Examensarbete

Implementing the Transport Layer Security Protocol for Embedded Systems

Examensarbete utfört i Informationsteori vid Tekniska högskolan i Linköping
av

Bengt Werstén

LITH-ISY-EX--06/3985--SE

Linköping 2007
Implementing the Transport Layer Security Protocol for Embedded Systems

Examensarbete utfört i Informationsteori
vid Tekniska högskolan i Linköping
av

Bengt Werstén

LITH-ISY-EX--06/3985--SE

Handledare: Viiveke Fåk
isy, Linköpings universitet
Mattias Svanström
Enea Services Linköping

Examinator: Viiveke Fåk
isy, Linköpings universitet

Linköping, 16 May, 2007
Implementation och anpassning av Transport Layer Security för inbyggda system

Bengt Werstén

Web servers are increasingly being used in embedded devices as a communication medium. As more systems connect to the Internet, the need for security is increasing. The Transport Layer Protocol (TLS) is the successor of Secure Socket Layer (SSL) and provides security in almost all secure Internet transactions. This thesis aims to investigate if TLS can be adapted to embedded systems without sacrificing much of the system resources available.

A literature study and an implementation of TLS have been performed. The literature study determined resource intense parts of TLS, hardware support as well as export laws applicable to TLS. The different parts of the implementation are evaluated on an ARM7-core to determine the execution times. The results for the symmetric ciphers AES and 3DES are compared when measuring execution times using both software and hardware solutions. The size of the implementation is also measured.

TLS was shown to be able to integrate on embedded systems. Practical issues such as certificates and keys can be solved in different ways to suite the target environment. The largest remaining issue is the execution time for asymmetric algorithms. The results that are provided clearly illustrates that the RSA used for key exchange is very time consuming. Alternative solutions to gain better performance are discussed.
Abstract

Web servers are increasingly being used in embedded devices as a communication medium. As more systems connect to the Internet, the need for security is increasing. The Transport Layer Protocol (TLS) is the successor of Secure Socket Layer (SSL) and provides security in almost all secure Internet transactions. This thesis aims to investigate if TLS can be adapted to embedded systems without sacrificing much of the system resources available.

A literature study and an implementation of TLS have been performed. The literature study determined resource intense parts of TLS, hardware support as well as export laws applicable to TLS. The different parts of the implementation are evaluated on an ARM7-core to determine the execution times. The results for the symmetric ciphers AES and 3DES are compared when measuring execution times using both software and hardware solutions. The size of the implementation is also measured.

TLS was shown to be able to integrate on embedded systems. Practical issues such as certificates and keys can be solved in different ways to suite the target environment. The largest remaining issue is the execution time for asymmetric algorithms. The results that are provided clearly illustrates that the RSA used for key exchange is very time consuming. Alternative solutions to gain better performance are discussed.
Acknowledgments

I would like to take this opportunity to thank a lot of people granting me a great master thesis work experience.

First of all I would like to thank my supervisor at Enea, Mattias Svanström. Without him this project never would have come through. I would also like to thank my supervisor and examiner at ISY, Viiveke Fåk, who supported me with feedback and for always keeping a positive attitude.

A big thanks goes to all the people working at Enea in Linköping. I would especially like to thank Matthias Bergvall, for always answering my questions and giving me good advice.

Finally, I thank my family and friends for always supporting me.

Bengt Werstén
# Contents

1 Introduction .................................................. 1
   1.1 Background ................................................. 1
   1.2 Objectives .................................................. 1
   1.3 Limitations .................................................. 2
   1.4 Method ....................................................... 2
   1.5 Thesis disposition ......................................... 3
   1.6 Notes ....................................................... 3

2 Basics of Cryptography ........................................ 5
   2.1 Algorithms .................................................. 5
       2.1.1 Symmetric-key Cryptography ......................... 5
       2.1.2 Public-key Cryptography ............................ 7
   2.2 Digital Certificate ......................................... 9
   2.3 Hash Functions ............................................. 9
   2.4 Cryptanalysis ............................................... 10

3 The TLS Protocol ............................................... 11
   3.1 Introduction ............................................... 11
       3.1.1 Basics .................................................. 11
   3.2 Structure .................................................. 12
       3.2.1 Handshake Protocol ................................... 13
       3.2.2 Change Cipher Spec Protocol ....................... 14
       3.2.3 Application Data Protocol .......................... 15
       3.2.4 Alert Protocol ....................................... 15
   3.3 Session resumption ......................................... 15
   3.4 HTTPS ...................................................... 15

4 Export of Cryptography ....................................... 17
   4.1 International regulations .................................. 17
       4.1.1 Wassenaar Arrangement .............................. 17
       4.1.2 OECD .................................................. 18
       4.1.3 United States ......................................... 18
       4.1.4 European Union ...................................... 19
   4.2 Export Control on the TLS Protocol ....................... 19
Chapter 1

Introduction

This chapter gives an introduction to this master thesis. The objectives and the methods are described. Limitations are discussed, and the thesis disposition is presented.

