CHALMERS

Implementation and Evaluation of Secure Industrial Ethernet Communication

Master of Science Thesis, Communication Engineering

KAN YU

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, August 2010

Abstract

Automation network security becomes increasingly important due to the introduction of Ethernet-based fieldbus protocols and cryptographic algorithms play a vital important role in these protocols. Choosing the most suitable cryptographic algorithms under consideration of security and performance according to different application cases is essential. In this thesis, we first present a comprehensive survey of most commonly used cryptographic algorithms which can be applied in automation networks and then identify our candidates based on existing literature and related works for further evaluation in ARM platform for industrial purpose. Finally, according to our evaluation results, we choose suitable algorithms for different applications: for symmetric algorithms, Twofish is recommended for best performance and eXtended Tiny Encryption Algorithm (XTEA) and Corrected Block Tiny Encryption Algorithm (XXTEA) are recommended for the least footprint; for Message Authentication Code (MAC) algorithms, UMAC is strongly recommended for excellent speed; for asymmetric algorithms, Elliptic Curve Cryptography (ECC) has much better performance than RSA at the same security level in our platform.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Automation Network Security

Automation technology is a synthesized technology involving control theory, information theory, system engineering, computer technology, etc. Industrial automation, which may refer to process automation or factory automation, is the most important part of automation applications.

Industrial automation network is built into a hierarchical structure. It consists of a number of different components at different hierarchy levels. For instance, it may include application specific devices, such as sensors, at the bottom, and plant LANs at the top [51]. Figure 1.1 illustrates network architecture of a process automation system. Different from other architectures, the process automation system in Figure 1.1 is isolated from the outside due to the security consideration. Since there is no connection to the outside, no security attack from outside can threaten the automation system. It shows that security is a vital important issue in designing an automation network [50].

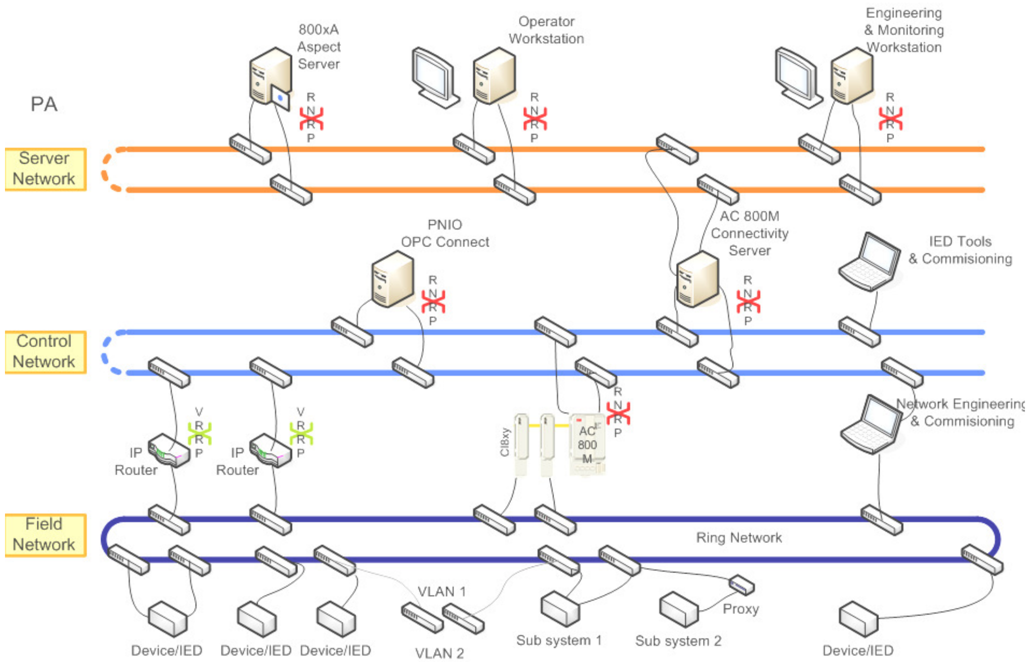


Figure 1.1 Example of Network Architecture for a Process Automation System [50]

However, not all automation systems are designed to be a closed network. There may be requirements for an automation system to communicate with the outside, such as remote control service. Thus, the inside automation network is exposed to adversaries from the outside who are able to launch different security attacks against the inside network according to different automation systems. The common method to protect the entrance is to setup a security zone between inside and outside networks, such as a firewall, to filter the incoming and outgoing data.

Automation network security issues are not only provided by network architecture, but also from communication protocols. A number of different network protocols are used at different hierarchy levels. Common used protocols at the higher level of automation networks are Object Linking and Embedding for Process Control (OPC), Manufacturing Message Specification (MMS), IEC 61850 and Inter Control Centre Protocol (ICCP) [51]. This thesis focuses more on the security at lower level. At lower field network level, different protocols are designed according to different physical linking methods. Field buses are employed in traditional field networks, which offer no security features against any security attack. Ethernet and TCP/IP protocols are used in the newly generated Ethernet-based field buses. A number of protocols are specified in IEC standards series, for more details see [50] [51]. Compared with traditional field networks, such networks are more vulnerable to security attacks, so security services should be provided. Nowadays, wireless systems are commonly used at device level. Due to the radio connection, automation systems are even more vulnerable to security attacks than the former one. Security is greatly considered and well addressed in the wireless sensor network protocols, such as WirelessHART [117] and ISA 100 [118]. Power line is another method to construct field networks. A number of standards specify the way to transmit data on power lines. For more details see [51].

1.2 Research Problem

Nowadays, since Ethernet-based field buses are increasingly used in automation systems, more and more attentions are paid to the security of Ethernet-based fieldbus protocols. However, most protocols were designed before with fewer concerns about security. For instance, PROFINET IO, one of the Ethernet-based fieldbus protocols, has no explicit security countermeasures in the protocol, so a number of possible attacks can be launched against PROFINET IO nodes to get unauthorized access. [50] has shown the concept of security modules to provide security in PROFINET IO without violating the standards, but cryptographic algorithms to provide confidentiality or message authentication are still not specified. Another problem is that at the lower field bus level, application specific devices are most embedded devices has less CPU processing power and memory than desktop computers. Therefore, cryptographic algorithms should be carefully chosen with these considerations.

1.3 Research Approach

In order to choose the most appropriate cryptographic algorithms for automation fieldbus networks balanced both in security and performance, a survey of cryptographic algorithms based on existing literature and authoritative recommendations is necessary. This survey involves most widely used cryptographic algorithms including symmetric algorithms, asymmetric algorithms and MAC algorithms. Then several candidates from each catalog will be chosen for further evaluating and assessing based on the comparison of security and performance between each other. The software implementations of all candidates are written in pure C language based on existing open source projects or cryptographic libraries. Finally, existing automation equipment is used as benchmarking platform to evaluate the performance of all candidates and followed by our analysis. The goal of this thesis is to propose suggestions of cryptographic algorithms suitable for automation field networks.

1.4 Thesis Contributions

The main contributions of this thesis are:

1. A comprehensive survey of most commonly used cryptographic algorithms including symmetric algorithms, asymmetric algorithms and MAC algorithms with security and performance analysis.
2. Evaluation of candidate algorithms using software implementation and a comparison with the performance of all candidates.
3. Propose suggestions of suitable cryptographic algorithms in our specific platform according to different applications: for symmetric encryption, Twofish is recommended for the best performance and XTEA or XXTEA for the least footprint; for MAC algorithms, UMAC is recommended for the best performance, CMAC for applications with AES already implemented; for asymmetric algorithms, ECC is recommended compared to RSA.

1.5 Thesis Outline

This report has the following structure.

Chapter 1 Introduction describes the purpose and scope for this thesis.

Chapter 2 This chapter introduces the basic concept of security, security attacks, cryptography and network security. It gives a brief understanding to readers who are not familiar with this domain.

Chapter 3 In this chapter, a comprehensive survey of cryptographic algorithms including symmetric algorithms, asymmetric algorithms and MAC algorithms is presented. The candidates are selected based on the security and performance analysis.

Chapter 4 This chapter introduces our benchmarking platform, the source of software implementation of all algorithms, the cipher parameters we chose in our evaluations and the methods we use to measure the performance.

Chapter 5 This chapter presents the evaluation results of all algorithms. Analysis and suggestions are given according to the comparison of performance.

Chapter 6 This chapter presents the conclusions and future works after this thesis.

2 SECURITY AND NETWORK SECURITY

2.1 Security

In ancient wars, keeping information unknown from enemies is one of the essential parts to the victory. As rapid development of communication and network technology, what we said, we wrote and we behaved are always under the threat of being monitored, eavesdropped, modified and analyzed by any adversary[1]. From [3], several types of adversaries and their goals are listed in Table 2.1. Therefore, the security of information is considered to be more and more important.

Table 2.1 Examples of people who cause security problems and why [3]

Adversary	Goals
Student	To have fun snooping on people's e-mail
Cracker	To test out someone's security system; steal data
Sales rep	To claim to represent all of Europe, not just Andorra
Businessman	To discover a competitor's strategic marketing plan
Ex-employee	To get revenge for being fired
Accountant	To embezzle money from a company
Stockbroker	To deny a promise made to a customer by e-mail
Con man	To steal credit card numbers for sale
Spy	To learn an enemy's military or industrial secrets
Terrorist	To steal germ warfare secrets

Security mainly concerns confidentiality, integrity and availability [2].To understand security in communication and networking, knowledge of related issues, such as cryptography and network security is necessary. This part, we introduce several important issues related to information security as the preliminary knowledge of the thesis.

2.2 Objectives

Not only keeping information unknown from adversary is required, the objective of security also refers to other issues. A summary of some information security objectives, shown in Table 2.2, is given in [60]. During a transaction, all parties involved should make sure that certain security objectives have been achieved.

[1] An adversary in this thesis means a person or orgnization who intends to obtain crucial or sensitive information from a system or make the system out of working via illegal ways, such as eavesdroping, hijacking, denial of service atacks ect.

Table 2.2: Some information security objectives. [60]

Security Objectives	**Explanations**
Privacy or confidentiality	Keeping information secret from all but those who are authorized to see it.
Data integrity	Ensuring information has not been altered by unauthorized or unknown means.
Entity authentication or identification	Corroboration of the identity of an entity (e.g., a person, a computer terminal, a credit card, etc.).
Message authentication	Corroborating the source of information; also known as data origin authentication.
Signature	A means to bind information to an entity
Authorization	Conveyance, to another entity, of official sanction to do or be something.
Validation	A means to provide timeliness of authorization to use or manipulate information or resources.
Access control	Restricting access to resources to privileged entities.
Certification	Endorsement of information by a trusted entity.
Timestamping	Recording the time of creation or existence of information.
Witnessing	Verifying the creation or existence of information by an entity other than the creator.
Receipt	Acknowledgement that information has been received.
Confirmation	Acknowledgement that services has been provided.
Ownership	A means to provide an entity with the legal right to use or transfer a resource to others.
Anonymity	Concealing the identity of an entity involved in some process.
Non-repudiation	Preventing the denial of previous commitments or actions.
Revocation	Retraction of certification or authorization.

2.3 Security Attack

At present, the main factors causing insecure problems are due to the flaws in the design of algorithms, protocols, systems and database. An adversary is able to take advantage of any flaw to launch different types of attack. In [105], security attacks can be classified into passive attacks and active attacks. An "active attack" means an adversary trying to affect their operations or alter system resources. A "passive attack" means an adversary trying to spy information from the system but does not affect system resources.

2.3.1.1 Active attack

An active attack can be launched to modify a data stream or create a fake data stream. It can be categorized into four types: replay, masquerade, modification of messages and denial of service. [4]

A masquerade takes place when one entry pretends to be another authorized or legitimate entry in order to get access to restricted resource or infiltrate the system. It is also known as a spoofing attack. A reply attack means that an adversary captures a data unit and retransmits it to produce an unauthorized effect. Modification of messages usually involves altering, delaying or recording a legitimate message to produce an unauthorized effect. The denial of service attacks come in a variety of forms. There are three main forms of denial of service attack: consumption of scare resource, alteration of configuration information and physical destruction of network components. It aims to decrease the availability of the system.

Attacks are actually quite difficult to be prevented, since there are a lot of potential vulnerabilities in the design of software and network. Fortunately, according to the properties of active attacks, it is possible to detect them and recover information from any disruption or delays caused by them.

2.3.1.2 Passive attack

Passive attack takes place when an adversary eavesdrops, monitors the transmissions between two communication entries to obtain useful information. There are two types of passive attacks: release of message contents and traffic analysis [4].

The release of message contents is referred that an adversary can learn the contents of information between two entries when they are communicating, such as a telephone conversation or an electronic mail message. Traffic analysis is much more difficult than the release of message contents. In order to prevent any adversary from knowing the information, two entries may mask the contents of messages or other information traffic. Therefore, if an adversary captures a message from then, he has no idea what the message means without knowing the way to extract the message. However, a smart adversary might still be able to observe the pattern of the message. For instance, an adversary could still observe the frequency and the length of messages being exchanged. This leaked information might be helpful for the adversary to guess the nature of the communication which was taking place [4].

Passive attacks present the opposite characteristics of active attacks. Since passive attacks do not affect system resources at all, it is very difficult to detect them. One of the most effective way to prevent passive attacks is to encrypt messages. However, it will obviously increase the complexity of communications to some extent.

2.4 Cryptography

As mentioned above, one way to prevent passive attacks is to use encryption scheme to protect information. Encryption/decryption scheme is related to cryptographic algorithms. The definition of cryptography is given in the book [60]: “Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication.”

Cryptography has a very long history. It can be traced to 4000 years ago from its initial and limited use by the Egyptians. Kings and generals communicated with their armies using some simple and basic masking methods to prevent the enemy from knowing sensitive military information. One of the earliest cryptosystems is called shift ciphers, often attributed to Julius Caesar. Encryption is simply shifting each letter in the text by certain places in the alphabet. Decryption is accomplished by shifting letters back. This cryptosystem had been used for centuries. Even as late as 1915, the Caesar cipher was still in use [5]. Later in the sixteenth century, a variation of the shift cipher, named Vigenère cipher, was invented. Even into the twentieth century, this cryptosystem was widely considered to be secure. However, in the 1920s, Friedmen finally developed additional methods to break Vigenère and its related cipher.

Although those classical cryptosystems are too weak to be broken by computers today, they give very good illustrations of several of the important ideas of cryptology. Modern cryptography today draws heavily upon mathematics, computer science and cleverness, but it still uses many same basic principles as classical cryptosystems, such as transposition, substitution, padding or block division. The huge difference is that the proliferation of computers in the mid 20th century called for the demand to protect information in digital form. Also with the help of super computers, modern cryptographic algorithms should be design as complex as possible against all kinds of cryptanalytic attacks.

In modern cryptosystems, there are a number of a number of basic cryptographic primitives used to provide information security. These can mainly be divided into three catalogs: symmetric-key primitives, public-key primitives and unkeyed primitives. Figure 2.1 provides a schematic listing of the primitives considered and how they relate [60]. More details are discussed as follows.

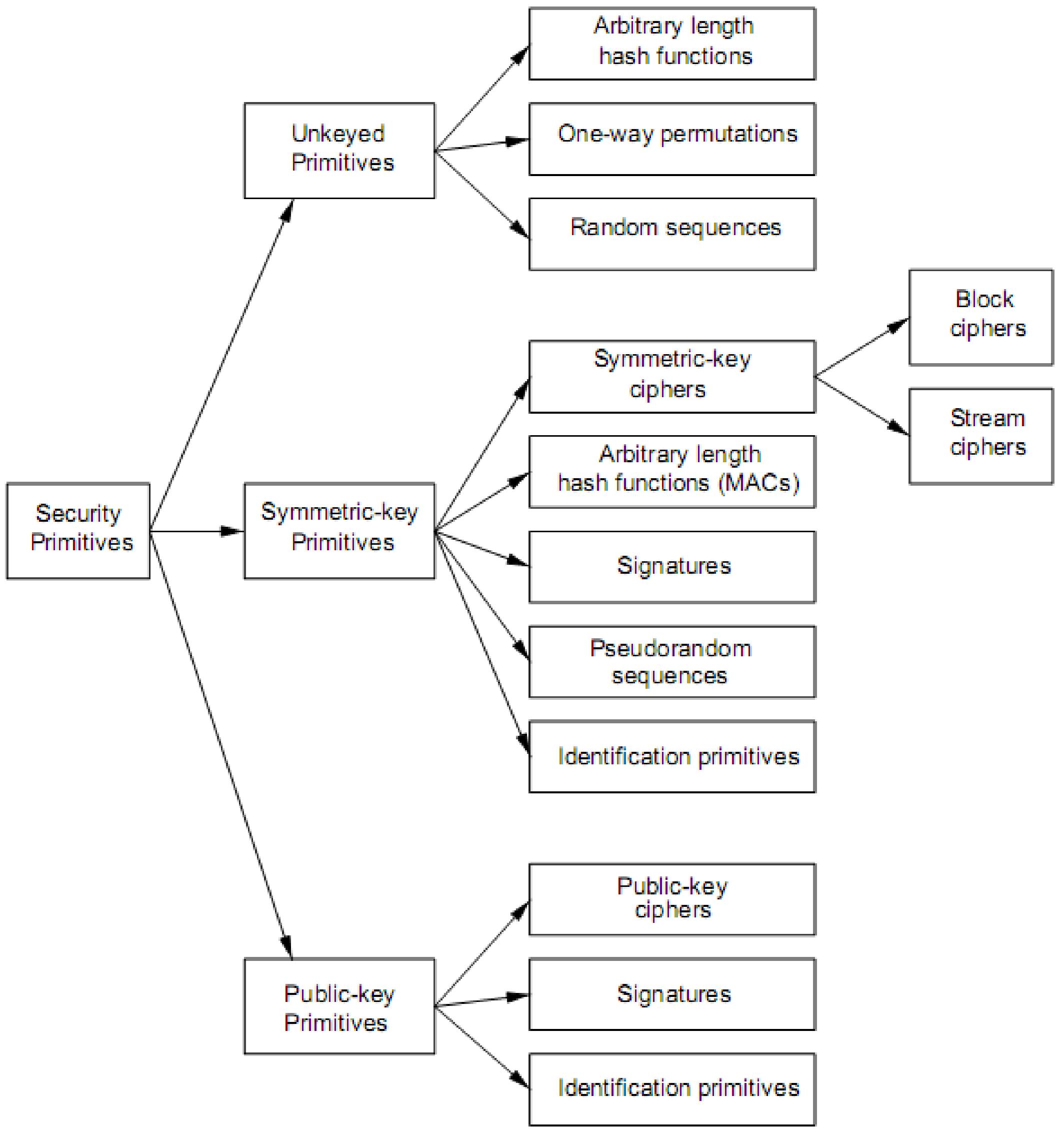


Figure 2.1: A taxonomy of cryptographic primitives. [60]

2.4.1 Basic Concept

Before introducing more details of cryptographic algorithms, some basic terminology and concepts are necessary.

1) Encryption domains

A denotes a finite set named the alphabet of definition. For instance, *A* = {0, 1}, the binary alphabet, is a frequently used alphabet of definition.

M denotes a set named the *message space*. *M* consists of strings of symbols from an alphabet of definition. Elements in *M* are called *plaintext.*

C denotes a set named the *ciphertext space*. *C* also consists of strings of symbols from an alphabet of definition. Elements in *C* are called *ciphertext.*

2) Encryption and decryption transformations

K denotes a set named the key space. Each element $e \in K$ is called a key. Each *e* uniquely determines a bijection from *M* to *C*, defined as an encryption function E_e. Conversely, an element $d \in K$ also uniquely determines a bijection from *C* to *M*, defined as a decryption function. The keys *e* and *d*, sometimes denoted by *(e; d)*, are referred to as a key pair. Note that *e* and *d* could be the same.

An encryption scheme is constructed with a message space *M*, a ciphertext space *C*, a key space *K*, a set of encryption transformations $\{E_e: e \in K\}$, and a corresponding set of decryption transformations $\{D_d: d \in K\}$. [60]

3) Communication participants

An entry is referred to a party who sends, receives or manipulates information. A sender is a legitimate entry who transmits information. A receiver is an entry to whom the corresponding sender intends to transmit information. An adversary is an entry who tries to defeat security strategy and obtain the information transmitted between a sender and a receiver.

4) Achieving confidentiality

In order to achieve confidentiality, an encryption scheme should be used as follows. As described in the book [60], the sender and the receiver should firstly choose or exchange a key pair (*e, d*) in a secure way. If the sender wishes to send a message $m \in M$, he computes $c = E_e(m)$ and transmits it to the receiver. When receiving c, the receiver computes $m = D_d(c)$. Finally, the original message m is recovered, since for each $e \in K$, there is a unique $d \in K$ such that $D_d = E_e\text{-}1$; that is $D_d (E_e (m)) = m$ for all $m \in M$.

5) Cryptology and cryptanalysis

The three names, *cryptography*, *cryptology* and *cryptanalysis*, are often used interchangeably. Cryptology is the all-inclusive term for the study of communication over nonsecure channels [1]. The process of designing systems to do this is called cryptography.

Cryptanalysis deals with breaking cryptographic techniques or information security services. Another word *cryptosystem* is referred to a set of cryptographic primitives used to provide information security services [60].

2.4.2 Symmetric Encryption

Symmetric cryptography uses one private key for both encryption and decryption. That is for the sets of encryption and decryption transformations $\{E_e: e \in K\}$ and $\{D_d: d \in K\}$, $e = d$. Other terms used in the literature are single-key, one-key, private key, and conventional encryption [60]. All of the classical (pre - 1970) cryptosystems are symmetric.

A further division of symmetric cryptosystems is block ciphers and stream ciphers. To use block cipher, the plaintext should be divided into an identical fixed length. Then each block of the plaintext is encrypted respectively. Different from block ciphers, stream ciphers work on individual plaintext digits with a time-varying transformation.

To apply a block cipher to encrypt a plaintext with an arbitrary length, there are several different of modes of operation. A mode of operation is a technique to enhance the effect of a cryptographic algorithm or to adapt the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream [4]. Five commonly used modes of operation are defined by NIST, which are listed in the Table 2.3.

Table 2.3: Block Cipher Modes of Operation [4]

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	• Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed j bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	• General-purpose stream-oriented transmission • Authentication
Output Feedback	Similar to CFB, except that the input to the encryption algorithm is the preceding DES	• Stream-oriented transmission over noisy channel (e.g.,

(OFB)	output.	satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	• General-purpose block-oriented transmission • Useful for high-speed requirements

Symmetric cryptography algorithms are typically fast and suitable for processing large amount of data. The main disadvantage of symmetric algorithms is that sender and receiver have to agree on a same private key in a secure manner prior to their communications.

Symmetric cryptography can provide not only confidentiality, but also a degree of authentication, since data encrypted with one private key cipher cannot be decrypted with any other keys. Therefore, as long as the symmetric key cipher is kept secret by the two parties using it to encrypt communications, each party can be sure that is communicating with the other as long as the decrypted messages continue to make sense.

