 6, 1 (February 2007), p. 1-25.