1.1 Background

Embedded devices are increasingly being equipped with Internet accessible features such as web servers. When more embedded systems go online, the demand for security is greater than ever. Commonly used protocols to secure Internet transactions are the Secure Sockets Layer (SSL) and its successor, the Transport Layer Security (TLS).

The TLS protocol has become a standard and is supported in nearly every modern web browser. TLS is designed for maximum security and have large resource requirements.

1.2 Objectives

The main objective is to implement the TLS protocol in such a way that it can be used on a resource-limited system. The evaluation system should use an Advanced RISC Machine (ARM) CPU core and run the operating system Operating System Embedded (OSE). The implementation should use as few resources as possible and leave a small footprint. To test the system, a minimal web server should run on the target responding to request made by a modern web browser such as Mozilla Firefox. The implementation should be investigated and discussed to cover the following questions:

- What features of TLS can be left out while still supporting the specification?
- What are the most important design decisions that will significantly alter system resource utilization?
- Are there alternative solutions to previously implemented solutions that might suite other systems?
The offered security of the implementation should be discussed and followed by suggested improvements. There is a growing market for CPU cores with built-in hardware support for different cryptographic features. Some of these techniques should be investigated and evaluated to measure the gain in performance and security. The results should be compared with software solutions built for the same system.

This thesis should also contain a brief discussion about export laws of cryptographic systems in different countries.

1.3 Limitations

Since nearly every embedded system function as a server and is connected by common client software, only the server part of the TLS protocol should be implemented. Every feature and cryptographic algorithm in the TLS specification should not be implemented and evaluated. The most common cryptographic algorithms should be covered.

1.4 Method

The work starts with theoretical studies of the TLS protocol to understand what is necessary and how it could be implemented. This includes the message system and the different types of cipher suites the protocol uses. This study also has the purpose of locating parts that are resource-intensive and therefore subject to optimization. The theoretical studies also covers current export regulations in the U.S. and the EU.

The next part is the actual implementation at the target in use. The main objective of this part is a working system within the resources available in the target system.

The last part is an attempt to improve the working implementation. The objective is to reduce memory and computation usage as much as possible. Hardware solutions are compared with software solutions with focus on memory usage and execution time.
1.5 Thesis disposition

This section contains a brief description of every chapter and appendix in the thesis.

- Chapter 1, Introduction, gives a short introduction of the objectives, method and limitations of the thesis.

- Chapter 2, Cryptographic Basics, presents an introduction to cryptography to the reader. The chapter describes different algorithms and some computer security fundamentals.

- Chapter 3, The TLS Protocol, describes the TLS protocol. The purpose, functionality and structure are presented.

- Chapter 4, Export of Cryptography, presents some cryptographic export regulations. It will also describe how export controls apply to the TLS protocol.

- Chapter 5, Related Work, contains information on related work that has been done in this area.

- Chapter 6, Development Environment, gives a brief description of the tools used when developing.

- Chapter 7, Implementation Considerations, analysis and discusses different techniques used when implementing the system to save computer resources.

- Chapter 8, Hardware Support, contains a short investigation of currently available hardware support for cryptographic computations. A selected device is evaluated.

- Chapter 9, Analysis and Results, presents the results from the evaluation on the target device.

- Chapter 10, Conclusions and Further Studies, contains a final conclusion and suggestions of further studies.

- Appendix A, Dictionary, contains an alphabetized list of explained abbreviations used in this thesis.

1.6 Notes

A list with explanations of all abbreviations can be found in appendix A. Common terms are explained in the related section when introduced.
Chapter 2

Basics of Cryptography

The term cryptography is derived from the two Greek words kryptós (“secret”) and gráfo (“write”) and is the science of message security. It is an ancient mathematical subject and has only recently become a branch of information security. By using a key it is possible to use a cipher to transform a readable message, plaintext, into an unreadable message, ciphertext. This procedure is known as encryption. The very same key or a related key can be used to do the inverse transformation, decryption. [12, 18]

This chapter will introduce different cryptographic algorithms and describe some computer security fundamentals.

2.1 Algorithms

Algorithms used for encryption/decryption fall into two categories: symmetric-key and public-key cryptography. [36]

2.1.1 Symmetric-key Cryptography

In symmetric-key algorithms, the sender and receiver have the same set of keys and use the same algorithms to encrypt and decrypt messages. The keys used for encryption and decryption can differ but there is a transform to go from one key to the other. If a message in plaintext \( m \) shall be transmitted from Alice to Bob they have to agree on a key \( k \). Alice begins by encrypting the message \( m \) to \( c \) using a one-to-one mapping function \( E_k \):

\[
c = E_k(m).
\]  

(2.1)

The ciphertext message \( c \) will now be sent over the network to Bob. Bob uses \( D_k \), the inverse function of \( E_k \), on the message \( c \) to transform it back to the original plaintext message \( m \):

\[
D_k(c) = D_k(E_k(m)) = E_k^{-1}(E_k(m)) = m.
\]  

(2.2)
A general outline of this concept is shown in Figure 2.1. [34, 36]

Known symmetric-key algorithms that are subjects of investigation in this thesis are Rivest Cipher 4 (RC4), Triple Data Encryption Standard (3DES) and Advanced Encryption Standard (AES).