2.4.3 Asymmetric Encryption

Asymmetric cryptography is also referred to public-key cryptography. Different from symmetric encryption, a pair of keys is required during asymmetric encryption, public key and private key. That is for the sets of encryption and decryption transformations $\{E_e: e \in K\}$ and $\{D_d: d \in K\}$, $e \neq d$. The asymmetric encryption and decryption are described as follows.

An entry Alice has a pair of keys, *e* and *d*. Assume that *e* is Alice's public key and *d* is Alice's private key. In a secure system, it is a computationally infeasible task of computing d given *e*. We define an encryption transformation E_e using public key *e* and a decryption transformation D_d using private key *d*. Another entry Bob wants to send a message *m* to Alice. First, Bob should obtain an authentic copy of Alice's public key. Then he encrypts the message m applying the encryption transformation E_e to obtain the ciphertext $c = E_e(m)$ and send *c* to Alice. For decryption, Alice uses the decryption transformation D_d to recover the original message: $m = D_d(c)$.

As the definition of public-key encryption, the public key need not be kept secret. Therefore, any entry can obtain a copy of Alice's public key. Any ciphertext encrypted using Alice's public key can only be decrypted by Alice. The reason why Bob should get an authentic copy of Alice's public key is that if the public key is not authentic, but transmitted through an unsecure channel, this public key may be manipulated by an adversary. Then Alice will be able to decrypt the cipher to obtain the original message. One way to obtain an authentic copy of public keys is through Certification Authority (CA).

Each wide used asymmetric cryptography algorithm is based on one of the intractability of certain mathematical problems. There are several famous intractable mathematical problems, such as integer factorization problem and discrete logarithm problem. Each of them has one or several typical and widely used cryptography algorithms or standards. For instance, the most famous public-key algorithm RSA is based on integer factorization and one of the most widely used public-key algorithms ECC is based on discrete logarithm problem.

One of the most widely used applications of public-key algorithms is key exchange or key negotiation for symmetric encryptions. Since for symmetric cryptosystems, a sender and a receiver may be hundreds of kilometers apart and have not agreed on a same secure key to use. With public-key algorithms, this problem can be solved amazingly. Another application of public-key algorithms is to achieve authentication. As mentioned above, any message encrypted with Alice's public key, only Alice with her private key can decrypt this ciphertext, which can guarantee Alice's identity.

When comparing symmetric encryption with asymmetric encryption, one of the most distinguished differences is the complexity. Since big number operation is involved in the calculation of public-key encryption and extremely long key is required to achieve a certain level of security, all public-key algorithms are much slower by an order of magnitude than symmetric-key algorithms. Other differences are listed in [60], shown in the Table 2.4.

Table 2.4: Differences between symmetric and asymmetric encryption [60]

Differences	**Symmetric Encryption**	**Asymmetric Encryption**
An extensive history	Long	Short
Data throughput	Fast	Slow
Key length	Relatively short	Long
Generating pseudorandom number	Yes	No
Key management	Poor	Strong
Digital signature	Inefficient	Efficient

2.4.4 Hash and MAC Algorithms

One of the basic components of many cryptographic algorithms is hash function, often informally called a one-way hash function. The definition of hash function is described in the book [60]: “A hash function is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values.” More properties of hash function which should be satisfied are:

- Consider *h* as a hash function. Given a message *m*, then the message digest *h (m)* can be computed efficient and fast.
- Hash function *h* must be a one-way function. That is, Given another message digest *y*, it is computationally infeasible to find an *m’* with *h(m’) = y*.
- Hash function *h* must be collision-free. That is, it is computationally infeasible to find two messages m_1 and m_2 with $h(m_1) = h(m_2)$.

As the definition, hash function implies an unkeyed hash function. At the highest level, hash functions may be divided into two catalogs: unkeyed hash functions and keyed hash functions. In [60], it gives a simplified classification, illustrated in Figure 2.2.

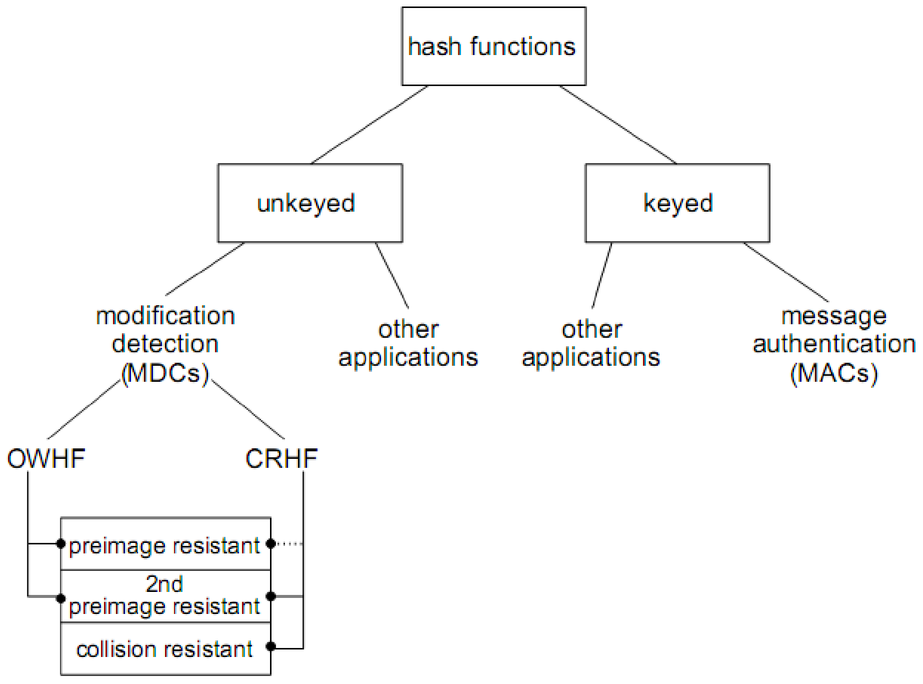


Figure 2.2: Simplified classification of cryptographic hash functions and applications. [60]

In the Figure2.2, MDCs represent modification detection codes. It is a subclass of unkeyed hash functions. It can provide data integrity assurances, since even 1 bit in the input message changes; the whole output will be totally different. Therefore, to verify the integrity of an input message, the receiver can compute the message digest using the same hash function as the sender and compare it with the original message digest provided by the sender. If they are equal, it is confident that the message is not altered during the transmission. MACs represent message authentication codes. Different from MDCs, MACs have two different parameters, a message input and a secret key. The secret key is only shared by the sender and the receiver. Therefore, MACs cannot only provide message integrity, but also authentication, assuring the source of the message.

Another application of hash function is digital signature. Since the length of a digital signature is at least as long as the document being signed, it is much more efficient to sign the hash value of the document rather than the whole document [1]. Through this way, both time and space are saved. However, one important thing should be mentioned that the hash function for the digital signature must be carefully design, especially strong collision-free. Otherwise two messages m_1 and m_2 will have the same hash value, which leads to the same digital signature value.

2.4.5 Cryptanalytic attacks

Previously, we have mentioned several methods of security attacks. However, there are several types of attacks aiming to break encryption schemes, in order to recover plaintext from ciphertext, or even more drastically, named cryptanalytic attacks.

There are four basic types of cryptanalytic attacks that an adversary might be able to use. The differences between these attacks are the amounts of information the adversary can obtain when trying to determine the key.

- Ciphertext only: The adversary only has a copy of a ciphertext. This is the most unsatisfactory case for the adversary to deduce the key of a cryptosystem.
- Known plaintext: The adversary has a copy of a ciphertext and its corresponding plaintext.
- Chosen plaintext: The adversary is able to choose any plaintext and obtain the corresponding ciphertext and trying to use the resulting ciphertext to deduce the secret key of the cryptosystem.
- Chosen ciphertext: The adversary is able to choose any ciphertext and obtain the corresponding plaintext and trying to use the result to deduce the secret key of the cryptosystem.

Most of these types of cryptanalytic attacks also apply to digital signature schemes and message authentication codes.

One of the most famous assumptions in modern cryptography is *Kerckhoffs's principle* [1]. This important principle refers that the security of a cryptosystem should not be based on the obscurity of the algorithms used, but only based on the secret key. It means that the adversary may have knowledge of the algorithms, but if he has not any idea of the secret key, the whole cryptosystem is still considered to be secure enough.

2.5 Communication and Network Security

Previously, we have introduced many basic cryptographic tools, from symmetric algorithms to hash functions. However, only with these tools, we are far from making our communication secure enough. Cryptographic algorithms should be applied based on a certain security protocol or mechanism. A faulty designed security protocol or mechanism with serious flaws, even with the most secure cryptographic algorithm, can be easily broken by an adversary without any cryptanalytic attack. For instance, for a poor design secure protocol, an adversary may simply parse a data packet and apply a reply attack to obtain the important information without knowing the secret key. Therefore, a well designed security protocol or mechanism is vital important.

This part we will introduce several important and widely-used security protocols and mechanism to carry out secure transactions over unsecure channels.

2.5.1 Kerberos

Originally, Kerberos in the Greek and Roman mythology is a many-headed dog guarding the entrance of Hades. Now Kerberos is referred to a trusted third party used for authentication and authorization originally as a part of Project Athena at MIT in early 80s last century.

One of the significant features and design objectives of Kerberos is *Single Sign On (SSO)*, which means that a user only needs to sign in once for credentials and then obtain a Kerberos ticket-granting ticket (TGT) and with TGT the user gains access to all systems. Other design objectives of Kerberos are: i) secure: An eavesdropper should not be able to impersonate a user; ii) Modular and distributed architecture: servers can back up each other for reliable and the system is able to support a large number of clients and servers for scalable [4]. Kerberos is very is suited for large environments, two reasons are: i) No individual computers have to do authentication; ii) Application servers only have to share a secret with the Kerberos server.

Kerberos performs both authentication and authorization. An entire authentication or authorization progress of a Kerberos system is quite complicated. For more details of authentication see the book [4]. There are two versions of Kerberos in common use. Version 4 implementations still exist, using symmetric algorithm DES for encryption. Version 5 corrects some of the security deficiencies of version 4 and has been issued as a proposed Internet Standard [106]. In version 5, new symmetric algorithm AES is supported.

2.5.2 SSL/TLS

Secure Sockets Layer (SSL) was developed by Netscape in order to perform http communications securely. The version 3 was released in 1995 and is the preferred version today. Transport Layer Security (TLS) is a slight modification of SSL version 3 and was released by IETF in 1999. The latest version as of today is TLS 1.2. [1]

SSL is designed to make use of TCP to provide a secure and reliable end-to-end service. Therefore, SSL can be used to secure all TCP connections. SSL is two layers of protocols, rather than a single security protocol. It is illustrated in Figure 2.3.

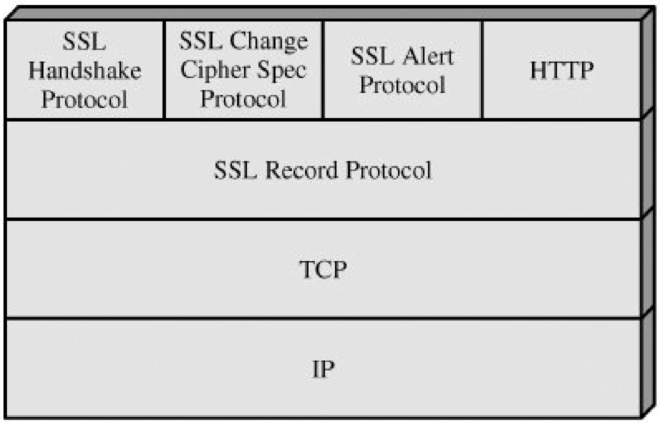


Figure 2.3: SSL Protocol Stack [4]

From the figure, SSL consists of two main components. The first component is SSL Record Protocol, which is responsible for compressing and encrypting data and serves for various higher-layer protocols. It provides both confidentiality and message integrity. A collection of three higher-layer protocols, SSL Handshake Protocol, SSL Change Cipher Spec Protocol, and SSL Alert Protocol, is the second component of SSL. This component is responsible for setting up and maintaining the parameters used by SSL record protocol [1]. Among these protocols, SSL Handshake Protocol is the most complex one. It is not only used for authentication between a server and client, but also for negotiation of encryption and MAC algorithms and secret keys to be used to protect data sent in an SSL record. The Handshake Protocol is used before any application data is transmitted [4]. For more details of SSL/TLS see [1] [4].

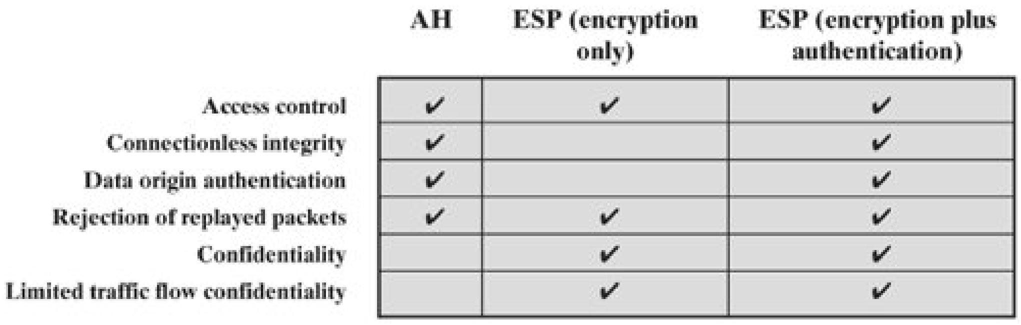
2.5.3 IPSec

Different from SSL/TLS, IPSec can encrypt and/or authenticate all traffic at IP level, not TCP level. Therefore, all distributed applications, including remote logon, client/server, e-mail, file transfer, web access, and so on, can be secured.

The IPSec specification is quite complicated, consisting of a number of documents. Among all these documents, the most important ones are [107], [108], [109] and [110], which are issued in November of 1998.

Two protocols with different headers in IPSec can be used to provide security. One is an authentication protocol with Authentication Header (AH), the other is a combined encryption/authentication protocol with Encapsulating Security Payload (ESP). For ESP, there are two cases: with and without the authentication option. These two protocols can provide different services, which is illustrated in Table 2.5.

Table 2.5: IPSec Services [4]



	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✓	✓	✓
Connectionless integrity	✓		✓
Data origin authentication	✓		✓
Rejection of replayed packets	✓	✓	✓
Confidentiality		✓	✓
Limited traffic flow confidentiality		✓	✓

IPSec also has two modes: Transport Mode and Tunnel Mode. Transport mode provides protection for upper-layer protocols. Tunnel mode provides protection to the entire IP packet. With tunnel mode, a number of hosts on networks behind firewalls may engage in secure communications without implementing IPSec [4].

According to the properties of IPSec, there are several outstanding benefits of IPSec: i) Since IPSec is applied at IP level, it is totally transparent to all above applications. All upper layer software on a user or server system will not be affected. ii) IPSec can be transparent to end users. iii) IPSec can provide security for individual users if needed. [4]

2.5.4 Virtual Private Networks

If an international company has many branches locating in different places in the world, for information security, it is necessary to connect every computer through an underlying local or wide-area network to setup a network which is isolated from other networks. Traffic separation can be obtained in several ways: a) Physical separation using different hardware; b) Temporal separation (separation in time); c) Logical separation, for example, by software as with VLAN; d) Cryptographic separation. The most secure and reliable way is the last one and the solution is Virtual Private Networks (VPN).

Two basic technologies make VPN possible: encryption and tunneling. For confidentiality, VPN encrypts output data and encapsulating it into another packet, such as IP-in-IP encapsulation, for transmission. When this encrypted packet reaches the end of the tunnel, the router responsible for receiving will parse the packet, decrypts the data and forward the inner data packet.

There are three basic types of VPN systems: Host-to-Host VPN, Remote Access VPN and Site-to-Site VPN. For Remote Access VPN, a user needs to connect to a VPN gateway, which authenticates and gives access to authorized resources in the site. For Site-to-Site VPN, a VPN gateway is also involved. The sending VPN gateway will encrypt all outgoing messages and the receiving VPN gateway will do the decryption. Since tunneling technology is used, it is totally transparent to all users, which is a main advantage of VPN.

3 SURVEY OF CRYPTOGRAPHIC ALGORITHMS

This chapter is a survey that covers most of the wide-used cryptographic algorithms, including symmetric ciphers, asymmetric ciphers and MAC algorithms. For each kind of algorithm, we first give a brief introduction about its basic information and then analyze its security and performance. All the analysis is based on related works that have been done before by other researchers. In order to make our survey more convincing and help us to find the most suitable algorithms, we studied as many algorithms as we are able to. For performance analysis, the best method is to implement them in target devices and evaluate the results. However, due to the limitation of time, we cannot implement all studied cryptographic algorithms. Therefore, based on related works, we pick up candidates within much smaller subset for further implementation in our target devices. At this end, according to the analysis, we choose three candidates from symmetric algorithms and MAC algorithms, two candidates from asymmetric algorithms, for the software implementation in ARM platform.

3.1 Symmetric Algorithms

We have previously introduced that symmetric algorithms use only one secret key for both encryption and decryption. Symmetric algorithms are typically fast and suitable for processing large amount of data. There are a number of widely used symmetric algorithms, which is listed and briefly described and analyzed as follows.

3.1.1 Introduction

3.1.1.1 Data Encryption Standard (DES) / 3DES / DES-X/ DESL:

DES is a block cipher, one form of symmetric cryptography algorithms, which was designed by IBM and selected by the National Bureau of Standards (NBS) in the early 70'ies. It has been the standard encryption algorithm for civilian applications for more than 25 years. It has been considered completely insecure due to the short key length.

Since the key length of DES is too short, an improvement to enhance security is to encrypt data using DES more than once. However, double DES encryption (2DES) is also considered insecure due to the meet-in-the-middle attack. Triple DES (3DES) is considered to be temporarily secure enough and still widely used recently.

DES-X is another variant on the DES block cipher intended to increase the complexity of a brute force attack using a technique called key whitening. Another reason for DES-X is that the speed of 3DES is unacceptable in many situations. Therefore, there is a need for an efficient way to strengthen DES.

DESL (DES Lightweight extension) is a new extension to DES and was suggested by A. Poschmann et al. in 2006 as a new alternative for ultra-low-cost encryption. To decrease chip size requirements it uses only one S-Box repeated eight times. It therefore requires 38% fewer transistors than the smallest DES implementation [99].

3.1.1.2 Blowfish / Twofish:

Blowfish was designed by Bruce Schneier in 1993 [103]. Blowfish is still considered to be secure, since there is no effective cryptanalysis found. It also provides a decent encryption performance in software implementation. However, Bruce Schneier himself recommended using the more advanced version - Twofish instead.

Twofish is another block cipher published in 1998 by Counterpane Labs. It was one of the five Advanced Encryption Standard (AES) finalists. However, it was not selected by NIST as AES, since the winner of AES (Rijndael) is considered to have better performance than other finalists in both hardware and software in average. Twofish allows a wide range of tradeoffs between size and speed. It is also designed to be efficient on a wide range of platforms. Although it was not selected as AES, it may still be the suitable choice in our case due to the different platform.

3.1.1.3 Tiny Encryption Algorithm (TEA)/ XTEA / XXTEA

The TEA is a block cipher presented in 1994 [104]. The aim of TEA is to minimize memory footprint and maximize speed. It is a Feistel type cipher that uses operations from mixed (orthogonal) algebraic groups. There are two variants of TEA - extended TEA (XTEA) and Corrected Block TEA (XXTEA), which were designed to correct weaknesses in the original TEA.

3.1.1.4 Rijndael Algorithm (AES):

Rijndael was selected the winner of AES by NIST in 2000 [1]. It is based on a design principle known as a Substitution permutation network. It is fast in both software and hardware. Unlike its predecessor DES, Rijndael does not use a Feistel network.

3.1.1.5 Skipjack Algorithm:

Skipjack was developed by the U.S. National Security Agency (NSA). It is one of the simplest and fastest block cipher algorithms, which is very important for embedded systems. Skipjack or a variant of Skipjack is now used in TinySec , SenSec and MiniSec in Wireless Sensor Networks [45] [44] [46].

3.1.1.6 Scalable Encryption Algorithm (SEA):

The Scalable Encryption Algorithm was proposed by Franccois-Xavier Standaert et al. and is designed for processors with a limited instruction set. The proposed design is parametric in the text, key and processor size, provably secure against linear/differential cryptanalysis, allows efficient combination of encryption/decryption and “on-the-fly" key derivation. Target applications for such routines include any context requiring low-cost encryption and/or authentication [101].

3.1.1.7 HIGHT Algorithm:

HIGHT is another block cipher proposed by Deukjo Hong et al. and provides low-resource hardware implementation, which is proper to ubiquitous computing device such as a sensor in Wireless Sensor Network (WSN) or a RFID tag. HIGHT does not only consist of simple operations to be ultra-light but also has enough security as a good encryption algorithm [100].

3.1.1.8 Other Symmetric Cryptographic Algorithms:

Besides the algorithms mentioned above, there are still a number of other symmetric algorithms. However, we cannot include all symmetric algorithms in our survey due to the time limitation. We will skip the rest and the reasons will be given as follows:

FEAL is no longer considered in our survey, since the inventor of FEAL, Akihiro Shimizu and Shoji Miyaguchi from NTT, noticed that Camellia is strongly recommended to replace FEAL in new applications in the future for efficiency and security.

SAFER++$_{128}$ is not included, since according to [6] there some concerns about certain structural properties of SAFER++$_{128}$ and the low security margin of SAFER++$_{128}$.

MARS is not included. The reason is its high algorithmic complexity, which results in high RAM and ROM usage [7].

Serpent is not included, since it consistently performs poorly in software encryption and decryption due to its large security margins [7]. Therefore, in the competition of AES, supporter of Serpent prioritized security over performance.

SHACAL-2 is a new 256-bit hash function-based block cipher recommended by NESSIE, but has not been studied by NIST or CRYPTREC. Also 256-bit is a little too long as a symmetric algorithm for embedded applications.

Although IDEA, RC5, RC6, MISTY1, KASUMI and Camellia are also considered to be fast, efficient and secure algorithms, as their authors announced, they are all patented. And IDEA has very unbalanced performance of encryption and decryption. For more details see [102]. If these algorithms are used for commercial purpose, some extra fee needs to be paid to authors or certain organizations. Some of them are only free to use in certain projects, such as OpenSSL. From the related work [96], these algorithms cannot provide outstanding and better performance than unpatented algorithms and the cost is always a vital and important issue. Therefore, they are not considered in our study.

Remaining symmetric algorithms are not included due to the time limitation.

3.1.2 Security and Performance Analysis

3.1.2.1 Security Analysis

Security of cryptographic algorithms should be considered with high priority. However, the security of cryptographic algorithms is very difficult to measure. Some basic information related to security, such as key length and number of rounds, is shown in Table 3.1.