RC4 is a stream cipher designed as a trade secret by RSA Security in 1987 by Ron Rivest. A stream cipher encrypts and decrypts one character, or byte, at a time while a block cipher works on blocks of bytes. The RC4 algorithm was later anonymously posted on the Internet. The RC4 algorithm is usually implemented under the name Alleged RC4 (ARC4) to avoid trademark and copyright issues. ARC4 supports different key lengths. 40-bit and 128-bit keys are specifically supported and the algorithm can be written in only few lines of C code. It is recommended to change the key frequently to maintain sufficient security. It does contain several weaknesses in its key-scheduling algorithm further described in [17]. [17, 18]

3DES is a block cipher that uses the Data Encryption Standard (DES) algorithm three times. The DES itself was developed in the 1970s and is a frequently used cipher. The algorithm uses 56 bits for the key and is believed to be vulnerable only to brute-force attacks. In 3DES, the original message is first encrypted with one key. The encrypted output data from the first step is decrypted using a second key. The output data is then encrypted using a third key in the last step. Note that the output data from the second step will be in plaintext if the same keys are used in the first two steps. A common approach is to use the same key a in the first and third step and a different key b in the second step. This will only produce 112 bits of security instead of the maximum 168 bits but is fully compatible with DES when a is equal to b. [18]

AES is the successor of 3DES and is also a block cipher. It is the result of a competition created by the National Institute of Standards and Technology (NIST) in 2001. AES is based on a winning algorithm called Rijndael. AES only supports key sizes of 128, 192 and 256 bits while Rijndael can handle additional key sizes. Every block of data will be transformed in four steps and this procedure is called a round. The number of rounds is dependent of the key size and is not fixed as for 3DES. The final round is different and excludes one of the steps. [18, 23]
Block Cipher Modes of Operation

A mode is a combination of a cipher and preceding or postceding operations. The basic mode for a block cipher is called Electronic Codebook (ECB). Every block is encrypted without any extra operations and independently of other blocks. This is not safe if the message contains many blocks of the same content generating a ciphertext where these part can be directly located. To prevent this behavior every block can be made dependent of the last encrypted block. In Cipher Block Chaining (CBC), each block is XOR’ed with the last encrypted block before encryption. This requires an extra value of the same size as one block to use with the first encrypted block. This value is called Initiation Vector (IV). Another mode similar to CBC is Cipher Feedback (CFB). CFB starts by encrypting the IV and XOR the result with a plaintext block. The output block is used as the new input IV for the next block. This does not require the entire block to encrypt or decrypt. One bit can be processed at the time. A modification of CFB by using the encrypted output instead of the resulting ciphertext as new input is called Output Feedback (OFB). The advantage is that the same algorithm can be used for both encryption and decryption. The differences between CBC, CFB and OFB is illustrated in Figure 2.2. Another way resulting in a block cipher working as a stream cipher without any IV is to use the mode called Counter (CTR). This is similar to OFB but uses a counter instead of the encrypted output as the new input. [28]

![Figure 2.2. Block cipher modes of operation.](image)

2.1.2 Public-key Cryptography

Public-key cryptography is also known as asymmetric cryptography. Different keys are used in each direction. The sender and receiver both have two keys. One is public and is known by all and one is private and is only known by the keeper. This form of cryptography was introduced in the 1970s and solves some of the problems with symmetric-keys. [12, 36]

When Alice and Bob wants to communicate in a safe way using a symmetric algorithm they have to share and know the key before communication starts.
If Alice and Bob have never met and determined a key, communication is not possible. A public key algorithm solves this. Alice can send her public key $n$ in plaintext to Bob. Bob uses this key and encrypts a message with the public key of Alice and sends this to Alice. This message is asymmetric since it cannot be decrypted using the public key used for encryption. It can only be decrypted using the private key $d$. Public-key algorithms can be used for key exchange between Alice and Bob as shown in Figure 2.3 on page 8. In response to the plaintext message sent by Alice, Bob generates and sends a key encrypted with the public key from Alice. Only Alice can decrypt this message and a symmetric method can further on be used with the generated key from Bob. [36]

![Diagram of public-key cryptography](image)

**Figure 2.3.** Public-key cryptography.

Since public-key methods are much slower than the symmetric counterpart they are well suited for short messages such as a key exchange to be used in symmetric algorithms. [36]

When the public and private key in an asymmetric algorithm are interchangeable (i.e. an encryption with the private key can only be decrypted using the public key), it can be used for authentication. One well-known algorithm that can be used for both key exchange and authentication is RSA. [12]

Ron Rivest, Adi Shamir and Len Adleman published RSA in 1977. An equivalent system was described in a document dated 1973 by the mathematician Clifford Cocks. This was an internal document owned by a British cryptographic agency and was not released until 1997. [36]

The first step is to create a set of keys. It begins by selecting two large prime numbers $p$ and $q$ and computing

$$n = pq.$$  \tag{2.3}

Choose an $e$ that satisfies

$$gcd(e, (p - 1)(q - 1)) = 1.$$  \tag{2.4}

Compute the unique value $d$ such that

$$de = 1 \pmod{(p - 1)(q - 1)}$$  \tag{2.5}

The set $(n, e)$ is the public key and $(n, d)$ is the private key. A message $m$ can be encrypted with the public set to create a ciphertext $c$

$$c = m^e \pmod{n}$$  \tag{2.6}
The decryption uses the private set to recreate the plaintext \( m \)

\[
m = c^d \pmod{n}
\]

(2.7)

To ensure high security, RSA must use big key lengths of at least 1024 bits. This requires special libraries to support mathematical operations on large numbers when implementing RSA support. [22]

### 2.2 Digital Certificate

To verify that a certain public key belongs to an identity, a digital certificate can be used. A digital certificate binds a public key to an identity. To be able to trust that the sender of a certificate is the real owner, a third party needs to be involved. A trusted third party can sign a certificate by encrypting it with a private key. The third part is usually a Certificate Authority (CA). A CA is an organization that provides signing of certificates. It is also possible for the certificate owner to sign the certificate himself. A signed certificate contains an encrypted checksum, signature. A commonly used certificate format is the Directory Authentication Framework (X.509). [12]

X.509 defines what should be present in a certificate. This includes the identity, public key information of the issuer and owner and validity interval. It is validated by obtaining the public key of the issuer and decrypting the signature. A hash (see section 2.3) of all other data is calculated and should match the decrypted signature. The validity interval is checked to ensure that this certificate has not expired. The user can now use the public key of the certificate owner to continue the communication. [12]

### 2.3 Hash Functions

A hash function takes an input of arbitrary size and produces an output, a message digest or checksum, of fixed size. The most important property of a hash function is that the original message cannot be generated from the message digest. Another important property is that two different messages have a very low probability of resulting in the same message digest, a collision. Hash functions can be used to add a message digest to the end of a sent message. The receiver can, if the algorithm for creating the message digest is known, verify the message digest. If the receiver calculates the message digest from the message and gets the same result, the message can be assumed to be free from accidental, but not necessarily of intentional modifications. Example of hash functions includes Message-Digest algorithm 5 (MD5) and Secure Hash Algorithm - Version 1.0 (SHA-1). [12, 18]

Ron Rivest created MD5 in 1991. MD5 works on blocks of 512 bits and produces a 128-bit output. It is designed to be quick on 32-bit machines. It was assumed that an order of \(2^{128} \) operations was required to create any message having a given message digest. In March 2006, Vlastimil Klima [20] used a technique
called tunneling and showed that it is possible to create a MD5 collision within one minute on an Intel Pentium, 1.6 GHz. [20, 27]

SHA-1 was designed by the National Security Agency (NSA) and published in 1995. It is based on the MD5 algorithm using the same block size as input but produces a 160-bit output. There exists other algorithm in the SHA family producing hashes of greater lengths. Even the SHA-1 algorithm has flaws and it is possible to find collisions in $2^{63}$ operations. [18, 24, 39]

To add integrity to the checksum, a shared secret key is used. A checksum created with the use of a secret key is called Message Authentication Code (MAC). This prevents a man in the middle to modify the message without access to the key. A common solution is to use a block cipher to create a MAC. When the MAC is created using cryptographic hash functions such as MD5 and SHA-1 it is called a Hash Message Authentication Code (HMAC). Hash functions where not designed using keys so a special construction using a hash function had to be created. The HMAC algorithm is described in [10]. [9]

### 2.4 Cryptanalysis

Cryptanalysis can be performed by anyone to evaluate the security of a system by trying to break it. Cryptographic algorithms are often subject of cryptanalysis by security experts. A new algorithm has to be examined carefully to evaluate possible vulnerabilities before it can be used. [12]

In the modern field of cryptanalysis, security experts are interested in computer-automated attacks to evaluate different cipher algorithms. A number of optimized algorithms have been introduced to break the modern ciphers. By interfering a connection, five different types of attacks can be used [13, 14]:

1. **Ciphertext only**, Use only the ciphertext and try to reveal the corresponding key or plaintext.

2. **Known plaintext**, Use the ciphertext and the corresponding plaintext to reveal the key.

3. **Chosen plaintext**, Use plaintexts of specific forms and compare it with its corresponding ciphertext to reveal the key.

4. **Chosen ciphertext**, Use a specific ciphertext as input to a receiver and analyze its output.

5. **Adaptively chosen plaintext/ciphertext**, Use information from previously analyzed data to adapt the next attack.
Chapter 3

The TLS Protocol

This chapter describes the Transport Layer Security (TLS) Protocol Version 1.0. Unless otherwise specified, TLS will be referring to TLS version 1.0.