Table 3.1: Details of Symmetric Cryptographic Algorithms

Algorithm Name	**Key Size (bit)**	**Block Size (bit)**	**Structure**	**Round**
DES	56	64	Balanced Feistel network	16
3DES	168, 112 or 56	64	Feistel network	48 DES-quivalent rounds
DES-X	184	64	Feistel network	16
Blowfish	32–448 bits in steps of 8 bits; default 128 bits	64	Feistel network	16

Twofish	128, 192 or 256 bits	128	Feistel network	16
TEA, XTEA	128	64	Feistel network	variable; recommended 64 Feistel rounds (32 cycles)
XXTEA	128	arbitrary, at least two words (64)	Unbalanced Feistel Network	depends on the block size; ~52+6*words (6-32 full cycles)
AES (Rijndael)	128, 192 or 256	128	Substitution-permutation network	10, 12 or 14 (depending on key size)
Skipjack	80 bits	64	unbalanced Feistel network	32
HIGHT	128	64	Feistel Network	32

Note: DESL has several modes, values change depending on working mode

SEA has various key size, round depending on parameters

The security of a cryptographic algorithm is related to how difficult it is for adversary to find out the secret key. The most naïve and the simplest way to determine the secret key is a brute force attack. Therefore, the key length is related to how much time an adversary will spend on searching the entire keyspace. However, longer keys cannot guarantee more difficulty for an adversary to determine the key, since the design of the algorithm also plays a critical role in security. Key length is a reference to analysis security of an algorithm, but it is far from enough. Respective security analysis is discussed as follows.

1) DES/3DES/DES-X/DESL:

DES is considered entirely insecure since the key length is too short. The key size is only 56 bits, so the key space size = $2^{56} \approx 7 * 10^{16}$. In January 2000, distributed.net organized idle CPU time of 100,000 computers to search 250G [key/sec]. DES is cracked in 22 hours with a brute force attack.

3DES is considered much more secure than DES due to the longer key size. However, according to [54], the meet-in-the-middle attack on 3DES can reduce its “efficient” key length to 112.

The key size of DESX is increased to 56 + 2$\times$64 = 184 bits. However, the efficient key size is only increased to 56 + 64 – 1 – lb(M) = 119 – lb(M) $\approx$ 119 bits, where M is the number of known plaintext/ciphertext pairs the adversary can obtain, and lb denotes the binary logarithm. According to the research done by Kaliski and Robshaw, DESX actually improves security against differential and linear cryptanalysis, increasing the required number of known or chosen plaintext s to be in excess of 260 [55]. Also it has been proven, in a strong sense, to add much strength against exhaustive key search.

DESL can be used in simple mode, with only 56 bit key size. This implementation is only relevant for the application where short-term security is needed, or where the values protected are relatively low [99]. If a higher security level is needed, the key-whitening can be used. In [99], designer declares that DESL can be resistant to Differential Cryptanalysis and Davis Murphy Attack, Linear Cryptanalysis, and if key whitening is used, then against brute force attack. However, since it is new, no current attack is found on DESL.

2) Blowfish/Twofish:

There is no effective cryptanalysis on the full-round version of Blowfish known publicly as of 2009. In 1996, Serge Vaudenay found a known-plaintext attack requiring 2^{8r+1} known plaintexts to break, where r is the number of rounds. Moreover, he also found a class of weak keys that can be detected and broken by the same attack with only 2^{4r+1} known plaintexts. This attack cannot be used against the regular Blowfish; it assumes knowledge of the key-dependent S-boxes. Vincent Rijmen, in his Ph.D. thesis, introduced a second-order differential attack that can break four rounds and no more. There is no known way to break the full 16 rounds, apart from a brute-force search [59].

Up to now, Twofish is still considered to be secure enough. One famous cryptanalysis on Twofish block cipher published by Shiho Moriai and Yiqun Lisa Yin in 2000 is a truncated differential cryptanalysis of 12- and 16-round version [97]. The paper claimed that a 16-round truncated differential with probability of about $2^{-57.3}$ is discovered. They claimed a good pair of truncated differentials can be found only with roughly 2^{51} chosen plaintexts. However, Twofish is still far from being broken.

3) Tiny Encryption Algorithm (TEA)/ XTEA / XXTEA:

There are several weaknesses in TEA. However, most notably it suffers from equivalent keys—each key is equivalent to three others, which means the effective key size is only 126 bits. [63] Therefore, TEA is a definitely not a smart choice as a cryptographic hash function.

Revisions of TEA, XTEA and XXTEA are designed due to this problem. As of 2004, the best attack reported on XTEA is a related-key differential attack on 27 out of 64 rounds of XTEA, requiring $2^{20.5}$ chosen plaintexts and a time complexity of $2^{115.15}$. XXTEA is considered to be secure enough. However, recently on May 4, 2010, a chosen plaintext attack for XXTEA using about 2^{59} queries and negligible work was announced in [8].

4) AES (Rijndael):

Unlike most other block ciphers, AES has a very neat algebraic description [56]. In 2002, a theoretical attack, termed the "XSL attack", was announced by Nicolas Courtois and Josef Pieprzyk, purporting to show a weakness in the AES algorithm due to its simple description [57]. Since then, other papers have shown that the attack as originally presented is unworkable.

On July 1, 2009, Bruce Schneier presented a related-key attack on the 192-bit and 256-bit versions of AES discovered by Alex Biryukov and Dmitry Khovratovich; the related key attack on the 256-bit version of AES exploits AES' somewhat simple key schedule and has a complexity of 2^{119} [52]. This is a follow-up to an attack discovered earlier in 2009 by Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic, with a complexity of 2^{96} for one out of every 2^{35} keys.

The first attack against a reduced 8-round version of AES-128 was published in November 2009. This known-key distinguishing attack is an improvement of the rebound or the start-from-the-middle attacks for AES-like permutations, which view two consecutive rounds of permutation as the application of a so-called Super-Box. It works on the 8-round version of AES-128, with a computation complexity of 2^{48}, and a memory complexity of 2^{32} [53].

Although, many attack methods on AES appear these years, AES is still considered to be a very secure algorithm, and in June 2003, the US Government announced that AES may be used to protect classified information.

5) Skipjack Algorithm:

The best and most efficient cryptanalysis of Skipjack is announced by Biham, Shamir and Biryukov's attack. They used a new cryptanalytic technique, based on impossible differentials to show that Skipjack reduced from 32 to 31 rounds can be broken by an attack which is faster than exhaustive search. Truncated differentials and later a complementation slide attack were published against all 32 rounds of Skipjack cipher. However, Reichardt and Wagner showed that there are no meaningful truncated differentials for Skipjack with full 32 rounds, providing heuristic evidence that Skipjack may be secure against truncated differential distinguishing attacks [9]. In [96], author concluded that Skipjack with full 32 rounds is secure enough as of today, with a security margin of 2013.

6) SEA Algorithm:

SEA is designed to be resistant to known attacks: Linear and differential cryptanalysis, extensions of linear and differential cryptanalysis, a dedicated attack against a modified version, square attacks, truncated and impossible differentials, interpolation attacks, slide attacks, related-key attacks, and algebraic attacks [101]. Since it is a new algorithm, no practical attack is found on SEA currently.

7) HIGHT Algorithm:

In [100], the authors claim that HIGHT is secure enough for cryptographic applications. It can be resistant to a number of different cryptanalytic attacks, such as differential cryptanalysis, linear cryptanalysis, truncated differential cryptanalysis, impossible differential cryptanalysis, saturation attack, boomerang attack, interpolation, higher order differential attack, slide attacks, related-key attacks, and algebraic attacks [100]. However, in [10], authors analyzed the resistance of HIGHT against impossible differential attacks by mounting new 26-round impossible differential and 31-round related-key impossible differential attacks where the former requires time complexity of $2^{119.53}$ reduced round HIGHT evaluations and proved it is slightly better than exhaustive search.

3.1.2.2 Performance Analysis

As we mentioned before, performance analysis in this part is based on related work. It gives us basic information how these algorithms perform. In the end, it helps us to reduce our candidates to a smaller subset.

1) Performance Comparison of DES, 3DES, Blowfish and AES.

Although, as we introduced previously that DES is considered to be entirely insecure and Blowfish is the previous version of Twofish, it is still necessary for us to know how well these algorithms perform. [64] has given more prospective about the performance of these compared algorithms. They conducted it on two different machines: P-II 266 MHz and P-4 2.4 GHz and implemented them in Java. The final results showed that Blowfish has a very good performance compared to other algorithms. Also it showed that AES has a better performance than 3DES and DES. Amazingly it shows also that 3DES has almost 1/3 throughput of DES, or in other words it needs 3 times more time than DES to process the same amount of data. For more details see [64].

2) Performance Comparison of Skipjack, AES, Twofish, RC5, RC6, MISTY1, KASUMI, Camellia [96].

Although, RC5, RC6, MISTY1, KASUMI, Camellia are not in our list, it is still helpful to have brief idea of their performance. In [96], the authors gave a comprehensive comparison of all these block cipher based on WSN. For more details see [96]. They presented a ranking of these ciphers based on code size, data memory, en/decryption efficiency and key setup efficiency.

According to their evaluation result, Skipjack, either size-optimized or speed-optimized, is on the top in every test issue. Rijndael has median performance among all competitors. Compared with Rijndael, Twofish is better with respect to footprint. All the rest, RC5, RC6, MISTY1, KASUMI and Camellia did not provide more attractive and overwhelming performance than Rijndael, Skipjack and Twofish. Since these algorithms are patented, we will not consider them in our study.

3) Performance Comparison of DES, AES, DES-X, DESL, HIGHT, TEA, XTEA [98].

In [98], authors implemented these algorithms for the 8-bit AVR architecture, which is a modified Harvard architecture. They took MICA Motes, which is a platform for testing query processing techniques over ad-hoc sensor networks, as an adequate target platform and used an ATmega128 or ATmega128L microcontroller as CPU. The ATmega128 (L) is equipped with 128 kbytes of Flash memory and 4 kbytes of SRAM. They implemented them in AVR-Assembly language by themselves, except for the AES, which is an implementation of the Chair for Communication Security at the Ruhr University of Bochum, and SEA, which is an existing implementation in assembly language available. For more details see [98]. Finally, they came to a conclusion of throughput of encryption and decryption of these algorithms, which is shown in Figure 3.1.

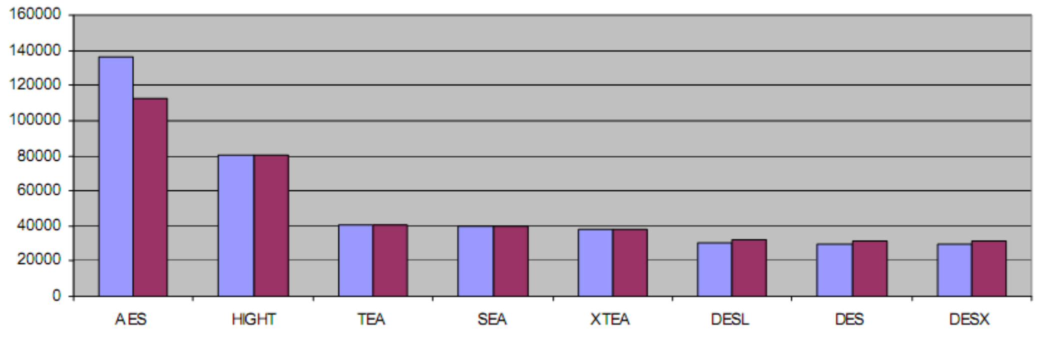


Figure 3.1: Throughput of encryption and decryption of algorithms [98]

From these results, it is obvious that AES has the highest throughput of encryption and decryption. The total throughputs of TEA, XTEA, SEA, DESL, DES and DES-X are very close to each other.

3.1.3 Candidates

As described above, our survey list of symmetric algorithms includes: DES, 3DES, DES-X, DES-L, Blowfish/Twofish, TEA/XTEA/XXTEA, Rijndael, Skipjack, SEA, and HIGHT. Finally, we select Rijndael, Skipjack, XTEA, XXTEA and Twofish as our candidates for further software implementation. The reasons of algorithm selected are given as follows.

Rijndael is selected as our candidate, since it is the Advanced Encryption Standard selected by NIST after extensive security and performance evaluation and adopted as a U.S. Federal Information Processing Standard. It is also recommended by NESSIE as one of the 128 bits block ciphers [6].

Skipjack is selected as our candidate, since it is used as a default block cipher in TinySec [44] and MiniSec [46]. TinySec is an optional part of TinyOS, which is the de facto operating system for Wireless Sensor Networks, while MiniSec is a secure network layer protocol for Wireless Sensor Networks. SenSec, another cryptographic layer for Wireless Sensor Networks, [45] uses a variant of Skipjack, named Skipjack-X. Skipjack-X enhances the key length of original Skipjack to be resistant to brute force attacks. All these imply that Skipjack may be a good choice for embedded applications.

XTEA and XXTEA are selected as our candidates. The most attractive property of XTEA and XXTEA is their extremely simple implementation in software. As described above, XTEA also has decent performance. We add XXTEA as a candidate since it provides more security than XTEA, although the latest attack on XXTEA shows it does not provide intensive entire 128 bits security. We still would like to evaluate the performance on our platform.

Twofish is also selected as one of our candidates. Although Twofish did not win the competition of AES, it is still considered to be efficient on a wide range of platforms. Sano et al. in [47] implemented five AES finalists in a Z80 core with Toshiba’s arithmetic coprocessor. According to their results, Twofish is in the second place of performance, very closed to Rijndael, which is the first. Rijndael can provide good performance in average, but it is proven that Rijndael can perform better than Twofish in every platform. We would like to evaluate the difference between them in our platform.

All the rest are not selected as our candidates. The reason is briefly shown as follows.

DES is not a good candidate, since its key length is too short to be resistant to brute force attack. Although it is efficient in both software and hardware implementation, it cannot prove convincing security. 3DES is not selected as a candidate, since its performance is unacceptable. As analysis above, 3DES has only 1/3 throughput of DES, which cannot satisfy the performance requirement. DES-X is not selected. DES-X is DES compatible. The main motivation for DES-X is to improve on the resistance of DES to exhaustive key search attacks. It is a good choice for old platforms which already have DES in use. For new applications, it is better to use other block ciphers. DES-L, SEA and HIGHT are not selected as our final candidates, since they are brand new. They are not recently widely studied by researchers to show they are secure enough. Therefore, very rare applications use these algorithms as their block cipher schemes. For our practical application in industrial networks, it is recommended to apply well studied and widely used block ciphers.

3.2 MAC Algorithms

Protecting the integrity of data is of utmost importance for industrial automation networks. In many cases not the confidentiality of the data, but its authenticity and integrity are vital important. Therefore, we need to study hash algorithms to guarantee the data integrity. Previously we have mentioned that hash functions can be catalogued into two types: *unkeyed hash functions* and *keyed hash functions*. Of the numerous categories in such a functional classification, two types of hash functions are considered: modification detection codes (MDCs) and message authentication codes (MACs).

For industrial automation networks, not only integrity, but also authentication is another important issue. As we introduced before, MAC has two parameters, a message input and a secret key, shared by the sender and the receiver. According to this property, MAC cannot only provide message integrity, but also authentication. Therefore, MAC is more suitable for our case rather than MDC.

Normally, MAC can be constructed in several different ways. There are two basic and widely used methods. One is MAC based on block cipher. The simplest way is CBC-MAC; the other is based on cryptographic hash functions. The later one can also be classified into two types. One is using traditional hash function to build MAC, such as HMAC-MD5 or HMAC-SHA1. The other is more advanced method, which is using universal hash function to construct MAC, such as UMAC. We will discuss them in details below.

3.2.1 Introduction

3.2.1.1 Block cipher based MAC

There are several MAC algorithms based on block cipher. The details are listed below:

1) CBC-MAC

The most commonly used MAC algorithm based on a block cipher makes use of cipher-block-chaining. A common way to create a MAC is CBC-MAC to encrypt the (padded) message with a block cipher in CBC mode using a fixed IV (e.g. 0). The last ciphertext block is the MAC. (All intermediate blocks discarded.) There are several variations of CBC- MAC:

- Prepend message with the length of message in bytes before MAC computation;
- Truncate the computed MAC if block size is larger than desired MAC size.

2) OMAC/CMAC and several variant of CBC-MAC

There are several variants of CBC-MAC and OMAC is one of them. OMAC is short for one-key MAC. OMAC allows and is secure for messages of any bit length (while the CBC MAC is only secure on messages of one fixed length, and the length must be a multiple of the block length). Officially there are two OMAC algorithms (OMAC1 and OMAC2). OMAC1 is equivalent to CMAC. NIST Special Publication 800-38B Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication has been finalized on May 18, 2005 [87].

Other variants of CBC-MAC are:

a) EMAC: The Encrypted MAC (EMAC), also known as double MAC (DMAC), is a popular variant of the CBC-MAC developed by the RACE project. It is derived from the CBC function by additionally encrypting the output with an independent permutation and secure without any restriction on the message space [12]. The EMAC is also is one of the message authentication codes recommended by NESSIE [6].

b) XCBC: The XCBC scheme was originally proposed by Black and Rogaway in 2000, with the objective of providing a provably secure CBC-MAC scheme which minimizes the number of block cipher encryptions and decryptions [14].

c) RMAC: It was proposed by Jaulmes, Joux and Valette, which is an extension of EMAC. The block cipher algorithms currently approved to be used in RMAC are the AES and triple-DES.

d) TMAC: stands for two-key MAC. It is a refinement of XCBC shown by Black and Rogaway. It was proposed by Kurosawa and Iwata with the goal of reducing the number of required keys from three to two [90].

e) PMAC: PMAC stands for Parallelizable MAC. It was created by Phillip Rogaway in 2002. PMAC is a simple and fully parallelizable block-cipher mode of operation for message authentication. It is deterministic, resembles a standard mode of operation (and not a Carter-Wegman MAC), works for strings of any bit length, and employs a single block-cipher key [85].

3.2.1.2 Hash function based MAC

MAC can be constructed with traditional iterative hash functions (different from new universal hash function). Two famous MAC algorithms are constructed based on hash functions, HMAC and TTMAC.

TTMAC, also known as Two-Track-MAC, was proposed by K.U.Leuven, Belgium and debis AG. It is based on a slightly modified version of the HASH function RIPEMD-160 taking advantage of the two trails used in its compression function [6]. It is in comparison with the MDx-MAC based on RIPEMD-160, much more efficient on short messages and percentage-wise slightly more efficient on long messages [18]. NESSIE recommended it as one of the secure and efficient MAC algorithms.

HMAC is one of the most successful constructions of MAC. It was first published in 1996 by Mihir Bellare, Ran Canetti, and Hugo Krawczyk. HMAC is a variant of NMAC, though NMAC is rarely used today. It does not require the direct loading of the key into the chaining variable of the compression function, but only calls to a hash function. Actually, it is a practical advantage to build MAC through this mechanism, due to the wide availability of free library code for hash functions. Since any iterative hash function can be used in the calculation of HMAC and the security and performance of HMAC is closely related to the inbuilt hash function, it is necessary for us to study traditional hash functions. Here we only list several widely used hash function families.

1) MD5

Message-Digest algorithm 5 (MD5) is a widely used cryptographic hash function with a 128-bit hash value and specified in [111]. MD5 was designed by Ron Rivest in 1991, improved for its previous version MD4. Although MD5 is slightly slower than MD4, it is proven to be more secure. However, MD5 is found not to be collision resistant [19].

2) SHA-1 / SHA -2

The Secure Hash Algorithm (SHA-1), based on MD4, was proposed by the U.S. National Institute for Standards and Technology (NIST) for certain U.S. federal government applications. SHA-1 is also one of the most widely used of hash functions, and is used in several widely-used security applications and protocols, such as Digital Signature Stand (DSS).

The SHA–2 family includes 224, 256, 384 and 512-bit variants. For more details of SHA-2 see [93]. SHA-256 is used to authenticate Debian Linux software packages and in the DKIM message signing standard; SHA-512 is part of a system to authenticate archival video from the International Criminal Tribunal of the Rwandan genocide. SHA-256 and SHA-512 are proposed for use in DNSSEC.

3) HAVAL

HAVAL was proposed by Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry in 1992. It compresses a message of arbitrary length into a digest of 128, 160, 192, 224 or 256 bits. In addition, HAVAL has a parameter that controls the numbers of passes a message block (of 1024 bits) is processed.

4) RIPEMD - 160

RIPEMD-160 is a hash function based on MD4, taking into account knowledge gained in the analysis of MD4, MD5, and RIPEMD. The overall RIPEMD-160 compression function maps 21-word input to5-word output. Each input block is processed in parallel by distinct versions of the compression function. The 160-bit outputs of the separate lines are combined to give a single160-bit output. For more details see [95]. There also exist 128, 256 and 320-bit versions of this algorithm, called RIPEMD-128, RIPEMD-256, and RIPEMD-320.

5) WHIRLPOOL

WHIRLPOOL is a cryptographic hash function, which is designed by Vincent Rijmen and Paulo S. L. M. Barreto. It operates on messages with the length less than 2^{256} bits, and produces a 512-bit long message digest. There are three versions of WHIRLPOOL. The final version was adopted by the International Organization for Standardization (ISO) in the ISO/IEC 10118-3:2004 standard [20].

6) SHA -3

The competition of SHA3 is still ongoing. Now it is on Round 2 and there are 14 candidates. Final result will be present in 2012. For more details see the web page of SHA3 competition at NIST.

3.2.1.3 Carter-Wegman MAC

Both block cipher based MAC and hash function based MAC are widely used, but nowadays it is widely considered that the state-of-the-art MAC algorithms are based on universal hash function. Universal hash functions, first introduced by Carter and Wegman, provide a unique solution to the aforementioned security problems. Roughly speaking, universal hash functions are collections of hash functions that map messages into short output strings such that the collision probability of any given pair of messages is small. A universal hash function family can be used to build an unconditionally secure MAC [81]. Recently there are two famous MAC algorithms based on strongly universal hash function and AES, named Poly1305-AES and UMAC.

1) Poly1305 - AES

Poly1305-AES is one of the state-of-the-art message-authentication codes suitable for a wide variety of applications. Poly1305-AES computes a 16-byte authenticator of a variable-length message, using a 16-byte AES key, a 16-byte additional key, and a 16-byte nonce. The security of Poly1305-AES is very close to the security of AES. There are several useful features of Poly1305-AES, such as extremely high speed, cipher replaceability and etc [91].

2) UMAC

Black et. al. describe a new provably secure message authentication code, called UMAC. UMAC is the fastest message authentication code that has been reported on in the cryptographic literature. First described in 1999, UMAC has undergone significant revisions since its introduction. Large algorithmic changes were made in 2000 to make UMAC faster on short messages. Small algorithmic changes were made in 2004 and a number of UMAC options were eliminated for simplicity [81] [83].

3.2.2 Security and Performance Analysis

3.2.2.1 Block Cipher Based MAC

1) CBC-MAC

In [21], Bellare, Kilian, and Rogaway have already proven that CBC-MAC construction is secure if the underlying block cipher is secure. However, in the end of [21], they emphasized a restriction of the use of CBC-MAC. The restriction is CBC-MAC does not handle variable-length inputs. If an adversary knows two legitimate message-tag pairs, it is extremely easy for him to generate a new legitimate message-tag. Other disadvantages are mandatory serial evaluation and no added resistance to key-search attacks [84].