3.1 Introduction

An early security protocol widely used was the Secure Socket Layer (SSL) developed by Netscape Communications in the early 1990’s. SSL version 2 was released to the public in 1994 but had some security weaknesses. For example, SSL version 2 uses a weak algorithm for message digest and does not encrypt all information. It made it possible for attackers to alter messages. Together with other flaws it was considered weak and the protocol was improved and released as version 3.0. A second revision was formed within Internet Engineering Task Force (IETF) to develop an Internet standard. This protocol was called Transport Layer Security (TLS) and is also known as SSL version 3.1. TLS and SSL version 3 are very similar making it easy to support both protocols. [12, 38]

3.1.1 Basics

The primary goal of the TLS protocol is to describe a framework providing privacy and data integrity. The TLS Record Protocol provides connection security that has the following properties [30]:

- **Privacy**, All data that is sent by the application is encrypted using symmetric cryptography. Any key used is unique for each connection and sent using asymmetric cryptography.

- **Reliability**, All encrypted messages include a message digest in the form of a keyed MAC. Plaintext messages does not include any message digest but will be validated before any application data is sent.

- **Identity**, Both parties (peers) can use digitally signed certificates to authenticate the identity of a peer. The authentication is optional but can be required by any of the peers.
3.2 Structure

The TLS protocol is composed of different layers. Located at the lowest level is the TLS Record Protocol. A header including message information, the message data and validation of the data known as footer defines the TLS Record protocol. The TLS Record Protocol should be on top of a transport protocol and the standard [30] specifies four higher-level protocols as shown in Figure 3.1.

If a higher protocol creates a message in plaintext it will be transformed\(^1\) in the TLS Record Layer by the following steps (illustrated in Figure 3.2): [30]

1. The plaintext message containing type, length and data will be used to calculate a MAC. The cipher suite determines the MAC in terms of type and size.

---

\(^1\)This transformation will only occur if the key exchange is complete (i.e. a Change Cipher spec has been sent)
2. If a block cipher is used, padding will be added to the plaintext message to ensure that the total length is an integral multiple of the block cipher’s block size. The length is also appended. Note that the length can be any value between 0 and 255 fulfilling the requirement. This makes it possible to use more than one block for padding. This is not the case in SSL v.3 where the minimum padding that fills the last block has to be used.

3. The plaintext message, MAC and padding data is encrypted and the total size will be updated. The type will remain and is only used when calculating the MAC. Note that the type and length of the message is always sent in plaintext.

[30]

3.2.1 Handshake Protocol

All TLS sessions start with a certain handshake sequence. A server and a client uses the Handshake Protocol to agree on cryptographic algorithm, authenticate each peer and exchange keys using asymmetric algorithms. A typical handshake sequence between a server and client is shown in Figure 3.3.

The first message sent by the client is the Client Hello message. The client shall send this after the underlying connection has been established. It could however be a response to a Hello Request sent by the server if the server initiated the communication. The Client Hello message contains the protocol version and cipher suites the client supports, a session id for session resumption (see section 3.3) and a random value. A cipher suite is a collection of three cryptographic algorithms. One asymmetric used for key exchange, one symmetric used for data transfer and a message digest algorithm. All cipher suites are predefined in [30].

The server responds to the client sending a Server Hello message containing a selected cipher suite and a random value. A Server Certificate always follows this message. The asymmetric algorithm in the cipher suite determines the type of certificate. If the algorithm was RSA, the certificate contains a public key used for encryption by the client. To end the sequence of Server Hello related messages, the server sends a Server Hello Done and waits for the client to respond.

Upon receiving the Server Hello Done, the client uses the agreed asymmetric algorithm and encrypts a message called Pre-Master Secret and sends this to the server in a message called Client Key Exchange. The Pre-Master Secret is used together with the random values to generate the Master Secret. The Master Secret is later used to create keys for encryption and keyed message digests. A message digest of all sent and received messages is calculated and sent to the server as a Finished message. Note that a Change Cipher Spec is sent between these two messages resulting in the Finished message being encrypted (see Section 3.2.2).

When the server receives the Finished message it will verify the message digest and calculate its own Finished message and send it to the client. Note that this will be different because the one that is sent second will include the prior one.

---

2 The TLS Protocol version 1.0 specifies a mandatory cipher suite using DHE for key exchange, DES for encryption and SHA-1 for MAC. In TLS version 1.1, DHE was changed to RSA. [30, 31]
If the client can verify the message digest the connection can be considered safe and the Application Data Protocol can be used.

[30]

### 3.2.2 Change Cipher Spec Protocol

The purpose of the Change Cipher Spec Protocol is to signal that any following messages sent by any other protocol during this session will be protected using the agreed cipher suite. This message has to be sent by both peers since it will only signal that the sender is sending protected messages. This message is only sent during the handshake phase after all keys are determined but before the Finished message containing verifying information. [30]
3.2.3 Application Data Protocol

All data sent by the application uses the Application Data Protocol. The Application Data Protocol can only be used when the handshake-phase is finished. The protocol does not include any extra information and the Record Layer carries the application data. The application data can be fragmented to fit the internal buffers in the TLS implementation and sent as a number of records. [30]

3.2.4 Alert Protocol

The Alert Protocol is used to signal that an error or warning has occurred. This message contains the description of the alert and the alert level. The alert level is divided into warning and fatal and the connection should always be closed if a fatal alert is detected. Descriptions of alerts and appropriate actions to handle alerts are given in the TLS specification. [30]

3.3 Session resumption

During the handshake phase, the server and the client negotiate a 256 bits session identifier. The client can use an old session identifier from an earlier connection or a currently in use session identifier from an existing connection. If no session is available, the client sends an empty session identifier.

If the identifier was non-empty, the server can choose to resume an earlier connection. This requires that keys and other session related information of the resumed session has been saved by both peers. If an old session is resumed, both peers only prepare and send the Finished message. This avoids the expensive asymmetric operations.

In the case of an empty or invalid session identifier, the server can create a new unique identifier for future resumption.

The server always has the option to return an empty session identifier to indicate that session resumption is unavailable. [30]

3.4 HTTPS

The main purpose of TLS is to secure Hypertext Transfer Protocol (HTTP) communication. HTTP Secure (HTTPS) is simply HTTP combined with SSL or TLS. The HTTP server initiates the TLS on port 443 and uses the TLS sockets in the same way as Transmission Control Protocol (TCP) sockets for plaintext communication on port 80\(^3\). [15, 26]

---

\(^3\)Port 80 is the default for regular HTTP communication and port 443 is the default for secure HTTP communication when using a TCP/IP connection. The port numbers can be chosen arbitrarily.
The acting client, normally a web browser, should initiate the connection sending a Client Hello. When the handshake phase is finished all HTTP data should be sent using the Application Data Protocol. The TLS layer should be transparent to the application and the HTTP data sent should not be any different from an unsecured HTTP connection.

The Uniform Resource Identifier (URI) format for HTTPS is different from HTTP and use the https as protocol identifier but use the same syntax as http. [15] An example URI for HTTPS is:

--- Example 3.1: URI for HTTPS

https://webmail.ida.liu.se/
Chapter 4

Export of Cryptography

Every country may have export regulations concerning cryptographic software. This chapter will present the cryptographic regulations in several countries. It will also describe how export controls apply to the TLS protocol.

4.1 International regulations

There are two large international organizations operating with cryptographic regulation issues. These are the Wassenaar Arrangement (WA) and the Organization for Economic Co-operation and Development (OECD). Each country may also have its own laws and regulations concerning export of cryptographic software. [35]

4.1.1 Wassenaar Arrangement

The WA is a global arrangement on export controls for conventional weapons and dual-use goods and technologies. The WA received approval by 33 countries\(^1\) in July 1996 \([6]\).

Cryptography falls in the category dual-use goods. Category 5, Part 2 of the list of dual-use goods specifies that the following functions are subject of export restrictions \([6]\):

- A symmetric algorithm using a key length above 56 bits.
- An asymmetric algorithm based on factorization of integers or logarithms in a multiplicative group of a finite field using a key length above 512 bits (e.g. RSA and DHE).
- An asymmetric algorithm based on other discrete logarithms using a key length above 112 bits (e.g. elliptic curve).

\(^{1}\)Argentina, Australia, Austria, Belgium, Bulgaria, Canada, Czech Republic, Denmark, Finland, France, Germany, Greece, Hungary, Ireland, Italy, Japan, Luxembourg, Netherlands, New Zealand, Norway, Poland, Portugal, Republic of Korea, Romania, Russian Federation, Slovak Republic, Spain, Sweden, Switzerland, Turkey, Ukraine, United Kingdom and the United States.
As of December 2000, mass-market software products containing cryptography is free from any export regulations within the member countries and can use arbitrarily key lengths. [35]

4.1.2 OECD

The OECD group has 30 member countries. The primary goal is to provide recommendations and internationally agreed instruments for governments to succeed in a globalized economy. The OECD provides only non-binding guidelines concerning cryptography. All guidelines are based on the following eight principles [3]:

1. **Trust in cryptographic methods**, A cryptographic method should be reliable and generate trust of use.

2. **Choice of cryptographic methods**, No authority should restrict any cryptographic method if not necessarily and all users should have the option to choose any method to use.

3. **Market driven development of cryptographic methods**, The development of cryptographic methods and standards should be market driven to meet the requirements from changing technologies.

4. **Standards for cryptographic methods**, The development of cryptographic standards should be encouraged at both national and international level.

5. **Protection of privacy and personal data**, Fundamental rights concerning privacy should be respected when using and developing cryptographic methods.

6. **Lawful access**, A national policy can allow lawful access to plaintext material or cryptographic keys.

7. **Liability**, Individuals and entities holding access to cryptographic material should be liable for any misuse.

8. **International co-operation**, Governments should co-operate to support the established policies.