However, there are still several advantages of CBC – MAC, which are arbitrary message lengths, efficiency, simplicity and familiarity, no re-keying and proven security. CBC-MAC is very efficient in some cases if block cipher is already used, since it is not necessary to build additional hash functions. However, it is considered that the speed of CBC-MAC is slower than HMAC in most cases.

2) Variants of CBC-MAC:

- *EMAC*: In [12], Erez Petrank and Charles Rackoff have proven that EMAC is secure. They mentioned that if there is an attack against it, then an attack with comparable parameters can also be set on the underlying block cipher. It means that the security of EMAC is proven on the assumption that the underlying block cipher is pseudo-random. For performance, Erez Petrank and Charles Rackoff also proved that there is almost no additional cost for EMAC on that of using CBC MAC to provide a secure solution for authenticating variable-length messages. Also in [6], NESSIE analyzed that the performance and key-agility of EMAC are reasonable. They mentioned that since the block length is smaller compared to the schemes based on a hash function, then EMAC is preferable for short messages.

- *XCBC*: XCBC is initiated by the important lemma in the literature. With this lemma, we can generate two computationally different pseudorandom permutations out of one secret pseudorandom permutation and we can prove the proposed construction is secure. However, in [24] the author introduced forgery attacks and key recovery attacks on XCBC. Another disadvantage of XCBC is that three keys are involved in the calculation of XCBC.

- *RMAC*: In [16] [22], researchers have analyzed the security of RMAC. In [22], NESSIE mentioned that the main advantage of the randomised variant RMAC is that it offers improved resistance against attacks that are based on internal collisions. On the other hand RMAC needs stronger assumptions for its security proof; for instance, the underlying block cipher must be secure against related-key attacks. As the same meaning, in [16], authors present a serious attack against RMAC using 3DES. They also present generic attacks against RMAC using any block cipher but with certain parameters. This attack is able to find one of the two keys in the system faster than by an exhaustive search. Also in [23], the author introduced trivial key recovery attack on the RMAC with about 2^n computations.

- *TMAC*: TMAC is a refinement of XCBC. In XCBC, three keys with total length (k + 2n) bits are used, where k is the key length of the underlying block cipher and n is its block length. In TMAC, total key length reduces to (k + n) bits, since TMAC just changes the third key K_3 in XCBC with $K2 \times x$. The proposers of TMAC proved that it is a variable length pseudorandom function with fixed length by assuming that the underlying block cipher is a pseudorandom permutation. However, in [23] [24] [25], all authors present key recover attacks against TMAC, they also gave the suggestion to improve the security of TMAC. For performance, inventor of TMAC declared several advantages of TMAC, such as efficiency, since TMAC uses max block cipher calls, no re-keying, backwards compatibility and simplicity.

- *PMAC*: The inventor of PMAC announced in [85] that PMAC is proven to be secure, as long as the underlying block ciphers meet a standard cryptographic assumption. However, in [25], authors present forgery and Key Recovery Attacks on PMAC and mentioned that PMAC have no significant advantage in comparison with other well-established MAC schemes. For performance, the inventor introduced in [85] that key setup of PMAC is very cheap: one block-cipher call and, if desired, a few XORs and conditional (128-bit) shifts; PMAC has a small footprint, even a memory-minimal implementation doesn't give up much speed. The biggest advantage of PMAC is the high efficiency in parallel environment. However, in a serial environment, PMAC is about as efficient as CBC MAC. In [86], the author shows that PMAC-AES128 is even 8% slower than CBCMAC-AES128 in a serial environment.

- *CMAC*: CMAC is one of the versions of OMAC. Different from XCBC and TMAC, only one key is needed in CMAC. The saving of the key length makes the security proof of CMAC substantially harder than those of XCBC and TMAC [89]. Proposer of CMAC proved that OMAC is secure, where the security analysis is in the concrete-security paradigm. In [24], Mitchell presents a key recover attack against CMAC. However, just soon after that, the proposer of CMAC immediately announced that Mitchell's result was incorrect [26]. After analysis of [24], they announced that Mitchell's claims do not give any information except for trivial and expected attacks, do not break the security bound of CMAC and do not find any "significant weakness" in CMAC. Therefore, CMAC is still secure until now.

- *Comparison of RMAC, EMAC, XCBC, TMAC and CMAC* [88]*:* In [88], Tetsu Iwata presented an entire comparison of these MAC algorithms. He showed that for security, EMAC, XCBC, TMAC and CMAC are better than RMAC. There is no significant difference among EMAC, XCBC, TMAC and CMAC in security. For key length, he showed CMAC gives the best performance, since CMAC is still as secure as EMAC, XCBC, and TMAC despite its optimal key length. After comparison in number of key schedulings, number of block cipher invocations and number of block cipher invocations during the pre-processing time, he made conclusion that CMAC gives the best performance and trade of in efficiency.

3.2.2.2 Hash Function Based MAC

1) MAC Algorithms:

- *TTMAC*: TTMAC stands for Two-Traces MAC algorithm. TTMAC is one of the recommended MAC algorithms by NESSIE. For security, since TTMAC is based on RIPEND-160, it takes advantage of the structure of RIPEND, which consists of two parallel trails. Since two trails are used, the proposer of TTMAC showed that there is impossible for an adversary to do a bijective operation, where he can choose the bijective. Another technology they used to guarantee the transformation of TTMAC to be a one-way function is that a feedback is used to counter a straightforward inverse operation. They also proved that TTMAC is resistant to general attacks, such as key searching, guessing the MAC and internal collision. NESSIE also analyzed the security of TTMAC. NESSIE declared that TTMAC has the highest security level of the MAC primitives considered by NESSIE [6]. They also mentioned that the security can be proven on the assumption that the underlying compression function is pseudo-random. However, the decision made by NESSIE was published in 2003. In 2004, the researcher Wang Xiaoyun had found the collision in RIPEND [94]. Although there are still no attacks published against TTMAC, we still suspect the security of TTMAC. For performance, NESSIE listed two advantages of TTMAC: especially efficient in the case of short messages and having optimal key-agility [6].

- *HMAC*: In [17], Bellare et al. gave a theoretical support for the security of HMAC. HMAC can be considered to be a particular case of NMAC, so the security of HMAC is closely related to the security of NMAC. According to the analysis of NMAC and the difference between NMAC and HMAC, he proved that the security of HMAC is based on the security of "built-in" compressing function. He also mentioned that two particular parameters, opad and ipad, are very important, since they provide computational independence between the two derived keys [17]. To the end, he proved that only one key used in HMAC does not provide less security than using two independent keys against exhaustive key searching attacks. NESSIE also proved the security of HMAC in [22]. In [34], authors presented distinguishing and forgery attacks against HMAC when HMAC employs hash functions that have slow difference propagations. For performance, how fast HMAC can achieve is essentially related to the performance of the underlying hash function [17]. However, one disadvantage of HMAC is that it is not very efficient short messages. For instance, when the length of an original message is less than one block length, HMAC still calls the built-in hash function for twice, which is significant inefficient.

2) Hash Functions:

Since the security and performance of HMAC are closely related to hash functions in use, we evaluate several commonly used hash functions.

- *MD5*: In 2004, collisions are found by researcher Wang Xiaoyun which makes MD5 no longer secure [19]. Years after that more and more attacks on MD5 were proposedd. MD5 now is weaker than weak. The recommendation of NIST is not to use HMAC-MD5 as MAC any more.

- *SHA-1 / SHA-2*: The hash-value of SHA-1 is 160 bits and five 32-bit chaining variables are used. SHA-1 was considered robust enough against brute-force attacks. However, in 2005, Chinese cryptographer Xiaoyun Wang found collision-finding attacks that require only 2^{63} operations, rather than the 2^{80} operations of the birthday attack. Such an attack is feasible for a very well-funded adversary. Therefore, in March 2006, the Policy on hash functions was published: “Federal agencies should stop using SHA-1 for digital signatures, digital time stamping and other applications that require collision resistance as soon as practical, and must use the SHA-2 family of hash functions for these applications after 2010.” However, we should notice that this attack on SHA-1 is more harmful to digital signature, but for MAC, this attack may make less sense. However, in [35], the authors presented distinguishing, forgery, and partial key recovery attacks on HMAC using collisions a reduced version of SHA-1. Therefore, for security consideration, new applications based on HMAC concerning security more than performance, HMAC-SHA2 should be considered instead of HMAC-SHA1.

  Although collisions were found in SHA-1, SHA-2 family is not affected. Until now, no efficient attacks on full SHA-2 family are proposed, but still several security analysis on SHA-2 were published. Yoshida and Biryukov showed at SAC 2005 a pseudo-collision for a simplified variant of SHA-256 (up to 34 steps). In [27], Somitra and Palash proposed improved attacks against 22, 23 and 24-step SHA-2 family using a local collision given by Sanadhya and Sarkar (SS) at ACISP ’08. However, these attacks do not affect full SHA-2 family. Therefore, SHA-2 is still considered to be secure. For performance, in [6], NESSIE wrote that the performance of SHA-2 family is acceptable, SHA-512 and SHA-384 being a big faster than Whirlpool on most platforms. SHA-256 is about twice faster on most platforms.

- *HAVAL*: Collisions were found in HAVAL in [19]. In 2006, cryptographer Wang Xiaoyun again gave the full key-recovery attacks on the HMAC/NMAC instantiated with 3 and 4-Pass HAVAL [28], which made HMAC-HAVAL totally insecure and it should not be used any longer.

- *RIPEND-160*: In 2004, collisions were found in RIPEND by cryptographer Wang Xiaoyun [19]. And in 2007, in [28], the authors showed the first cryptanalytic attacks on the last three rounds of RIPEMD-128. Since RIPEND-160 is an improved version of RIPEND and RIPEND-128, to our knowledge, there is no efficient attack against RIPEND-160 until now. For performance, in [95], it is shown that RIPEND-160 implemented both in 80x86 assembly language and C language and optimized for the Pentium processor is still slower than SHA-1.

- *WHIRLPOOL*: The proposer of WHIRLPOOL declared it is secure. He claimed WHIRLPOOL is resistant to differential attacks and attacks against the internal block cipher [31]. NESSIE also claimed WHIRLPOOL is a collision-resistant hash function. They claimed that the best known attack on WHIRLPOOL finds non-random properties when the compression function is reduced to six rounds or less (out of ten) [6]. However, in [32], authors presented a distinguishing attack on the full compression function of Whirlpool by improving the rebound attack on reduced Whirlpool with two techniques. For performance, WHIRLPOOL is not that fast. NESSIE also claimed that on most platforms it is a bit slower than SHA-512.

3.2.2.3 Cater – Wegman MAC

1) Poly1305-AES:

In [91], the proposer of Poly1305-AES proved the security of it. He claimed that the security of Poly1305-AES is very close to the security of AES. He proved that it can guarantee that the only way to break Poly1305-AES is to break AES, which is very secure. Even if an adversary is able to get all the authenticated messages, to check whether or not the receiver accepts a forgery and to affect the sender's choice of messages, Poly1305-AES is still very secure. However, the user of Poly1305-AES should be responsible to keep the secret key unpredictable and never use the same nonce for two different messages. Otherwise, Poly1305-AES cannot guarantee its security any longer.

For performance, the proposer declared in [91] that Poly1305-AES can achieve an extremely high speed, for example, fewer than 3.1L + 780 Athlon cycles for an L-byte message and 1000 keys can be handled simultaneously without cache misses. There is a comparison of Poly127-AES, which is a former version of Poly1305-AES, and SHA1, shown in the next part.

2) UMAC:

The security of UMAC is based on the security of its underlying cryptographic functions: the key-derivation function (KDF) and the pad-derivation function (PDF). These functions use a block cipher. The default block cipher is AES, which is considered to be secure. Another important property of UMAC is using UHASH function as the core technology. The UHASH function does not depend on cryptographic assumption, which means the strength of UHASH is guaranteed regardless of advances in cryptanalysis [33]. The analysis in [33] also shows that UMAC is very secure, resistant against several types of cryptographic attacks, such as reply attacks or side-channel attacks.

For performance, in [92], a comparison of UMAC performance and Poly127 and SHA1 is given. The performances were measured on 2.7GHz Pentium 4 running Red Hat Linux. The SHA-1 speeds were acquired by using the command "openssl speed sha1". Since theoretically the speed of HMAC-SHA1 should never be faster than SHA-1, SHA-1 can be used as a reference. The result shows that UMAC32, UMAC64 and UMAC96 are all much faster than HMAC-SHA1. Hash127-AES, also called Poly127-AES, a former version of Poly1305-AES, has a similar performance of UMAC, since they are both based on strong universal hash functions. In [36], the performance of UMAC is also measured by NESSIE. They claimed that for message authentication, UMAC is very fast on the PCs, slow on the Sun. However, for key setup stage, UMAC is exceedingly slow.

3.2.3 Candidates

As described above, there are three types of MAC algorithms in our survey list. For block cipher based MAC, there are original CBC-MAC, EMAC, XCBC, RMAC, TMAC, PMAC, CMAC (OMAC); for hash function based MAC, there are TTMAC and HMAC; for Carter-Wegman MAC, there are Poly1305-AES and UMAC. Since HMAC can adopt any hash functions, we have studied several hash functions, they are MD5, SHA-1, SHA-256, HAVAL, RIPEND-160, WHIRLPOOL. It is necessary to evaluate all these three types of MAC algorithms. We choose CMAC from block cipher based MAC algorithms, HMAC- SHA1 and HMAC-SHA2 from hash function based MAC algorithms and UMAC from Carter-Wegman MAC algorithms. The reasons of our choice are given as follows.

We select CMAC as a candidate, since it is NIST Special Publication 800-38B Recommendation for Block Cipher Modes of Operation in 2005. As we pointed out, the comparison between CMAC, EMAC, XCBC, RMAC and TMAC done by Tetsu Iwata in [88] shows that CMAC is the most efficient block cipher based MAC among them. Therefore, we select CMAC as our candidate from block cipher based MAC instead of EMAC. Another PMAC is proven to be very efficient in parallel environment, but in our case we cannot obtain any advantage from it.

We select HMAC-SHA1 and HMAC-SHA256 as our candidates. HMAC-SHA1 is widely applied in a number of projects and security protocols today, such as IPSec and TLS. We add HMAC-SHA256, one member of SHA-2 family, as one candidate due to the security consideration. Although there is no efficient attack proposed on HMAC-SHA1, the security of HMAC-SHA1 is still suspected, since as we introduced before that partial key recovers attacks on the reduced version of HMAC-SHA1 was proposed [35]. Therefore, NIST also recommended using HMAC- SHA256 for security consideration. We do not select MD5 and HAVAL as hash functions, since HMAC-MD5 and HMAC-HAVAL are almost broken. We do not select RIPEND-160, since the reduced version of RIPEND-160 is also insecure as we analyzed before. It is unnecessary for us to choose RIPEND-160, since we have already selected SHA-1. We do not select WHIRLPOOL hash function, since the message digest of WHIRLPOOL is 512 bits long, which may perform very slow in our platform. According to the results from NESSIE, HMAC-WHIRLPOOL is slightly slower than HMAC-SHA512, even HMAC-SHA384 [6]. Although security is vital important in our application, yet we still need to consider the performance of MAC algorithms.

We choose UMAC as one of our candidates, since UMAC is considered to be state-of-the-art MAC algorithm. UMAC is analyzed by NESSIE that it is by far the fastest of the MAC primitives [6]. Although Poly1305-AES is also based on universal hash functions and considered to be very efficient, we only need to choose one of them as our candidate

3.3 Asymmetric Algorithms

Previously, we have introduced that asymmetric algorithms require a pair of keys for encryption and decryption. Each entity can only keep its private key secret and make its public key known to every other entry. Asymmetric algorithms can be used to exchange key or key negotiation for symmetric encryption and achieve authentication, such as digital signature. Each widely used asymmetric cryptography algorithm is based on one of the intractability of certain mathematical problems. There are several famous intractable mathematical problems, such as integer factorization problem and discrete logarithm problem. Each of them has one or several typical and widely used cryptography algorithms or standards. Here we only include some of them according to related works in our survey and skip the rest.

3.3.1 Introduction

3.3.1.1 RSA Public-key Algorithm

The RSA encryption algorithm is named after Rivest, Shamir and Adleman. It is the most widely used public-key cryptosystem. It is based on the intractability of the integer factorization and may be used to provide both secrecy and digital signatures.

3.3.1.2 Rabin Public-key Algorithm

Rabin public-key algorithm was first introduced by Michael O. Rabin in 1979. Similar to RSA, Rabin is also based on the factorization problem of large numbers. It was the first example of a provably secure public key encryption scheme – the problem faced by a passive adversary of recovering plaintext from some given ciphertext is computationally equivalent to factoring. The disadvantage of Rabin cryptosystem is also quite obvious that each output of the Rabin function can be generated by any of four possible inputs, which means that if each output is a ciphertext, extra complexity is required on decryption to identify which of the four possible inputs was the true plaintext.

3.3.1.3 ElGamal Algorithm

The ElGamal public-key encryption is based on the Diffie-Hellman key agreement. Its security is based on the intractability of the discrete logarithm problem and the Diffie-Hellman problem. It was described by Taher Elgamal in 1985. Two kinds of ElGamal public-key algorithms are widely used, basic ElGamal and generalized ElGamal.

3.3.1.4 Elliptic curve cryptography (ECC)

Elliptic curve was first introduced by Miller and Koblitz in the mid-1980s into cryptography and then Lenstra showed how to use it to factor integers [1]. Nowadays, ECC has more attentions from people due to much smaller key size than RSA with the same level of security. It results in faster computation, less memory footprint and power consumption, which is quite useful for embedded systems and mobile devices. ECC includes many different types of primitives, such as Elliptic Curve Digital Signature Algorithm (ECDSA) for signatures and Elliptic Curve Diffie-Hellman (ECDH) for key agreement.

3.3.1.5 NTRU Algorithm

NTRU cryptosystem was proposed by Joseph H. Silverman, Jeffrey Hoffstein and Jill Pipher at Brown University. The encryption of NTRU is using a mixing system based on polynomial algebra and reduction modulo two numbers. These two numbers cannot be prime numbers. The decryption of NTRU is using an unmixing system based on elementary probability theory. The advantages of NTRU system are easy key generation, fast encryption/decryption speed and decent memory requirement [37].

3.3.1.6 Hyperelliptic Curve Cryptography (HECC)

Hyperelliptic Curves is first suggested by Koblitz in 1988. It is based on using the jacobian of a hyperelliptic curve defined over a finite field. One of the most obvious advantages of HECC is that the operand size for HECC is at least a factor of two smaller than the one of ECC, which makes HECC more attractive than ECC in resource constrained platforms.

3.3.1.7 Other Important Public Key Algorithms.

McEliece Algorithm: McEliece Algorithm was developed in 1978 by Robert McEliece and is based on error-correcting codes. McEliece Algorithm is considered extremely secure. Even Quantum computers do not seem to give any significant improvements in attacking code-based systems, beyond the generic improvements possible with Grover's algorithm [77].

However, it is not included in our survey, since the drawback of this algorithm is also quite obvious that its public key is also extremely large, which is not suitable for embedded and resource constrained applications.

Knapsack algorithm: Knapsack public-key encryption schemes are based on the subset sum problem. There are many variations of Knapsack algorithms, but Knapsack most, including the original - the Merkle-Hellman knapsack encryption scheme, have been demonstrated to be insecure [60]. Only the Chor-Rivest knapsack scheme is considered as a notable exception. However, a major drawback of the Chor-Rivest scheme is also that the public key is fairly large, which is also impractical in embedded systems. Therefore, it is not included in our survey.

All the rest of asymmetric algorithms will not be included in our list due to the time limitation.

3.3.2 Security and Performance Analysis

3.3.2.1 RSA Public-key Algorithm

1) Security:

The adversary who wants to attack RSA cryptosystem is facing the RSA problem (RSAP) and integer factoring problem. Today one can factor N with approximately 650 bits using thousands of cooperating computers and sophisticated algorithms. Therefore, the recommended key size should be 1024-2046 bits depending on the security level of applications.

There are several security threats depending on the usage of RSA for encryption. There are [60]:

i) Small encryption exponent *e*.
ii) Forward Search Attack
iii) Small Decryption Exponent
iv) Multiplicative Properties
v) Timing Attack
vi) Common Modulus Attack

A number of cryptanalysis and attacks were presented against RSA. For instance, Martin presented a new attack RSA–CRT employing Montgomery exponentiation even with the lack of the chosen plaintext condition [40].

2) Performance:

Since RSA is based on arithmetic modulo of large numbers, the speed of a RSA system depends on the large number calculation speed. RSA is much slower than other symmetric algorithms [67] .The traditional method for encryption and decryption is considered to be very inefficient. A lot of related works to enhance the speed of RSA system have been proposed these years. Some common methods are listed as follows to improve the encryption and decryption efficiency of RSA [67].

i) RSA with CRT
ii) Batch RSA
iii) MultiPrime RSA
iv) MultiPower RSA
v) Rebalanced RSA
vi) RPrime RSA

a) SW performance on common PC

According to [68], the author has done the performance comparison of all above variants. His measurements were done conducted on an AMD Athlon; Win XP and Linux platform, with 256 MB of RAM and using C language with GNU MP [58] (library GMP). For more details see [68]. The result shows that RSA is rather slow, even with enhancement and running on stationary common PC.

b) SW performance on embedded system;

As shown above, both computation and memory footprint are unacceptable to embedded system, since the CPU speed and the size of memory are much smaller and limited in embedded system. Although some implementations of RSA have been done in DSP or SoC, the author in [71] shows that ECC algorithm has much better performance in embedded system than RSA does.

3.3.2.2 Rabin Public-key Algorithm

1) Security

For a passive adversary who wants to recover the plaintext from the corresponding ciphertext in Rabin cryptosystem, he will face the SQROOT problem [60]. This problem is that factoring *n* and computing square roots modulo *n* are computationally equivalent. Therefore, assuming that factoring n is computationally intractable, Rabin cryptosystem is considered a provably secure public key encryption scheme to a passive adversary.

However, Rabin cryptosystem is very vulnerable to a chosen ciphertext attack. For more details see [60]. Therefore, it is not adviced to use Rabin cryptosystem in an original way. Usually, adding appropriate redundancy prior to encryption would prevent the system from a chosen ciphertext attack.

2) Performance

According to [11], Rabin encryption is an extremely fast operation as it only involves a single modular squaring. By comparison, RSA encryption with $e = 3$ takes one modular multiplication and one modular squaring. Rabin decryption is much slower than encryption, but comparable in speed to RSA decryption. The reason for slow decryption in Rabin cryptosystem is that each output of the Rabin decryption function generates the correct result in addition to three false ones. In order to identify the correct one out of four, extra complexity is required.

3.3.2.3 ElGamal Algorithm

1) Security

The difficulty of breaking the ElGamal cryptosystem is equivalent to solving the Diffie-Hellman problem [60]. Actually, ElGamal encryption can be simply considered as comprising a Diffie-Hellman key exchange to determine a session key and encrypting the message by multiplication with that session key. Therefore, the security of the ElGamal cryptosystem is based on the discrete logarithm problem.