4.1.3 United States

The Export Administration Regulations (EAR) of the Department of Commerce controls the export of cryptography in the US. The export policy is based on three principles: review of encryption products prior to sale, streamlined post export reporting, and license review of certain exports and re-exports of strong encryption to foreign governments. [2]
4.2 Export Control on the TLS Protocol

To export cryptography it shall be approved under an \textit{encryption licensing arrangement}. There are exceptions for certain encryption items. The following items only requires a submitted notification to the Bureau of Industry and Security (BIS) \cite{2}:

- Mass-market encryption software using a key length not above 64 bits.
- All encryption software using a key length not above 56 bits for symmetric algorithms, 512 bits for asymmetric algorithms (112 bits for elliptic curve).

All mass-market encryption software can be exported to non-government end users in any country\footnote{Except Cuba, Iran, Iraq, Libya, North Korea, Sudan and Syria.} under a license exception. This requires a technical review of the software by BIS. This review will add a 30-day waiting period before the encryption item can be exported. There exists a \textit{license free zone}, including all EU countries, to which encryption items can be sent without this 30-day waiting period. \cite{2, 35}

The US also promotes the use of a \textit{key-escrow} system. This system allows all users to use strong encryption if a trusted third party holds enough information to generate keys to read any sent or received messages. A court order can give a government agency the permission to extract such information from the trusted third party. \cite{35}

4.1.4 European Union

Since June 2000 when the EU Council of ministers adopted a new EU Regulation there exists a general export license covering encryption items. This applies to all items (except cryptanalytic items) and is not limited to mass-market items. This license is valid for export to other EU member states and other close trading partners\footnote{Including Australia, Canada, Japan, New Zealand, Norway, Switzerland and the United States.} \cite{1, 33}

Exporting items to other states can be granted by the member state in which the exporter is located. A member state also has the possibility to add further control for public security reasons. \cite{35}

4.2 Export Control on the TLS Protocol

Since TLS is containing cryptographic software it is considered a dual-use item and is therefore subject to export control. Most countries consider using Internet to transfer software between sites located in different countries as export. These transfers must obey the current export regulations. Since cryptography is important to ensure secure communication, many governments have relaxed the policies for especially e-commerce applications. \cite{35}
In both the US and the EU, cryptography that is not user-accessible and used for content decryption in digital right management systems is excluded from control. TLS falls into this category. If the TLS solution can be considered as mass-market cryptography, a country following the Wassenaar Arrangement is also released from control. [35]
Chapter 5

Related Work

After a brief study of related works I found two projects that was frequently referenced. The most well known project is the OpenSSL project. A lesser-known project is called MatrixSSL but was found to be the most related to the subject of this thesis. This chapter aims to briefly describe these projects and to distinguish similarities and differences with the goals of this thesis work.

5.1 OpenSSL

OpenSSL is an open source project providing a full-featured toolkit implementing SSL version 2, SSL version 3 and TLS version 1.0. OpenSSL supports features such as [5]:

- **Server and client side**, Both sides are supported in the library. OpenSSL also includes a server and a client program to use in test purposes.

- **Key generation**, Private and public keys for RSA and other asymmetric algorithms can be generated.

- **Certificate generation**, A public key can be turned into a certificate when combined with information about the certificate owner. Certificate requests can be created and these can either be sent to a CA or self-signed using OpenSSL.

- **Many ciphers and modes of operation**, Basically all ciphers and the different modes of operations are supported. All ciphers can be used with the OpenSSL command tool. This can be used to verify other implementations.

While providing an extensive support for SSL and TLS it is not practical to use in an embedded system. It is a large library built for server level platforms like the Apache Web server. OpenSSL does however provide many features to ease the testing of a smaller library.
5.2 MatrixSSL

Developed by PeerSec Networks, MatrixSSL is partially distributed under the GPL license. MatrixSSL is an embedded solution; it aims for a small code footprint and low resource utilization. It supports both the client and the server side in both SSL and TLS. TLS and the cipher AES is however only included with a commercial license. The free license supports, in addition to a SSL server and client side, the ciphers RSA, 3DES and ARC4. It also includes support for different certificate formats and certificate chain authentication. [4]

While MatrixSSL has the goal of being a general SSL solution for embedded devices the development part of this thesis aims to describe a minimal TLS solution aiming to only include what is necessary. Since the TLS and AES support is limited to the commercial license there will be no real comparison with the results of this thesis work. The specification promise a library size smaller than 50 kB but no performance figures related to cryptographic operations could be found.
Chapter 6

Development Environment

During the development part in this thesis work a collection of different tools were used. The actual development can be divided into two phases.

The first phase used the OSE Reference System to create a soft OSE kernel. This made it possible to develop, compile and run code on the reference system. The application was built as a load module that could be loaded at run time and is free to use platform services such as file systems and networks. The second phase was the evaluation on the target device.

This chapter aims to briefly describe what tools that were used and to suggest some alternatives.