Given the latest progress on the discrete logarithm problem [60], a 512-bit modulus p provides only marginal security from concerted attack. As of 1996, a modulus p of at least 768 bits is recommended. For long-term security, 1024-bit or larger modulus should be used.

2) Performance

Encryption under ElGamal requires two exponentiations; however, these exponentiations are independent of the message and can be computed ahead of time if need be. Decryption only requires one exponentiation. But as author in [76] pointed out, the computation of ElGamal is still more intensive (in terms of processor time) than RSA. Therefore, for embedded systems, ElGamal is not the best choice.

3.3.2.4 Elliptic curve cryptography (ECC)

1) Security

ECC is based on the algebraic structure of elliptic curves over finite fields. It is assumed that finding the discrete logarithm of an elliptic curve element is computationally unfeasible. It is well known for its small key size and computational efficiency, while preserving the same security level as the standard methods, such as RSA. Therefore, it has been incorporated into two important public-key cryptography standards, FIPS 186-2 [NIST00] and IEEE-P1363 [IEEE1363-00]. According to these two standards, ECC can be used over prime fields GF (p) and binary fields GF (2^m). In order to prevent known attack on ECC, a series of recommended curves with well-studied properties can be found in these standards [70].

To our knowledge, there is still no efficient attack proposed on the discrete logarithm problem for elliptic curves. However, there are still some attack methods, which may be security threats against ECC. The pohlig – Hellman attack may work in some situations. The Baby Step, Giant Step attack still works on ECC, but it requires too much memory to be practical in most situations [1]. In [73] [74], authors listed other potential attacks against ECC.

2) Performance

As described before, ECC can be basically used in either a prime field *GF (p)* or a binary field *GF (2^m)*, although other related work also has done in binary composite fields and prime extension fields. Here we just skip these two fields. Gura et al. claimed in [71] that binary polynomial field arithmetic is insufficiently supported by current microprocessors and would thus lead to lower performance. In [75], Michael Brown et al. proved it by software implementation over both prime and binary field and made a comparison of performance in both two fields on a Pentium II 400MHz workstation. Their result shows that for software implementation, ECC used over prime field has better performance than over binary fields for software implementation.

In order to prove ECC is an efficient and well performed public-key algorithm, a number of comparisons with other public-key algorithms have been done by many researchers. Gura in [71] et al. compared the performance of ECC with RSA in an embedded system, Atmel ATmega128 at 8MHz and CC1010 at 14.75MHz in assemble code. In order to make the result more convincing, they implemented both algorithms in an optimized way. For RSA, they used Chinese Remainder Theorem (CRT), Montgomery Multiplication and optimized squaring to accelerate the speed of the RSA computation. For ECC, they also used some techniques, such as Curve-Specific Optimizations, to accelerate ECC. According to their results, ECC has better performance in both computation time and memory footprint [71]. Jens-Peter Kaps in his doctor dissertation [81] compared the performance of ECC with Rabin in hardware implementation. His result showed that at the same security level, Rabin has the same memory footprint as ECC, but for encryption, Rabin has better performance than ECC. However, he did not compare the performance in decryption, which is the weakness for Rabin cryptosystem.

3.3.2.5 NTRU Algorithm

1) Security

Different from other asymmetric algorithms, the security of NTRU algorithm is based on the interaction of the polynomial mixing system and the difficulty of finding the shortest vectors in lattices. Joseph et al. also proved the security of NTRU cryptosystem. They claimed the NTRU cryptosystem is secure, unless there is a new breakthrough in lattice reduction. They also proved that it is resistant to brute force attacks, meets-in-the-middle attacks, multiple transmission attacks and lattice based attacks. An exhaustive list of papers related to the security of NTRU can be found in [86].

2) Performance

NTRU encryption is considered to be highly efficient and particularly suitable for embedded systems, while providing a level of security comparable to that of other established schemes, in particular RSA. In [81], author even compared NTRU with ECC in low power devices with hardware implementation. For more details see [81]. The result shows that NTRU is 1.5 times faster than ECC and only has 1/7 memory footprint than ECC at the same level of security in hardware implementation. The comparison in software implementation is shown in the website of NTRU Cryptosystems, Inc. They declare that in the Palm platform, NTRU still has better performance than ECC.

3.3.2.6 Hyperelliptic Curve Cryptography (HECC)

1) Security

An adversary who wants to attack HECC cryptosystems is also facing the Discrete Logarithm Problem (DLP). Against Hyperelliptic Curve Discrete Logarithm Problem (HEDLP), two types of attacks are quite more efficient than transitional ones, known as index-calculus attacks and Weil descent attacks [39]. The genus of hyperelliptic curve can affect the security of a HECC cryptosystem seriously. In [43], Pierrick Gaudry et al. described their breaking of a HECC cryptosystem based on a curve of genus 6 and proved that index-calculus attacks is faster than the Rho method for genus greater than 4. Therefore, most of HECC implementations only use binary fields and curves of genus 2 or 3 and certain hyperelliptic curves should be definitely avoided.

2) Performance

If an appropriate hyperelliptic curve is chosen, the operand size for HECC is much smaller than the one of ECC. For instance, if genus is chosen as 2, lower bound for HECC is around 80 bits, while for ECC at the security level, the key size is at least 160 bits. In [78] and [79], authors have implemented HECC in embedded systems using several types of microprocessor. The results show that optimized HECC has better performance than ECC.

3.3.3 Candidates

Up to now, we have introduced and analyzed asymmetric algorithms including RSA, Rabin, ElGamal, ECC, NTRU and HECC. Finally, we choose RSA and ECC as our candidates for further evaluation. The reasons are explained as follows.

We choose RSA as a candidate, since RSA is the most widely used public-key algorithms for decades. It is recommended by NESSIE for both asymmetric encryption schemes and digital signature [6]. Although we have described RSA for its inefficiency and low performance, we would like to evaluate the speed of RSA in our platform.

We choose ECC as the other candidate. ECC has been well studied and applied to a number of projects for years. All the facts show that ECC can provide enough security and more efficient than RSA. In 2005, the U.S. National Security Agency (NSA) announced that Suite B includes ECDH and ECDSA as for key exchange, digital signatures. NESSIE also recommended ECC as asymmetric encryption scheme and digital signature algorithm. We would like to compare how much faster ECC can perform compared to RSA in our platform.

We do not select Rabin algorithm, since its unbalance with respect to performance for encryption and decryption. Although according to our analysis above, for encryption, Rabin algorithm may perform even better than ECC, Rabin is quite inefficient for decryption due to its inherent property. We do not select ElGamal algorithm, since the computation of ElGamal is even more intensive than RSA. That we do not select NTRU as our candidate is not because of the performance, but the patent. Actually, according to analysis above, NTUR performs even better than ECC. Since it is patented, we cannot find any source code from any open source project. Therefore, we are not able to evaluate it in our platform. We do not choose HECC, since the use of HECC is still of pure academic interest. The research on HECC is still ongoing.

4 BENCHMARKING ON ARM PLATFORM

4.1 Introduction to ARM Platform

ARM (Advanced RISC Machines) is a world-leading provider of embedded microprocessors. It offers a very wide range of microprocessors based on ARM architecture, which is famous as high performance, small implementation size, low cost and low power consumption. The reason for these properties is due to the architectural simplicity of ARM core. Except microprocessors, ARM also provides a series of ARM core technologies, architecture extensions and system-on-chip schemes.

ARM architecture includes a number of key features of RISC [48]:

- Large uniform register file;
- Load/store architecture: data-processing operations not directly on memory contents, but only on register contents;
- Simple addressing modes: all addresses being determined from instruction fields and register contents only;
- Uniform 16 × 32-bit register file.

ARM CPU core includes a series of types, such as ARM7, ARM7TDMI (Thumb), ARM9TDMI and so on. Here we use the STM32F103ZE development board from IAR System as our evaluation platform. STM32F103ZE is a 32-bit ARM Cortex-M3 Microcontroller from ST. ARM Cortex-M3 is a high-performance and industry-leading RISC core with maximum 72 MHz frequency and 1.25 DMIPS/MHz (Dhrystone 2.1) performances at 0 wait state memory access for highly deterministic real-time applications.

STM32F103ZE has 256 to 512 Kbytes of Flash memory, up to 64 Kbytes of SRAM, flexible static memory controller with 4 Chip Select. It supports Compact Flash, SRAM, PSRAM, NOR and NAND memories. It supports three low-power modes, sleep mode, stop mode and standby mode to achieve the best compromise between low power consumption, short startup time and available wakeup sources. It has up to 112 fast I/O ports and up to 11 timers. It can provide a low-cost platform that meets the needs of MCU implementation, with a reduced pin count and low-power consumption, while delivering outstanding computational performance and an advanced system response to interrupts. Therefore, it suitable for a wide range of applications:

- Motor drive and application control
- Medical and handheld equipment
- PC peripherals gaming and GPS platforms
- Industrial applications, PLC, inverters, printers, and scanners

- Alarm systems, video intercom, and HVAC

4.2 Methodology and Consideration

For evaluation, we will introduce the implementation tools we use, the software implementation sources, the cipher parameters we choose and the methods we used to measure processing time and footprint of each algorithm.

4.2.1 Implementation Tools and Settings

We use IAR Embedded Workbench IDE 5.50 as our evaluation tool. The IAR Embedded Workbench IDE is a powerful integrated development environment. It seamlessly integrates the highly optimizing IAR C/C++ Compiler, the IAR assembler, the versatile IAR ILINK Linker including accompanying tools and the IAR C-SPY Debugger.

We use the most of default project settings of IAR Embedded Workbench, except optimization settings. In IAR Embedded Workbench, there are several optimization levels: None, Low, Medium and High. For High level of optimization, there are also three options: Size, Speed and Balance. On different level of optimizations, there are several transformations: Common subexpression elimination, Loop unrolling, Function inlining, Code motion, Type-based alias analysis, Static variable clustering and Instruction scheduling. In our evaluation, we choose high level of optimization and enable all the transformations for optimizations.

STM32F103ZE supports maximum 72 MHz frequency. In our evaluation we set this maximum frequency. For stack and heap, we set 0x4000 for stack size and 0x8000 for heap size.

4.2.2 Implementation Sources

In our evaluation, we use only pure C software implementations, but not assemble language or hardware implementations. In order to achieve reasonable performance of each algorithm and give a fair comparison, we adapt existing source code of each algorithm from the inventor of the algorithms or famous open source projects. We will explain where the implementation of each algorithm is adapted and briefly introduce several open source projects.

1) Symmetric algorithms

The implementation of AES is from OpenSSL. Openssl project is an open source project. It includes implementation of the Secure Sockets Layer and Transport Layer Security protocols as well as a cryptography library. A number of basic cryptographic functions and utility functions are included in its core library implemented in C programming language. We adapted the latest version 1.0.0 released on March 29, 2010. AES from OpenSSL is optimized and fully accelerated with 10 tables. We used all default configurations of algorithms from OpenSSL. For instance, we keep Macro definition FULL_UNROLL undefined in order to enable some loop unrolling.

The implementations of Skipjack are adapted from FreeBSD and TinySec. Here we use two implementations from two different sources, since implementations from these two sources are quite different. We want to check the difference of performance using these two source codes. The reason we choose the source, TinySec, is that it is an optional part of TinyOS, which is the de facto operating system for WSN. Implementation of Skipjack from TinySec is written in NesC language. We transplant the algorithm into C language without changing anything. FreeBSD is a free operating system widely used in servers, desktops and embedded devices. Implementation of Skipjack is a part of FreeBSD kernel opencrypto code.

The implementation of XTEA and XXTEA are entirely from the inventor of these algorithms. When he proposed these two algorithms, he also provided C language implementations. We do not change anything of XTEA and slightly change a bit of XXTEA without affecting its performance.

The implementation of Twofish is also from its inventor Bruce Schneier. Bruce Schneier provided two versions of implementations of Twofish. One is reference C implementation; the other is optimized C implementation. Our code is adapted from optimized version without changing anything.

2) MAC algorithms

The implementation of CMAC is from [112]. [112] has already provided the C language implementation of CMAC, only without AES functions. Therefore, we apply AES core functions from OpenSSL which is used above.

The implementation of HMAC is from [113]. The implementation of HMAC in C language is given as an appendix of [113]. The hash functions SHA-1 is adapted from PolarSSL. PolarSSL is a light-weight open source cryptographic library using C programming language. It is very easy for us to include cryptographic capabilities in our embedded applications with as little hassle as possible. For the hash function SHA-256, we once again choose two different sources. One is also from PolarSSL; the other is from [49].

The implementation of UMAC is adapted from [33]. UMAC has several versions. It was first described in 1999. Great changes were made in 2000 in order to improve its performance fro short messages. The final version was released in 2004 with only slight changes for simplicity. We choose the 2000 version in our evaluation, since this version has already included every part including block cipher part.

3) Asymmetric algorithms:

The implementation of RSA and ECC are both from LibTomCrypt. LibTomCrypt is a portable ISO C cryptographic library meant to be a tool set for cryptographers who are designing cryptosystems. Quite different from symmetric and MAC algorithms, asymmetric algorithms involve big number operations. Therefore, besides LibTomCrypt, we also use LibTomMath as our big number library. LibTomMath is a library of source code which provides a series of efficient and carefully written functions for manipulating large integer numbers. It was written in portable ISO C source code so that it will build on any platform with a conforming C compiler. Another option is OpenSSL. It also includes these two implementations. However, the implementations of RSA and ECC from OpenSSL are platform depended and the documentation of source code is rather poor. Therefore, we gave up OpenSSL and choose LibTomCrypt and LibTomMath.

4.2.3 Cipher Parameters

Here we will illustrate the parameters we chose for each algorithm.

1) Symmetric algorithms

The parameters of symmetric algorithms are listed in Table 4.1

Table 4.1: Parameters for Symmetric Algorithms (Bytes)

Parameters	AES	Skipjack	XTEA	XXTEA	Twofish
Block Length	16	8	8	Arbitrary(16)	16
Key Length	16	10	16	16	16
Rounds	10	32	32	19(according to block size)	16

2) MAC algorithms

For MAC algorithms, except for secret key length, no specific parameters are needed. The key length we used is 16 bits long.

3) Asymmetric algorithms

For RSA encryption, we use the encryption scheme RSAES-OAEP. RSAES-OAEP is to combine RSA algorithm with the Optimal Asymmetric Encryption Padding (OAEP) method. OAEP is invented by Mihir Bellare and Phillip Rogaway [115]. When RSAES-OAEP is applied, plaintext should first be encoded with OAEP and then encrypted with RSA. Since textbook RSA algorithm is not secure, more precisely not CCA2 secure, RSA should never be used directly. After applying OAEP, the plaintext is first padded in a in a randomized way which greatly enhances the security of RSA encryption.

For RSA digital signature, there are several methods for padding. In our evaluation, we use RSA-PSS as our signature scheme. The reason we choose RSA-PSS instead of traditional PKCS #1 v1.5 is that the connection of PKCS #1 v1.5 signatures to the RSA problem has never been proven, while RSA-PSS, in contrast, has such a proof if one models its hash functions as "random oracles" as is commonly done.

Another important parameter for RSA is the public exponents. In our evaluation, we choose this public exponents $e = 65537$, which is a common value for RSA algorithm.

For both ECDH and ECDSA, we choose the elliptic curve secp160r1 in our evaluation. The curve"secp160r1" was standardized by Standards for Efficient Cryptography (SEC2). It has two advantages that can be used to speed up prime field arithmetic reduction and to speed up curve arithmetic double and add. Because its underlying prime field is based on a pseudo Mersenne prime the reduction in the prime field can be done by several shifts and adds which is much faster than any other known algorithm on constrained devices.

4.2.4 Methods for Measurement

In our evaluation, two main key features of all algorithms we will evaluate are memory and processing time consumption performance. The methods to measure these two values are introduced as follows.

4.2.4.1.1 Memory

To measure the footprint of each algorithm, there are two methods. One is to set IAR Embedded Benchmark IDE to generate a list file for each C file when compiling the project.

In the end of each list file, the usage of memory is shown into three types: code memory, const code memory and date memory. Code memory represents the size of executive program in the memory. Const code memory represents the footprint of the initialized const values. The data memory represents the size of RAM this C program file use. We can directly read all this information from a list file. Another method is to set IAR Embedded Benchmark IDE to generate a map file as one of the output files when compiling the project. Different from a list file, a map file will show the total footprint of the whole project. The name of each memory type is different from the name in list file. There are also three types of memory in a map file: read-only code memory, read-only data memory and read-write data memory, which are corresponding to code memory, const code memory and data memory in a list file. We have checked that they have same meanings.

The first method is more convenient than the second one, since each footprint of each algorithm can be directly read out through list files. If several algorithms are implemented in one project, some extra work is needed when the second method is used. That is we shall read the map file before one algorithm is added into the project and read the map file once again after it is added. By subtracting two memory sizes, we can get the footprint of this algorithm. However, the second method is more accurate than the first one. Since when compiling a project, not all codes in one C file are included. Even when one algorithm includes several C files, the footprint of this algorithm is not just simply obtained by adding all memory from involved list files. Through the second method, we can get more accurate result, since we can directly see when adding one more algorithm, how much of more memory is needed. Therefore, in our evaluation, we apply the second method to measure the footprint.

4.2.4.1.2 Performance

To measure the processing time of an algorithm, there are also two methods. One is to use profiling tools provided by IAR Embedded Benchmark IDE. When entering the debug mode, we can open profiling windows. Before running the program, we need to activate the tool. After the whole program runs to the end or is stopped by a breakpoint, we can obtain CPU cycles for each function in the project. If one function is called several times, accumulated CPU cycles can also be obtained. The other method to measure performance is to use a timer of the system. There are 11 timers provided by STM32F103ZE. After analysis, we consider that the timer systick is the most suitable one. The timer systick is a 24 bit timer built into the ARM core. It is simple and suitable for generating a tick for an operating system or for measuring delays. The units of measurement by this method are not CPU cycles, but ticks. The systick clock source in STM32F103ZE can be specified to two frequencies, 9 MHz and 72 MHz. In order to guarantee an accurate measurement, we measured Systick timer with an oscilloscope to verify the implement, which shows this method is also correct.

The advantage of the first method is simple. No extra codes and timers for measurement are needed. The performance of functions can be obtained at once. Therefore, it is very easy to find out which parts are the bottlenecks of the performance in this project. The two main disadvantages of this method are: first, it is impossible to measure the CPU cycles between two specific lines of the codes in a project, since the measurement is based on functions, only on progress; second, when a project is huge, a number of functions are involved, it is impossible to use this method, since IDE will announce you that no more breakpoints can be set, which means that extra resources are used for this method.

Finally, we choose the systick timer to measure the program performance. The reason is that we can measure the processing time between any two lines in the project, regardless of other uninteresting parts. Another reason is that our benchmarking project is huge, it is impossible for us to apply the first method due to the lack of system resources. We consider that specifying 9 MHz as the source clock is enough for accuracy. We write extra code for measurement using this timer. We read out the timer values before and after the test point. By subtracting two timer values, we can calculate the time consumption. One important thing when using the systick timer is that the counter will swap around when the timer decreases to zero. We also take this into account in our measurement.

5 RESULTS AND ANALYSIS

In this part, we will present the results of our evaluation. The results include two main parts: memory and time consumption. The methods for measuring both of two were introduced before. We performed our measurement according to different algorithms and running modes. All comparisons are shown below, followed by our analysis of the results.

5.1 Symmetric Algorithms

5.1.1 Memory

To measure footprint of all algorithms, we use the simplest mode, ECB mode, for encryption and decryption, since for ECB mode no Initialization vector (IV) is needed and no extra memory is needed which gives us more exact results. We applied both size and speed high optimization options for compiling. The results of two optimizations are shown in Table 4.2 and Table 4.3.

Table 4.2: Footprint using Size High Optimization (byte)

Memory Type	**AES**	**Skipjack(TinySec/ FreeBSD)**	**XTEA**	**XXTEA**	**Twofish**
Read-only Code Memory	3172	2262/ 3832	524	1174	7594
Read-only Data Memory	8600	508 / 470	180	2004	685
Read-write Data Memory	0	0/0	0	0	4628

Table 4.3: Footprint using Speed High Optimization (byte)

Memory Type	**AES**	**Skipjack(TinySec/ FreeBSD)**	**XTEA**	**XXTEA**	**Twofish**
Read-only Code Memory	3494	2456/ 3782	612	1238	10134
Read-only Data Memory	8602	508/ 474	180	2230	650

Read-write Data Memory	0	0/0	0	0	4628

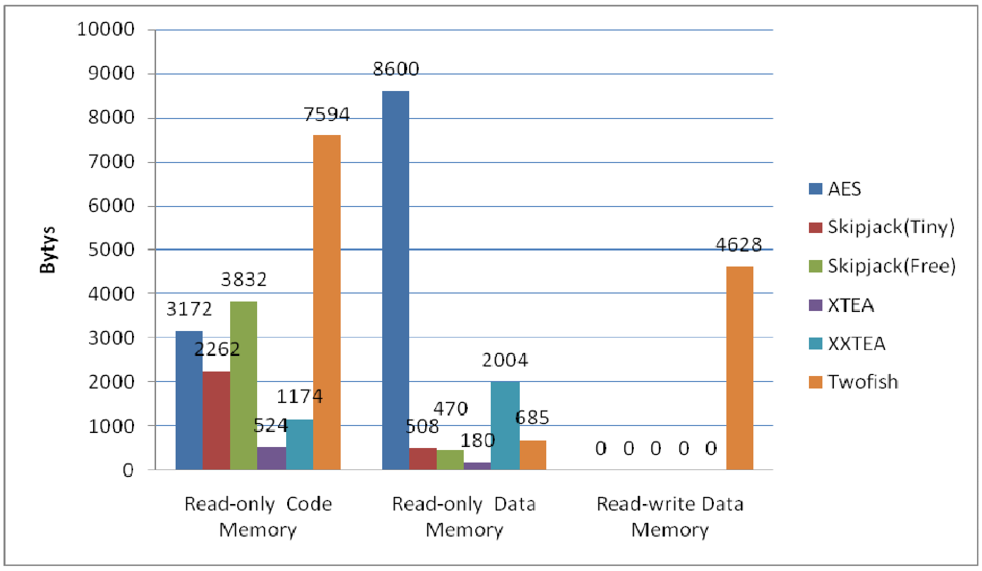


Figure 4.1 Footprint using Size High Optimization

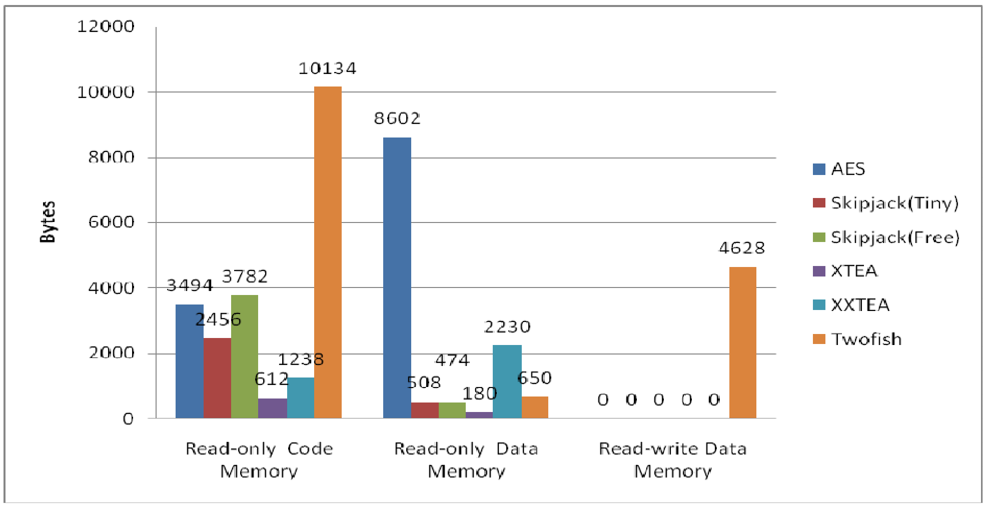


Figure 4.2 Footprint using Speed High Optimization

5.1.2 Performance

To evaluate the algorithms, we define the range of the plaintext length, from 8 bytes to 1024 bytes. First, we will evaluate the performance of key setup stage using both size high optimization and speed high optimization. The result is shown in Table 4.4 and 4.5, followed by Figure 4.3 and 4.4 illustrating this result. Then, we evaluate the time consumption for encrypting different length of plaintext using both size high optimization and speed high optimization, which are listed in Table 4.6, 4.7 and plotted in Figure 4.5, 4.6. As we mentioned above, the timer Systick we used is set to 9 MHz. In Table 4.6 and Table 4.7, we give two types of values: tick numbers and time consumption values.

Table 4.4: Size Optimized Key Setup Performance (systick/ms) of Symmetric Algorithms

Memory Type	**AES (tick/ms)**	**Skipjack(TinySec) (tick/ms)**	**Skipjack(FreeBSD) (tick/ms)**	**XTEA (tick/ ms)**	**XXTEA (tick/ms)**	**Twofish (tick/ms)**
Encrypt Key Setup	144/0.016	57/0.006	5774/0.642	0	0	24/0.003
Decrypt Key Setup	506/0.057	57/0.006	5774/0.642	0	0	24/0.003

Table 4.5: Speed Optimized Key Setup Performance (systick/ms) of Symmetric Algorithms

Cipher	AES	Skipjack(TinySec)	Skipjack(FreeBSD)	XTEA	XXTEA	Twofish
Encrypt Key Setup	127/0.014	51/0.005	6091/0.535	0	0	24/0.003
Decrypt Key Setup	526/0.057	51/0.005	6091/0.535	0	0	24/0.003

Table 4.6: Size Optimized Encryption Performance (systick/ms) of Symmetric Algorithms

Message Length (byte)	**AES (tick/ms)**	**Skipjack(TinySec) (tick/ms)**	**Skipjack(FreeBSD) (tick/ms)**	**XTEA (tick/ms)**	**XXTEA (tick/ms)**	**Twofish (tick/ms)**
8	287/0.032	301/0.034	136/0.015	238/0.027	378/0.041	250/0.028
16	282/0.032	588/0.066	257/0.029	461/0.051	378/0.041	250/0.028
32	552/0.062	1161/0.130	501/0.056	908/0.101	742/0.081	456/0.051
64	1094/0.122	2308/0.259	988/0.110	1801/0.201	1471/0.160	867/0.096

128	2176/0.243	4599/0.516	1962/0.218	3589/0.400	2929/0.319	1689/0.188
256	4339/0.484	9183/1.031	3910/0.435	7162/0.799	5843/0.637	3334/0.371
512	8668/0.966	18351/2.061	7806/0.869	14310/1.596	11674/1.272	6624/0.736
1024	17324/1.934	36688/4.121	15598/1.737	28607/3.191	23334/2.543	13205/1.467

Table 4.7: Speed Optimized Encryption Performance (systick/ms) of Symmetric Algorithms

Message Length (byte)	**AES (tick/ms)**	**Skipjack(TinySec) (tick/ms)**	**Skipjack(FreeBSD) (tick/ms)**	**XTEA (tick/ms)**	**XXTEA (tick/ms)**	**Twofish (tick/ms)**
8	294/0.031	281/0.031	125/0.138	157/0.017	373/0.040	216/0.024
16	289/0.032	553/0.061	240/0.026	303/0.033	372/0.040	216/0.024
32	568/0.063	1095/0.121	471/0.052	594/0.064	732/0.079	390/0.043
64	1129/0.125	2181/0.242	932/0.103	1177/0.128	1454/0.157	741/0.082
128	2250/0.248	4352/0.483	1853/0.204	2343/0.253	2900/0.312	1440/0.160
256	4492/0.496	8694/0.965	3698/0.407	4675/0.505	5788/0.623	2839/0.315
512	8976/0.991	17378/1.931	7385/0.814	9339/0.626	11566/1.245	5637/1.009
1024	17944/1.981	34745/3.860	14762/1.628	18668/2.017	23123/2.489	11233/1.248

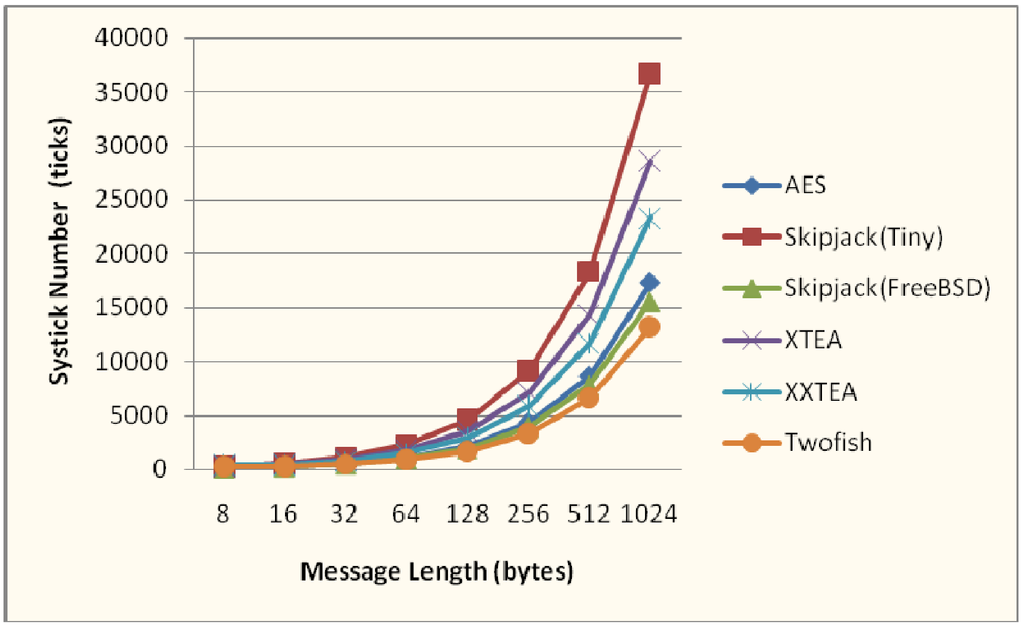


Figure 4.3 Size Optimized Encryption Performance of Symmetric Algorithms (systicks)

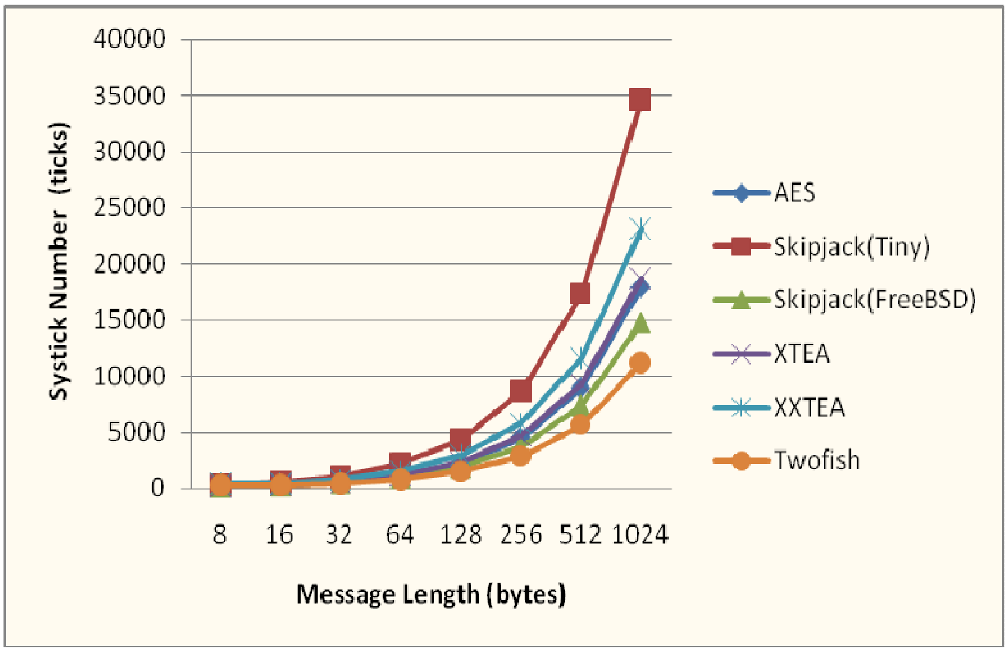


Figure 4.4 Speed Optimized Encryption Performance of Symmetric Algorithms (systicks)

As we introduced above, for symmetric encryption, there are several operating modes. Therefore, we apply these different modes to AES algorithm to encrypt different lengths of plaintext, which will give us information that how the operating mode affects performance in our platform. The result is shown in Table 4.8 and plotted in Figure 4.5.

Table 4.8: Four Operating Mode Performance of AES (Size/Speed Optimization)

Message Length (byte)	**ECB (tick)**	**CBC (tick)**	**OFB (tick)**	**CFB (tick)**	**CTR (tick)**
8	287/290	323/315	300/301	304/305	308/310
16	283/285	340/313	320/319	325/325	332/331
32	554/560	656/605	629/629	639/640	651/652
64	1095/1112	1289/1190	1247/1248	1268/1268	1290/1292
128	2179/2214	2556/2359	2482/2487	2522/2525	2566/2575
256	4347/4420	5087/4698	4953/4965	5034/5039	5121/5139

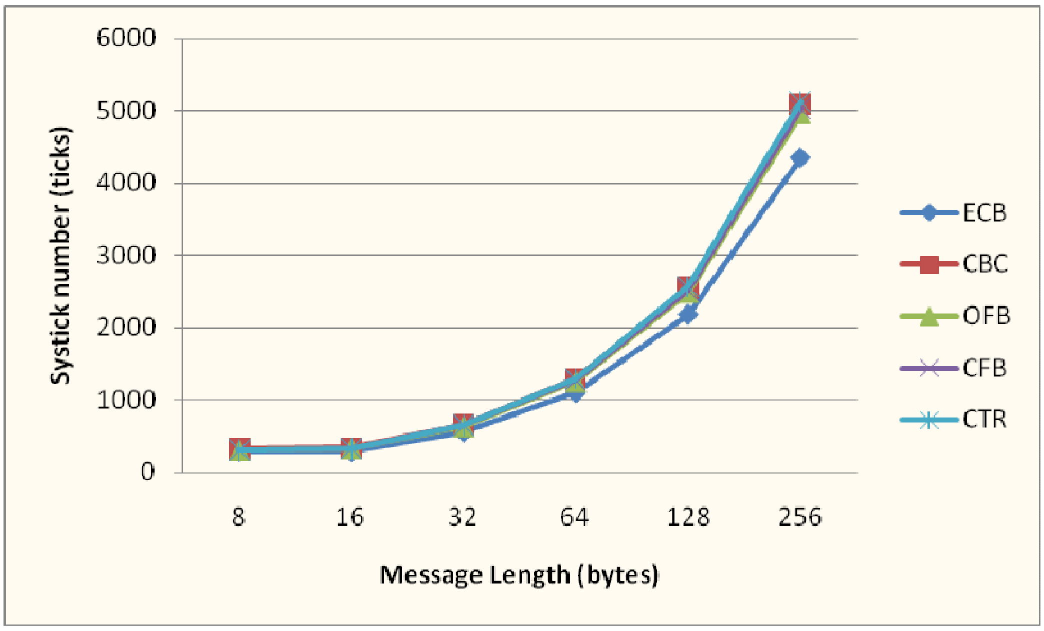


Figure 4.5 Size Optimized Operating Mode Performance of AES

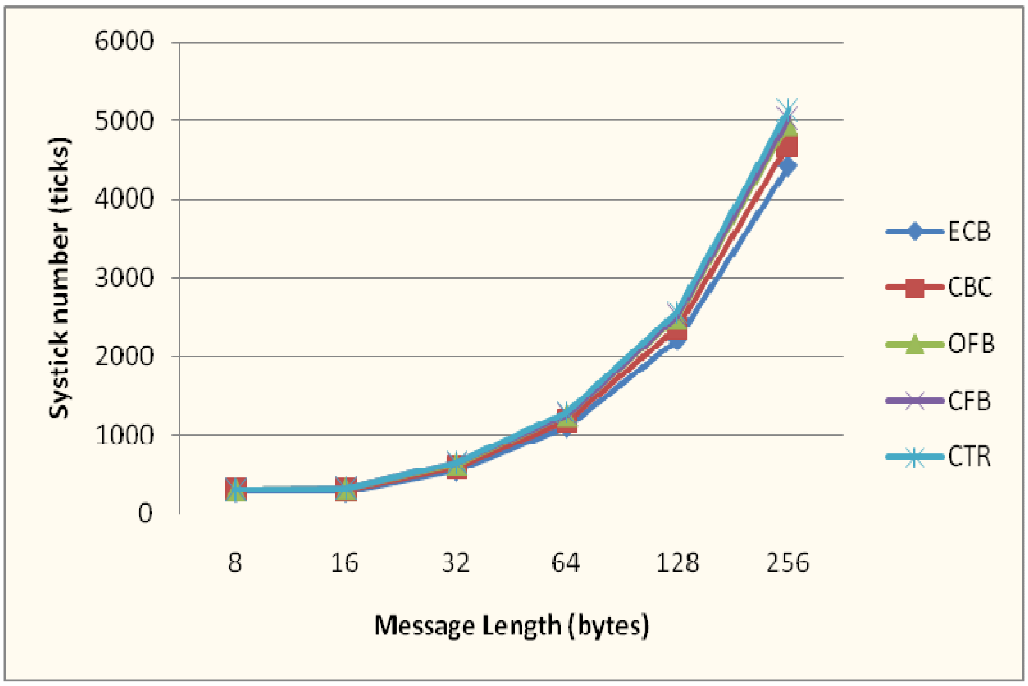


Figure 4.6 Speed Optimized Operating Mode Performance of AES

5.1.3 Analysis and Conclusion

According to tables above, it shows that size and speed optimization do make sense to several algorithms, but not all. For instance, the performance of XTEA using speed optimization is greatly improved compared to using size optimization. Taking a converse example, the size of Skipjack adapted from FreeBSD using speed optimization is even smaller than that using size optimization. Conversely, the speed performance of AES using size optimization is even better than that using speed optimization. It illustrates that the effect of optimization is related to the implementation of algorithms. The size optimization will not always improve the size occupation of the code and perform worse than the speed optimization and vice versa.

Taking footprint into account, XTEA and XXTEA take the least memory of all compared algorithms due to the extremely simple implementation. Therefore, XTEA and XXTEA are the most suitable algorithms when the memory of the system is strictly constrained. Skipjack algorithm requires medium size of memory among all algorithms. The footprint of AES and Twofish are quite close, but in different types. The code size of AES is smaller than Twofish. Since a great number of constant tables are used in AES, AES takes much more const code memory. Although the const memory for Twofish is extremely small, it requires much larger RAM than any other algorithms. Thus, both AES and Twofish are inappropriate choices for memory constrained applications.

When comparing the performance, we need to consider two stages. The first is key setup stage. The key setup stage is necessary for AES, Skipjack and Twofish. It means that if we need to apply one of these algorithms, we have to expand the cipher key into the encryption or decryption key schedule. For Skipjack and Twofish, the encryption and decryption key setup procedures are the same, but for AES, they are different. According to the result, the decryption key setup of AES is much slower than encryption key setup. There is no key setup stage for XTEA and XXTEA, which shows the simplicity of these two algorithms once again.

The most meaningful part of this evaluation of symmetric algorithms is to compare the encryption or decryption performance of all algorithms. Since the difference of performance between encryption and decryption of symmetric algorithm is not obvious, we only consider the encryption part in our comparisons. From the tables above, it is surprisingly that Twofish has the best performance of all in our platform. It proves that AES may perform the best in average, but when considering a particular platform, extra evaluation is necessary. Although the size of XTEA and XXTEA is very small, the performances of these two algorithms are worse than AES most of the time. The performances of Skipjack from TinySec and FreeBSD are extremely different. However, according to Figure 4.4, XTEA using speed optimization has very similar performance compared with AES. Skipjack from TinySec performs worst of all algorithms, but one from FreeBSD performs much better, even better than AES. The reason for this may be due to the transpformation from NesC to pure C. Skipjack from TinySec may perform efficiently in TinyOS platform, but not in our platform. It proves that the performance of an algorithm is greatly related to the way of implementation.

As we introduced before, the operating mode of symmetric ciphers is also an important factor. According to Table 4.8, ECB is the fastest operating mode among all. However, on the other hand, ECB is also the weakest mode in security. No applications should be implemented using ECB mode .From the results, the performance of CBC, OFB, CFB and CTR are quite similar to each other. When using size optimization, OFB mode has slightly better performance than other three advanced modes, which means a slightly more energy-efficient. Since there is no obvious difference between these operation modes, we shall choose encryption mode according to different applications. As we introduced before, both OFB and CFB modes can be used in stream-oriented transmission, since both of them can turn a block cipher into a stream cipher. CTR mode has more or less the same properties as OFB and CFB, but plaintext can be encrypted in parallel when using CTR mode. Therefore, it is suitable for operations on a multi-processor device. CBC mode is the most widely used mode, although it has several drawbacks. For instance, encryption using CBC cannot be parallelized. However, [114] proves that CBC mode is more secure than other modes due to less information leakage when IV is repeated.

Therefore, our conclusion is that in our platform, when the memory resources are not strained, Twofish is the top choice among all candidates. However, if memory occupation is necessary to be taken into account, we may consider XTEA and XXTEA depending on the security level and performance requirement. For operating mode, CBC, CTR, OFB and CFB have quite similar performances. Choose operating mode should be based on other requirements of applications instead of encryption speed. For instance, if encryption should be handled in parallel, CTR should be considered prior to other modes. One thing is important that XXTEA does not need any operating mode due to its natural property.

5.2 MAC Algorithms

5.2.1 Memory

MAC algorithms usually consist of two algorithms. For instance, CMAC-AES consists of CMAC algorithm and AES algorithm and HMAC-SHA1 consists of HMAC algorithm and SHA-1 algorithm. Therefore, the footprint of each MAC algorithm consists of two algorithms. The memory can be read out from map files as we introduced before. The results are listed in Table 4.9 and Table 4.10 and plotted in Figure 4.6 and 4.7 using both size and speed high optimizations.

Table 4.9: Footprint of MAC Algorithm using Speed High Optimization (byte)

Memory Type	CMAC-AES	HMAC-SHA1	HMAC-SHA256(Polar)	HMAC-SHA256	UMAC
Read-only Code Memory	3058	4676	9080	1104	4336
Read-only Data Memory	4194	84	84	311	4520
Read-write Data Memory	16	0	0	288	33444

Table 4.10: Footprint of MAC Algorithm using Size High Optimization (byte)

Memory Type	CMAC-AES	HMAC-SHA1	HMAC-SHA256(Polar)	HMAC-SHA256	UMAC
Read-only Code Memory	2080	4458	9486	902	3698

Read-only Data Memory	4196	86	86	313	4522
Read-write Data Memory	16	0	0	288	33444

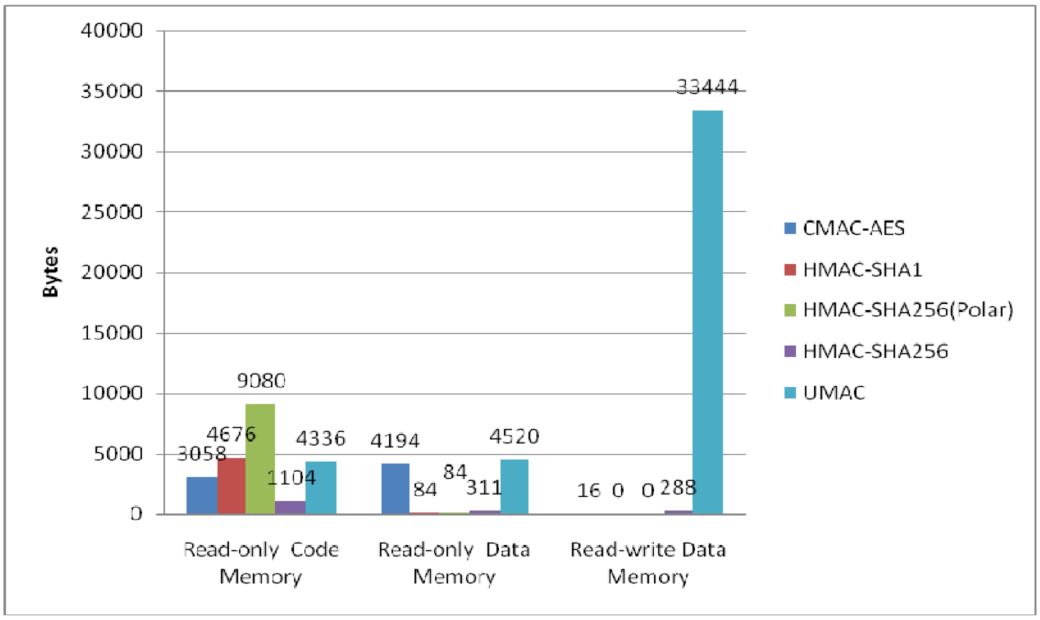


Figure 4.7 Footprint of MAC Algorithm using Speed High Optimization

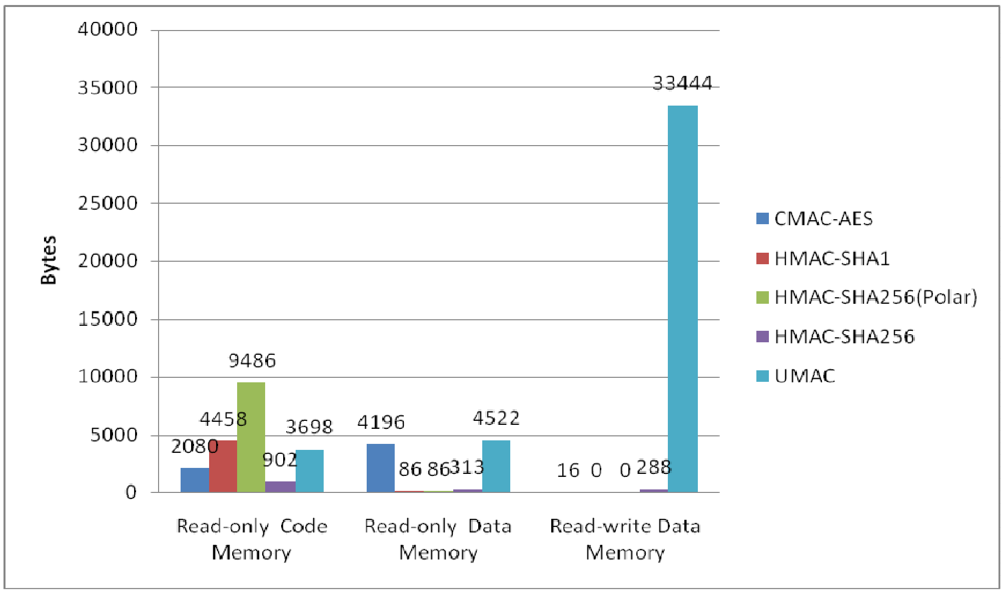


Figure 4.8 Footprint of MAC Algorithm using Size High Optimization

5.2.2 Performance

Same as the evaluation of symmetric algorithm, we also define the range of plaintext from 8 bytes to 1024 bytes. We choose a key length of 16 bytes in our evaluation. The results are listed in Table 4.11 and 4.12 and plotted in Figure 4.8 and 4.9.

Table 4.11: Speed Optimized Encryption Performance (systick/ms) of MAC Algorithms

Plaintext Legnth (byte)	CMAC-AES (tick/ms)	HMAC-SHA1 (tick/ms)	HMAC-SHA256(Polar) (tick/ms)	HMAC-SHA256 (tick/ms)	UMAC (tick/ms)	UMAC Key Setup (tick/ms)
8	985/0.109	1141/0.127	2245/0.250	3118/0.350	409/0.045	30556/3.389
16	946/0.105	1140/0.127	2244/0.250	3118/0.350	407/0.045	
32	1365/0.151	1139/0.127	2243/0.250	3118/0.350	401/0.044	
64	2199/0.244	1362/0.151	2735/0.304	3846/0.431	445/0.049	
128	3868/0.429	1586/0.176	3231/0.359	4552/0.511	534/0.059	
256	7206/0.801	2038/0.226	4223/0.469	5965/0.670	711/0.079	
512	13882/1.543	2940/0.327	6206/0.689	8791/0.989	1064/0.118	
1024	27234/3.028	4743/0.527	10171/1.130	14445/1.625	1773/0.197	

Table 4.12: Size Optimized Encryption Performance (systick/ms) of MAC Algorithms

Plaintext Legnth (tick/ms)	CMAC-AES (tick/ms)	HMAC-SHA1 (tick/ms)	HMAC-SHA256(Polar) (tick/ms)	HMAC-SHA256 (tick/ms)	UMAC (tick/ms)	UMAC Key Setup (tick/ms)
8	1061/0.118	1238/0.138	2236/0.249	3735/0.416	635/0.070	48749/5.415
16	1023/0.113	1235/0.138	2234/0.249	3734/0.416	633/0.071	
32	1457/0.162	1235/0.138	2234/0.249	3733/0.416	626/0.070	
64	2326/0.259	1476/0.165	2719/0.303	4610/0.513	670/0.075	
128	4064/0.451	1720/0.192	3207/0.357	5461/0.607	759/0.084	
256	7540/0.838	2205/0.246	4182/0.466	7163/0.797	937/0.104	

512	14492/1.61 0	3179/0.354	6132/0.682	10565/1.175	1290/0.144	
1024	28396/3.15 6	5125/0.570	10032/1.112	17373/1.931	1999/0.222	

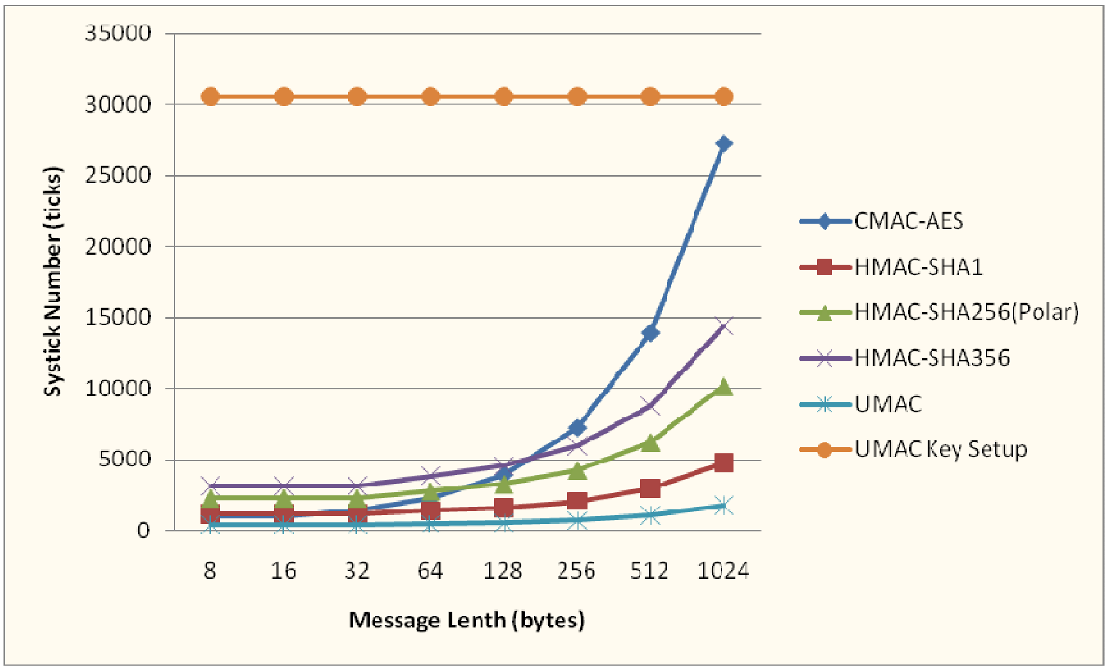


Figure 4.9 Speed Optimized Encryption Performance of MAC Algorithms (systick)

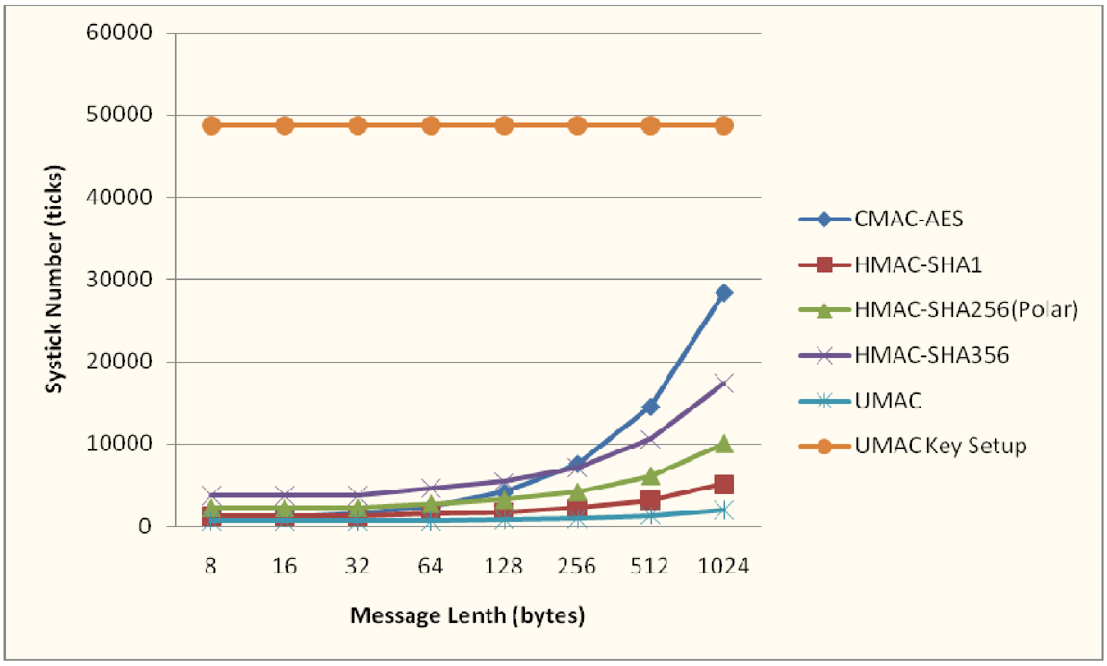


Figure 4.10 Size Optimized Encryption Performance of MAC Algorithms (systick)

5.2.3 Analysis and Conclusion

First, we consider about the footprint of each MAC algorithm. From the results, it is obvious that HMAC-SHA256 from [49] takes least memory, even less than HMAC-SHA1 adapted from PolarSSL. The footprint of HMAC-SHA256 from PolarSSL is about twice larger than HMAC-SHA1. The memory needed by CMAC-AES is medium, a bit larger than HMAC-SHA1, but smaller than HMAC-SHA256 from PolarSSL. UMAC requireds a huge memory, in all three types, especially RAM. It means that UMAC is not suitable for memory constrained devices.

Then, we analyze the performance of all MAC algorithms. From Table 4.11, 4.12 and Figure 4.8, 4.9, UMAC has outstanding performance from short messages to long messages among all algorithms. Even processing 1024 byte message, the time consumption of UMAC is still less than 0.3 ms. However, the disadvantage of UMAC is also quite obvious, which is the key setup stage. UMAC is the only MAC algorithm among all requiring key setup. This stage for UMAC is extremely slow. Even using speed high optimization, the time consumption is still more than 3 ms. This means that if the secret key is changed frequently, UMAC could be extremely inefficient. The performance of UMAC also explains why the footprint is huge, when comparing with others. CMAC-AES has also excellent performance with short messages. When the message length is shorter than 32 bytes, it is faster than other MAC algorithms, except UMAC. However, when a message length is longer than 32 bytes, the time consumption of CMAC-AES increases rapidly. The performance of HMAC-SHA1 is decent for both short and long messages. When authenticating a 1024-byte message, it takes only around 0.5 ms for HMAC-SHA1. HMAC-SHA256 from PolarSSL is about twice slower than HMAC-SHA1 due to the twice longer message digest, which is a quite reasonable result. However, the performance of HMAC-SHA256 from [49] is about triple worse than HMAC-SHA1. The reason is that in HMAC-SHA256 from [49] uses much fewer tables than one from PolarSSL. Therefore, the footprint of HMAC-SHA256 from [49] decreases significantly, on the price of performance.

According to the evaluation and analysis, we come to the conclusion that if the memory usage of an algorithm is not extremely important and the secret key of MAC algorithm is not changed frequently, then UMAC is the best candidate due to its outstanding performance for both short and long messages. If a system already has used AES algorithm as symmetric encryption scheme and transmission messages are not long, then CMAC-AES is also decent choice. In other cases, HMAC-SHA1 and HMAC-SHA256 can be selected according to the security level and performance requirement.

5.3 Asymmetric Algorithms

5.3.1 Memory

For asymmetric algorithms, a big number library is needed. Therefore, the total footprint of asymmetric does not only include algorithm itself, but also include the big number library. Since a number of files are involved, we read the footprint from the map file. One important thing is that when we use speed or balance high optimization, the encryption/ decryption and digital signature results are incorrect due to some over optimizations done by IAR built in compiler. Therefore, we only consider size high optimization in this part. The results of the footprints are shown in Table 4.13 and Figure 4.10.

Table 4.13: Footprint of Symmetric Algorithm using Size High Optimization (byte)

Memory Type	RSA-OAEP	RSA-PSS	ECDH	ECDSA
Read-only Code Memory	24320	34444	38068	35254
Read-only Data Memory	4017	9585	10977	9671
Read-write Data Memory	30200	30200	30200	26872

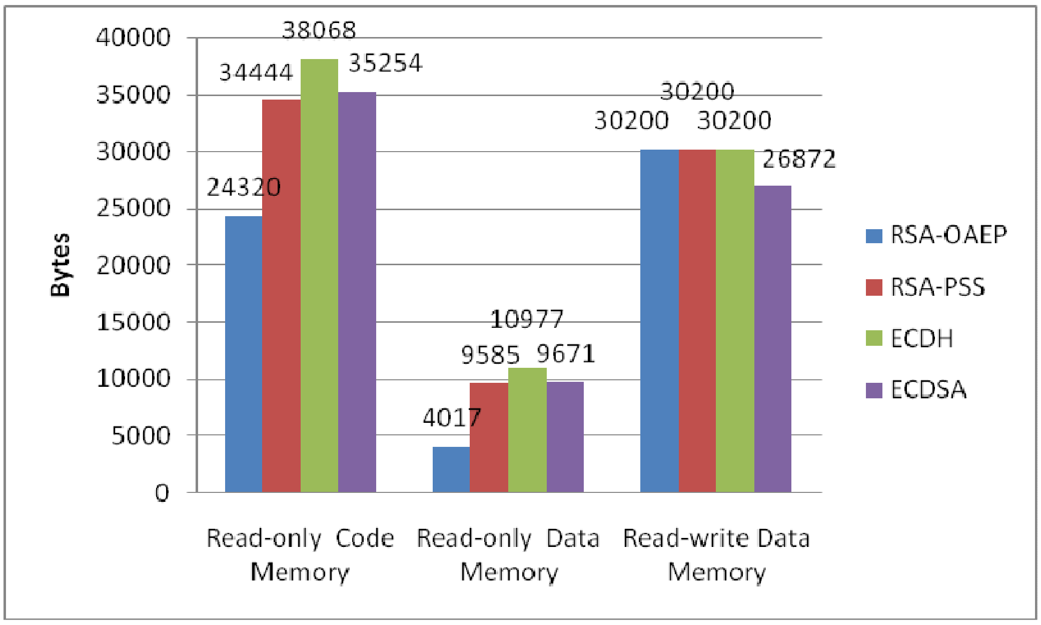


Figure 4.11 Footprint of Symmetric Algorithm using Size High Optimization

5.3.2 Performance

First, we compare the performance of the key setup stage for both RSA and ECC. In order to give a fair comparison, we choose the key lengths of both algorithms at the same security level. Therefore, we select ECC-160 corresponding to RSA 1024 and ECC-224 corresponding to RSA 2048. The results are shown in Table 4.14.

Table 4.14 Size Optimized Key Setup Performance of Asymmetric Algorithms (systicks/second)

Key Length	**ECC (tick/ms)**	**RSA (tick/ms)**
ECC-160 & RSA 1024	1253018 / 0.139	195373522 / 21.708
ECC-224 & RSA 2048	2284462 / 0.253	1719568935 / 191.063

Then, we evaluate the performance of encryption using ECC and RSA with different key lengths. Usually, asymmetric algorithms are used to encrypt secret keys for symmetric algorithms. Therefore, we choose our message length to be 128 bits. The results are shown in Table 4.15.

Table 4.15: Size Optimized Encryption Performance of Asymmetric Algorithms (systicks/second)

Message Length (bytes)	**ECC-160**		**ECC-224**		**RSA-1024**		**RSA-2048**	
	Enc.	Dec.	Enc.	Dec.	Enc.	Dec.	Enc.	Dec.
16	2438680	1269608	4541174	2301344	249575	2974389	871752	17744285

	/ 0.271	/ 0.141	/ 0.505	/ 0.256	/ 0.028	/ 0.330	/ 0.097	/ 1.972

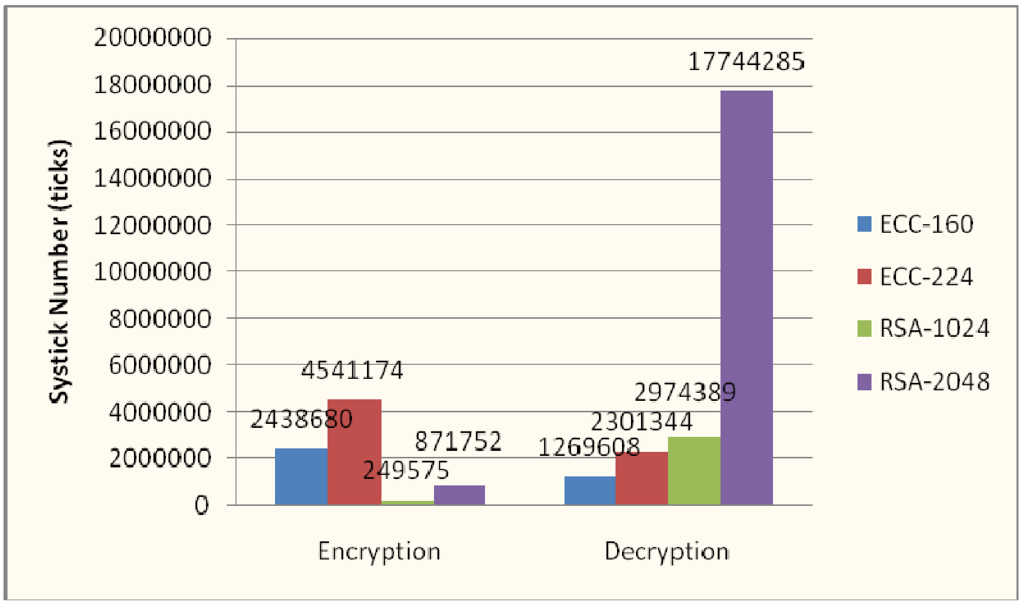


Figure 4.12 Size Optimized Enc/Dec Performance of Asymmetric Algorithms

Finally, we evaluate the performance of digital signature using ECC and RSA with different key lengths. Since for digital signature, only message digest is signed, instead of the whole message, we choose the message digest length to be 128 bits long. The results are shown in Table 4.16.

Table 4.16: Size Optimized Digital Signature Performance of Asymmetric Algorithms (systicks/second)

Message Length (bytes)	**ECC-160 (tick/ms)**		**ECC-224 (tick/ms)**		**RSA-1024 (tick/ms)**		**RSA-2048 (tick/ms)**	
	Sign.	Verify.	Sign.	Verify.	Sign.	Verify.	Sign.	Verify.
16	1267449 / 0.141	1769618 / 0.197	2306929 / 0.256	3210459 / 0.357	2969447 / 0.330	248543 / 0.028	17738811 / 1.971	867325 / 0.096

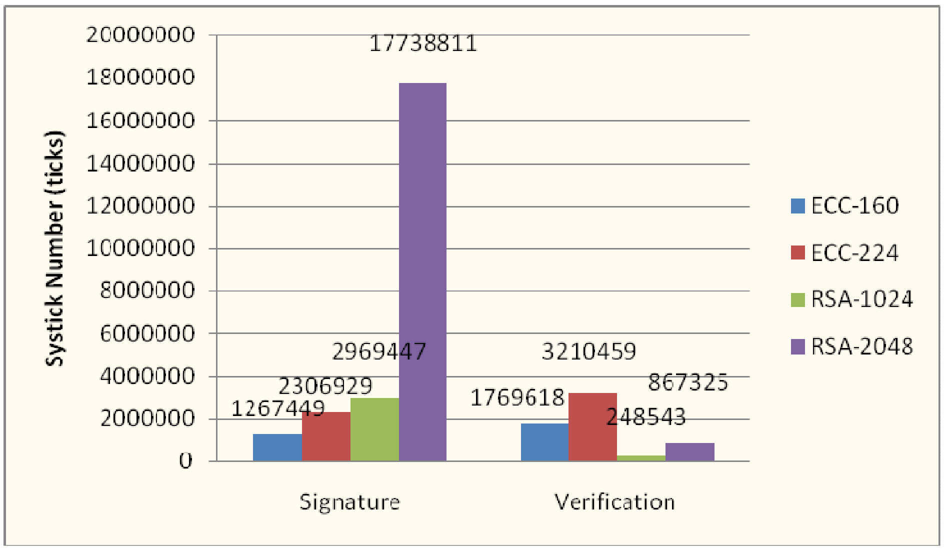


Figure 4.13 Size Optimized Digital Signature Performance of Asymmetric Algorithms

5.3.3 Analysis and Conclusion

From the results, we can see that the footprint and the performance are quite different from symmetric and MAC algorithms. Both RSA and ECC take a huge amount of memory, in both read-only code memory and read-write data memory. The cost of read-only data memory is still quite a lot, between 5000 – 10000 bytes. Therefore, both the software implementation using C of RSA and ECC asymmetric algorithms are not suitable for the memory constrained devices. Maybe more reasonable options are software implementation using assemble language or hardware implementation for these devices.

The huge difference of performance of these two asymmetric algorithms is the key setup stage. From Table 4.14, we can see that the time consumption of RSA key setup is extremely larger than ECC. For RSA 2048, it takes more than 3 minutes to finish the key setup stage, which is definitely unacceptable for any applications. Conversely, the performance of ECC key setup is quite decent. Both ECC-160 and ECC-224 only need less than 300ms.

When considering encryption and digital signature, the difference between RSA and ECC is that the performance of RSA is unbalanced. For instance, it takes RSA-2048 nearly 2 second to sign a message, but it only takes less than 0.1 second to verify it. For ECC, it has decent and balanced performance for encryption and decryption, signature and verification. The time consumption is between 0.15 – 0.5 seconds. The performance of ECC-160 is twice better than ECC-224 due to the twice security level. For RSA, the performance of RSA-2048 is almost six times worse than RSA-1024, which means that RSA is not appropriate for the high security level applications in embedded systems.

From the results above, we conclude that compared with RSA, ECC is much more suitable for embedded applications. ECC has decent performance in encryption scheme and digital signature. The performance of ECC can be improved by other implementations, such as hardware implementation acceleration.

6 CONCLUSION AND FUTURE WORK

6.1 Summary and Conclusion

Security is becoming one of critical factors in modern automation networks due to the introduction of Ethernet-based field buses. Countermeasures against security attacks are different according to hierarchical levels of automation systems. In this thesis, we focus on the security at lower field bus level. As we described before, Ethernet-based fieldbus protocols were designed with more concerns about safety rather than security. Cryptographic algorithms, which are critical parts in security, are not specified in many fieldbus protocols. Different cryptographic algorithms provide different objectives, such as confidentiality, integrity and authentication. The security and efficiency also vary a lot according to different algorithms. The goal of this thesis is to benchmark cryptographic algorithms for confidentiality, integrity and authentication in our specific platform. This includes symmetric algorithms, MAC algorithms and asymmetric algorithms.

In this thesis, we first briefly introduced several important concepts of security and cryptography, which are helpful to understand this thesis. Then, we presented a comprehensive survey of cryptographic algorithms. The goal of this survey is to select several candidates from different types of algorithms for further benchmarking due to the time limitation. In order to make our survey more convincing and find out the most suitable candidates for our evaluation, we studied most of widely used cryptographic algorithms which are considered to have outstanding performance and already applied in a number of security projects, standards or protocols. Based on the analysis both in security and performance, we chose AES, Skipjack, XTEA, XXTEA and Twofish from symmetric algorithms; CMAC, HMAC-SHA1, HMAC-SHA256 and UMAC from MAC algorithms; ECC and RSA from asymmetric algorithms.

We evaluate all candidates in our platform STM32F103ZE development board from IAR System using pure software implementation. In order to make our evaluation more accurate, we carefully choose the methods to measure footprint and time consumption. All software implementations are optimized version adapted from open source projects or security libraries. Finally, we presented the results of our evaluation and gave comparisons of footprint and performance of all candidates. From the results, we notice that it is impossible to select one most suitable algorithm for all situations. Different algorithms have different advantages and disadvantages. For symmetric algorithms, our surprising result is that Twofish has better performance than AES both in the speed of secret key setup and encryption. XTEA and XXTEA have extremely small footprint than any other algorithm. XTEA even has similar performance as AES, although the security of XTEA is much weaker than AES. Therefore, in our platform, Twofish is highly recommended when the performance of algorithms is prior to footprint. XTEA or XXTEA is recommended in applications where memory is strictly constrained and security is not. Since the performance of symmetric encryption using four operating modes are quite similar to each other except ECB, the selection of encryption mode should be based on application requirements, such as processing in parallel or security, rather than encryption speed. For MAC algorithms, UMAC is an obvious state-of-the-art algorithm. It has outstanding performance for both short and long messages compared to other MAC algorithms. The disadvantages of UMAC are the key setup stage and the footprint. CMAC-AES is very efficient for short messages, but extremely inefficient for long messages. Therefore, our suggestion is that CMAC-AES algorithm can be applied when messages are less than 64 bytes and AES is already implemented in the systems. UMAC is highly recommended for new applications except for memory constrained devices. For asymmetric algorithms, both RSA and ECC perform much slower than symmetric and MAC algorithms. ECC has acceptable performance in key generation, encryption and digital signature compared to RSA. Therefore, when an asymmetric algorithm is required in a new application, ECC should be chosen prior to RSA at the same security level.

6.2 Future Work

In this thesis, evaluation and comparison of cryptographic algorithms are based on software implementation using pure C programming. Several patented algorithms and newly proposed algorithms which are also considered to have excellent performance, such as Camellia, NTUR, HECC, are not included in our evaluation. Future work can include these algorithms in evaluation to compare with our results. Except for pure C program software implementation, cryptographic algorithms can also be implemented via other methods to improve performance, such as hardware implementation. It is also interesting to evaluate to what extent the performance of these cryptographic algorithms can be improved using these two implementation methods.

7 REFERENCES

[1] ”Introduction to Cryptography with Coding Theory”, 2nd edition, Wade Trappe, Lawrence C. Washington, Pearson 2006. ISBN 0-13-198199-4.

[2] Raymond R. Panko. *Corporate Computer and Network Security*, Second Edition, ISBN 0-13-612157-8.

[3] A. S. Tanenbaum. *Computer Network, Fourth Edition*. Pearson Education International, 2003.

[4] William Stallings, *Cryptography and Network Security 4th Edition*, Prentice Hall November 26, 2005

[5] Kahn, David (1967). *The Codebreakers*. pp. 631–2. ISBN 978-0-684-83130-5.

[6] NESSIE CONSORTIUM 2003. *Portfolio of recommended cryptographic primitives*. NESSIE Consortium.

[7] NECHVATAL, J., BARKER, E., BASSHAM, L., BURR, W., DWORKIN, M., FOTI,J., AND ROBACK, E. 2000. *Report on the Development of the Advanced Encryption Standard (AES)*. Tech. rep., NIST.

[8] Elias Yarrkov, *Cryptanalysis of XXTEA*, May 4, 2010: http://eprint.iacr.org/2010/254.pdf, 2010-05-10

[9] L.R.Knudsen, M.J.B. Robshaw, D. Wagner. *Truncated differentials and Skipjack*. CRYPTO 1999.

[10] Onur Özen, Kerem Varıcı, Cihangir Tezcan and Çelebi Kocair. *Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT*. Lecture Notes in Computer Science, ISBN 978-3-642-02619-5

[11] Krzysztof Pietrzak. *A Tight Bound for EMAC*. Automata, Languages and Programming. ISBN 978-3-540-35907-4.

[12] Erez Petrank and Charles Rackoff. *CBC MAC for Real-Time Data Sources*. Journal of Computer and System Sciences, pages 315–338, 2000.

[13] John Black, Phillip Rogaway. *CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions*. In Bellare, M., ed.: Advances in Cryptology — Crypto 2000.

[14] Chris J. Mitchell. *Partial Key Recovery Attacks on XCBC, TMAC and OMAC*. Cryptography and Coding.

[15] Tetsu Iwata, Kaoru Kurosawa. *Stronger Security Bounds for OMAC, TMAC, and XCBC*. Progress in Cryptology - INDOCRYPT 2003.

[16] Lars R. Knudsen, Tadayoshi Kohno. *Analysis of RMAC*. Fast Software Encryption.

[17] M. Bellare, R. Canetti, and H. Krawczyk. *Keying hash functions for message authentication*. CRYPTO 96.

[18] Bert den Boer, Bart Van Rompay , Bart Preneel and Joos Vandewalle. *New (Two-Track-) MAC Based on the Two Trails of RIPEMD*. Selected Areas in Cryptography

[19] Xiaoyun Wang, Hongbo Yu. *How to Break MD5 and Other Hash Functions*. Retrieved December 21, 2009

[20] Whirlpool homepage. http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html, 2010-04-12

[21] M. Bellare, J. Kilian, and P. Rogaway. *The security of the cipher block chaining message authentication code.* JCSS, vol. 61, no. 3, 2000. Earlier version in Advances in Cryptology — CRYPTO '94, LNCS 839, pp. 341–358, Springer-Verlag, 1994.

[22] NESSIE consortium. *Security Report, Version 2.0*, February 19, 2003.

[23] J. Sung, D. Hong and S. Lee. *Key Recovery Attacks on the RMAC, TMAC, and IACBC*. ACISP 2003, LNCS 2727, pp. 265-273, Springer-Verlag, 2003.

[24] C. Mitchell. *On the security of XCBC, TMAC, and OMAC*. Technical Report RHCL-MA-2003-4, Aug. 2003.

[25] Changhoon Lee, Jongsung Kim, Jaechul Sung, Seokhie Hong and Sangjin Lee. *Forgery and Key Recovery Attacks on PMAC and Mitchell's TMAC Variant*. Information Security and Privacy.

[26] Tetsu Iwata. *Comments on "On the security of XCBC, TMAC and OMAC" by Mitchell*. September 19, 2003.

[27] Somitra Kumar Sanadhya, Palash Sarkar. *New Collision Attacks against Up to 24-Step SHA-2*. Progress in Cryptology - INDOCRYPT 2008

[28] Hongbo Yu, Xiaoyun Wang. *Cryptanalysis of the Full HAVAL with 4 and 5 Passes*. Fast Software Encryption.

[29] Lin Li, Wang Xiaoyun. *Cryptanalysis of the Hash Functions RIPEMD-128 and HMAC-MD4*. Doctor dissertation, Shandong University (in Chinese).

[30] The RIPEMD-160 page: http://homes.esat.kuleuven.be/~bosselae/ripemd160.html, 2010-04-20

[31] Vincent Rijmen, Paulo S. L. M. Barreto. *The Whirlpool Hashing Function*. 2003-05-24

[32] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, Martin Schläffer. *Rebound Distinguishers: Results on the Full Whirlpool Compression Function*. Advances in Cryptology – ASIACRYPT 2009.

[33] Ted Krovetz. *UMAC: Message Authentication Code using Universal Hashing*, March 2006. RFC 4418, http://fastcrypto.org/umac/rfc4418.txt

[34] Jongsung Kim, Alex Biryukov, Bart Preneel, and Seokhie Hong. *On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1 (extended*

abstract). In Roberto De Prisco and Moti Yung, editors, SCN 2006, volume 4116 of Lecture Notes in Computer Science, pages 242–256. Springer, 2006.

[35] Scott Contini and Yiqun Lisa Yin. *Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions*. ASIACRYPT2006, volume 4284 of Lecture Notes in Computer Science, pages 37–53. Springer, 2006.

[36] NESSIE consortium. *Performance Report, Version 2.0*. February 19, 2003.

[37] Jeffrey Hoffstein, Jill Pipher and Joseph H. Silverman. *NTRU: A ring-based public key cryptosystem*. Algorithmic Number Theory, Volume 1423/1998.

[38] Jasper Scholten and Frederik Vercauteren. *An Introduction to Elliptic and Hyperelliptic Curve Cryptography and the NTRU Cryptosystem*. http://homes.esat.kuleuven.be/~fvercaut/papers/cc03.pdf , 2010-03-12

[39] Michael Jacobson, Jr., Alfred Menezes, Andreas Stein. *Hyperelliptic Curves and Cryptography*. Fields Institute Communications, 2004.

[40] Martin Hlaváč. *Known–Plaintext–Only Attack on RSA–CRT with Montgomery Multiplication*. Cryptographic Hardware and Embedded Systems - CHES 2009.

[41] M. O. Rabin. Digitalized signatures and public key functions as intractable as factorization. MIT/LCS/TR-212, Massachusetts Institute of Technology, Jan 1979.

[42] NTRU Cryptosystems. *Peer Review and Independent Scrutiny of the NTRUEncrypt Public Key Cryptosystem*, 2004. http://www.ntru.com/cryptolab/pdf/review.pdf, 2010-04-16.

[43] P. Gaudry. *An algorithm for solving the discrete log problem on Hyperelliptic curves*. In B. Preneel, editor, Advances in Cryptology: Proceedings of EUROCRYPT2000, volume 1807 of LNCS, pages 19–34, 2000.

[44] Li T., Wu H., Wang X., Bao F. *SenSec Design. Technical Report-TR v1.1*. InfoComm Security Department, Institute for Infocomm Research (2005)

[45] Karlof C., Sastry N., Wagner D. *TinySec: a Link Layer Security Architecture for Wireless Sensor Networks*. Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), pp. 162--175. Baltimore (2004)

[46] Luk, M., Mezzour, G., Perrig, A., Gligor, V. *MiniSec: a Secure Sensor Network Communication Architecture*. Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN 2007), pp. 479-488. ACM, New York (2007)

[47] SANO, F., KOIKE, M., KAWAMURA,S., AND SHIBA,M. 2001. *Performance Evaluation of AES Finalists on the High-End Smart Card.* In Proceedings of the 3rd AES Conference (AES3).

[48] ARM website: www.arm.com

[49] Implementation of SHA-224, SHA-256, SHA-384 and SHA-512: http://www.ouah.org/ogay/sha2/, 2010-04-03

[50] Johan Akerberg. *On Security in Safety-Critical Process Control*. Malardalen University Press Licentiate Theses No. 110.

[51] D. Dzung, M. Naedele, T.P. Von Hoff and M. Crevatin. *Security for Industrial Communication Systems*. Proceedings of the IEEE, 93(6): 1152-1177, June 2005.

[52] Bruce Schneier. *New Attack on AES*. Schneier on Security, A blog covering security and security technology.
http://www.schneier.com/blog/archives/2009/07/new_attack_on_a.html.

[53] Henri Gilbert; Thomas Peyrin. *Super-Sbox Cryptanalysis: Improved Attacks for AES-like permutations*. http://eprint.iacr.org/2009/531. Retrieved 2010-03-11.

[54] Mihir Bellare and Phillip Rogaway. *Introduction to Modern Cryptography*. Course notes for UCSD course CSE207.

[55] Philip Rogaway. The Security of DESX. Department of Computer Science, University of California, Davis, CA 95616 USA, DRAFT 2.0, July 5 1996.

[56] Sean Murphy. University of London. http://www.isg.rhul.ac.uk/~sean/. Retrieved 2008-11-02.

[57] Bruce Schneier. *AES News, Crypto-Gram Newsletter, September 15, 2002*. http://www.schneier.com/crypto-gram-0209.html. Retrieved 2007-07-27.

[58] B. Schneier. *Description of a New Variable-Length Key, 64-Bit Block Cipher*. http://www.schneier.com/paper-blowfish-fse.html

[59] Serge Vaudenay (1996). *On the Weak Keys of Blowfish*. http://lasecwww.epfl.ch/php_code/publications/search.php?ref=Vau96a Retrieved 2006-08-23.

[60] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. *Handbook of Applied Cryptography*. Fifth Printing (August 2001)

[61] Shiho Moriai, Kazumaro Aoki, and Kazuo Ohta. *The best linear expression search of FEAL*. IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan), Vol. E79-A, No. 1, pp.2--11, 1996 (The extended abstract was presented at CRYPTO'95).

[62] Kazumaro Aoki. *On cryptanalysis with impossible differentials*. In 1999 Symposium on Cryptography and Information Security, number T4-1.3 in SCIS'99, International Conference Center Kobe, Kobe, Japan, 1999. Technical Group on Information Security (IEICE). (in Japanese).

[63] Kelsey, John; Schneier, Bruce; Wagner, David (1996). *Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES*. Lecture Notes in Computer Science 1109: 237–251.

[64] Abdel-Karim Al Tamimi. *Performance Analysis of Data Encryption Algorithms*. http://www.cs.wustl.edu/~jain/cse567-06/ftp/encryption_perf/index.html#3

[65] Bart Preneel, Katholieke Universiteit Leuven. *Software performance of encryption algorithms and hash functions.* ESAT-COSIC, Kardinaal Mercierlaan 94, B–3001 Heverlee, Belgium

[66] Ying-yu Cao; Chong Fu. *An Efficient Implementation of RSA Digital Signature Algorithm*. Intelligent Computation Technology and Automation (ICICTA), 2008 International Conference on Volume 2, 20-22 Oct. 2008 Page(s):100 – 103.

[67] Garg, D.; Verma, S. *Improvement over Public Key Cryptographic Algorithm*. Advance Computing Conference, 2009. IACC 2009. IEEE International 6-7 March 2009 Page(s):734 - 739 Digital Object Identifier 10.1109/IADCC.2009.4809104

[68] A.A. Mamun, M. M. Islam, S.M. M. Romman, A.H.S.U Ahmad. *Performance Evaluation of Several Efficient RSA Variants.* IJCSNS VOL.8 No.7, July 2008, pp. 7-l1.

[69] Douglas R. Stinson. *Cryptography: Theory and Practice, Second Edition.* ISBN 7-5053-8465-1.

[70] I. Branovic, R. Giorgi, E. Martinelli. *A workload characterization of elliptic curve cryptography methods in embedded environments*. June 2004 MEDEA '03: Proceedings of the 2003 workshop on MEmory performance: Dealing with Applications , systems and architecture

[71] N. Gura, A. Patel, A. Wander, H. Eberle, S. C. Shantz. *Comparing elliptic curve cryptography and RSA on 8-bit CPUs*. Proceedings of Cryptographic Hardware and Embedded Systems (CHES), pp. 119-132, 2004.

[72] Hai Yan; Zhijie Jerry Shi. *Studying Software Implementations of Elliptic Curve Cryptography*. Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on 10-12 April 2006 Page(s):78 – 83

[73] I. Blake, G.Seroussi, N.Smart. *Elliptic Curves in Cryptography*. Cambridge Univ. Press, 1999

[74] L.Washington. *Elliptic Curves: Number Theory and Cryptography*. Chapman & Hall/ CRC Press, 2003

[75] Michael Brown,Darrel Hankerson, Julio López and Alfred Menezes. *Software Implementation of the NIST Elliptic Curves over Prime Fields.* CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA , April 2001.

[76] Sam Simpson. “PGP DH vs. RSA FAQ. 1999, http:// www.scramdisk.clara.net/pgpfaq.html#SubDH

[77] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. *Attacking and defending the McEliece cryptosystem*. Department of Mathematics, Statistics, and Computer

Science (M/C 249), University of Illinois at Chicago, Department of Mathematics and Computer Science, echnische Universiteit Eindhoven.

[78] Lejla Batina,DavidHwang, Alireza Hodjat, Bart Prenee, and Ingrid Verbauwhede. *Hardware/Software Co-design for Hyperelliptic Curve Cryptography (HECC) on the 8051 µP*. University of California, El. Engineering Dept., Los Angeles, CA 90095, Katholieke Universiteit Leuven, ESAT/COSIC, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium

[79] T. Wollinger, J. Pelzl, V. Wittelsberger, C. Paar, G. Saldamli, and C ¸. Ko¸ c. *Elliptic and hyperelliptic curves on embedded µP.* ACM Transactions on Embedded Computing Systems, 3(3):509–533, 2004.

[80] D. Bailey, D. Coffin, A. Elbirt, J. Silverman, A. Woodbury. *NTRU in Constrained Devices*. Proc. Cryptographic Hardware and Embedded Systems, Paris, France, 2001

[81] Jens-Peter Kaps. *Cryptography for Ultra-Low Power Devices*. A Dissertation, WORCESTER POLYTECHNIC INSTITUTE, In partial fulfillment of the requirements for the Degree of Doctor of Philosopy in Electrical Engineering

[82] Rodrigo Roman, Cristina Alcaraz , Javier Lopez. *A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes*. Mobile Networks and Applications archive Volume 12 , Issue 4 (August 2007) Pages: 231 - 244 Year of Publication: 2007

[83] J. Black, S. Halevi, H. Krawczyk, T. Krovetz and P. Rogaway. *UMAC: Fast and Secure Message Authentication.* Lecture Notes in Computer Science, Advances in Cryptology — CRYPTO' 99

[84] John Black, Phillip Rogaway. *A Suggestion for Handling Arbitrary-Length Messages with the CBC MAC.* NIST Second Modes of Operation Workshop, August 2001.

[85] Phil Rogaway's page on PMAC, http://www.cs.ucdavis.edu/~rogaway/ocb/pmac.htm, the inventor of PMAC, 2010-05-12.

[86] J. Black, P. Rogaway. *A Block-Cipher Mode of Operation for Parallelizable Message Authentication.* Advances in Cryptology – Eurocrypt '02, February 15, 2002

[87] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication.* NIST Special Publication 800-38B.

[88] Tetsu Iwata. *Comparison of CBC MAC Variants and Comments on NIST's Consultation Paper*. Department of Computer and Information Sciences, Ibaraki University 4–12–1 Nakanarusawa, Hitachi, Ibaraki 316-8511, Japan.

[89] Tetsu Iwata and Kaoru Kurosawa. *OMAC: One-Key CBC MAC*. Department of Computer and Information Sciences, Ibaraki University, 4–12–1 Nakanarusawa, Hitachi, Ibaraki 316-8511, Japan

[90] Tetsu Iwata and Kaoru Kurosawa. *TMAC: Two-Key CBC MAC*. Department of Computer and Information Sciences, Ibaraki University, 4–12–1 Nakanarusawa, Hitachi, Ibaraki 316-8511, Japan

[91] Daniel J. Bernstein. *ThePoly1305-AES Message-Authentication Code*. Department of Mathematics, Statistics, and Computer Science, the University of Illinois at Chicago.

[92] UMAC webpage: http://fastcrypto.org/umac/

[93] Yuliang Zheng, Josef Pieprzyk and Jennifer Seberry. *HAVAL—A One-Way Hashing Algorithm with Variable Length of Output (Extended Abstract)*. Appeared in "Advances in Cryptology—AUSCRYPT'92," Lecture Notes in ComputerScience, Vol. 718, pp.83-104, Springer-Verlag, 1993.

[94] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu. *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD.* The School of Mathematics and System Science, Shandong University, Jinan250100, Institute of Software, Chinese Academy of Sciences, Beijing100080, Dept. of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai, China

[95] Hans Dobbertin, Antoon Bosselaers and Bart Preneel. *RIPEMD-160: A Strengthened Version of RIPEMD.* Lecture Notes in Computer Science, Fast Software Encryption, DOI: 10.1007/3-540-60865-6_44

[96] Yee Wei Law, Jeroen Doumen, Pieter Hartel. *Survey and benchmark of block ciphers for wireless sensor networks*. Transactions on Sensor Networks (TOSN) , Volume 2 Issue 1, February 2006

[97] Shiho Moriai, Yiqun Lisa Yin. *Cryptanalysis of Twofish (II)*. TECHNICAL REPORT OF IEICE, 2000

[98] Sören Rinne,Thomas Eisenbarth and Christof Paar. *Performance Analysis of Contemporary Light-Weight Block Ciphers on 8-bit Microcontrollers*. Horst Görtz Institute for IT Security Ruhr University, Bochum 44780 Bochum, Germany

[99] A.Poschmann, G.Leander, K.Schramm and C.Paar. *New Light-Weight DES Variants Suited for RFID Applications.* In Proceedings of FSE 2007, 2007.

[100] D. Hong et. al. *HIGHT: A New Block Cipher Suitable for Low-Resource Device*. In Proceedings of CHES2006, 2006.

[101] F. X. Standaert, G. Piret, N. Gershenfeld, and J. J. Quisquater. *SEA: A Scalable Encryption Algorithm for Small Embedded Applications*. Workshop on RFIP and Light weight Crypto, Graz, Austria, 2005.

[102] Thomas Eisenbarth, Christof Paar, Axel Poschmann, Sandeep Kumar, Leif Uhsadel. *A Survey of Light weight Cryptography Implementations.* Design and Test of ICs for Secure Embedded Computing.

[103] Bruce Schneier. *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*. Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204.

[104] Wheeler, David J.; Needham, Roger M. *TEA, a tiny encryption algorithm.* Lecture Notes in Computer Science (Leuven, Belgium: Fast Software Encryption: Second International Workshop) 1008: 363–366.

[105] R. Shirey. *Internet Security Glossary*. RFC 2828, May 2000

[106] J. Kohl, C. Neuman. *The Kerberos Network Authentication Service (V5)*. RFC 1510, September 1993

[107] S. Kent, R. Atkinson. *Security Architecture for the Internet Protocol*. RFC 2401, November 1998

[108] S. Kent, R. Atkinson. *IP Authentication Header*. RFC 2402, November 1998

[109] S. Kent, R. Atkinson. *IP Encapsulating Security Payload (ESP)*. RFC 2406, November 1998

[110] D. Maughan, M. Schertler, M. Schneider and J. Turner. *Internet Security Association and Key Management Protocol (ISAKMP).* RFC 2408, November 1998

[111] R. Rivest. *The MD5 message-digest algorithm*. RFC 1321, April 1992.

[112] JH. Song, R. Poovendran, J. Lee, T. Iwata. *The AES-CMAC Algorithm*. RFC 4493, June 2006

[113] P. Cheng, R. Glenn. *Test Cases for HMAC-MD5 and HMAC-SHA-1.* RFC 2202, September 1997.

[114] Chris Karlof, Naveen Sastry, David Wagner. *TinySec: A link layer security architecture for wireless sensor networks*. In SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems. ACM Press, New York, NY, USA, 162–175.

[115] M. Bellare and P. Rogaway. *Optimal asymmetric encryption*. Advances in Cryptology - Eurocrypt'94, Lecture Notes in Computer Science, volume 950, pages 92--111.Springer Verlag, 1994.

[116] James Nechvatal,Elaine Barker, Lawrence Bassham, William Burr,Morris Dworkin, James Foti, Edward Roback. *Report on the Development of the Advanced Encryption Standard (AES)*. Publication Date: October 2, 2000

[117] Wireless Systems for Industrial Automation: Process Control and Related Applications, International Society of Automation (ISA) Standard 100.11a, Draft 2a, 2009.

[118] HCF WirelessHART Specification