6.1 Software tools

In the first phase the main tool for almost all software development was Emacs. Another tool suitable for development for the reference system is Eclipse that also integrates run mode debugging. The code was compiled for the reference system using gcc. OSE Illuminator was used for system-level debugging. Illuminator also provides memory usage surveillance to detect memory usage peaks and possible memory leaks. To analyze and confirm the network streams the tool Ethereal was used. Ethereal has support for TLS packages by default and can also detect corrupted TLS records.

The only tool used in the second phase was the IAR Embedded Workbench. It supports code editing, compiling and target debugging. An alternative solution, using only free GNU tools, is to compile with arm-gcc\(^1\) and use gdb for target debugging.

\(^1\)In this case suitable since the target device uses an ARM7 core.
6.2 Hardware Evaluation Kit

The target board was a part of an evaluation kit\(^2\) from Atmel. The device was the AT91SAM7XC256 from the SAM7XC series. The device was programmed and debugged using the JTAG-interface. One serial port was used for the user interface. The cryptographic features on this particular device are described in Section 8.3 on page 34.

\(^2\)Product Code: AT91SAM7XC-EK.
Chapter 7

Implementation Considerations

When implementing the TLS protocol, there are many design decisions that can save memory or improve performance. In this chapter some of the most important parts of TLS will be identified and discussed in order to achieve good utilization of the available resources. The chapter is divided into memory and performance considerations. Since these are seldom independent the chapter ends with a brief discussion concerning the balance of memory usage and performance.

7.1 Memory Utilization

This section mainly focuses on the memory management issues and optional feature included in the TLS implementation. Mandatory features with good potential for optimization are also considered.

7.1.1 Code

The code size of a TLS implementation can be reduced significantly by excluding features in the specification that are optional according to the specification. Some features will however increase the average performance and should only be excluded if necessary.

If the target of the TLS implementation will only be accessed with a web browser or another client-side software there is no need to include any client-side code. By only supporting the server-side the following features can be excluded:

- **The client hello message**, This message is very similar to the already included server hello and is not a significant saving.

- **The client key exchange message**, This is the answer to a server certificate and includes certificate authentication and asymmetric key encryption. By excluding the certificate authentication, there is no need to store any root certificates signed by a CA.
• **Optional handshake messages**, It is the server that controls what messages to be sent during the handshake phase. A server can request a client certificate and anonymous negotiation. By only supporting the server-side, the flow can be controlled and therefore minimized.

An important decision is the selected cipher suites to support. A cipher suite contains an asymmetric and symmetric encryption method and a MAC function.

The choice of public key method is usually RSA or authenticated DHE using Digital Signature Standard (DSS) for signing. Both RSA and DHE require a library with support for large integer calculations. This library can be optimized for space by only supporting the operations used by the application and by implementing seldom used function in a very straightforward way. There are many alternative solutions for many algorithms that require more code space but yield a performance improvement. Some different techniques are discussed in Section 7.2.1 on page 29. Only the RSA support was implemented in this thesis. The main reasons for this decision were that DHE with DSS requires one more message, called *server key exchange* to be sent in the handshake phase and the fact that DSS is currently limited to only provide 1024-bits security [28].

There are usually three different methods for symmetric encryption, RC4, 3DES and AES. RC4 is very small algorithm compared to both 3DES and AES. The AES algorithm is different depending on key size and can be made smaller by selecting only to support one key size. Both 3DES and AES can be made smaller by replacing pre-calculated tables with code for run time calculations.

The only MAC algorithms included in the TLS protocol are MD5 and SHA-1. Since both are used to generate keys, both have to be implemented even if never used in any cipher suites. The choice of MAC should be based on performance tests and the current security level.\(^1\)

TLS and SSL version 3 (SSLv3) are very alike and use the same format in the client hello and server hello messages. This means that without any additional code it is possible to detect a SSLv3 client hello. To fully support SSLv3 the implementation has to be extended where the difference occurs. The following changes were made from SSLv3 to TLS (considering only the server side implementation):

• **Keyed MAC**, TLS uses a keyed-hash MAC, called HMAC, when generating message digests, calculating the master secret and generating keys. SSLv3 use regular hash functions. However, since both methods use the MD5 algorithm and the SHA-1 algorithm, no extra functions have to be included.

• **Padding**, As mentioned in section 3.2, the padding length used in TLS can be of any length resulting in a message length that is an integral multiple of the block cipher's block size. Every padding byte must be filled with the value equal to the padding length. The smallest padding possible shall be used in SSLv3 and the content should be random data. One

\(^1\)Due to lacking security offered by MD5 and SHA-1, the Internet Draft of TLS version 1.2 suggests to remove MD5 and SHA-1 as the only used MAC algorithms and allow extended algorithms providing higher security. [32]
solution covering both protocols is to always use the minimum length and the padding length as content.

- **Structural**, Minor changes, easily covered by if-blocks, include what messages to be included in the finished hash, allowed cipher suites and additional alarm codes.

SSL version 2 (SSLv2) does not use the same structure for message records and do not provide the same security as SSLv3 and TLS. There is no need to support the SSLv2 protocol unless the other peer has no other option than to use SSLv2. However, some web browsers, such as Mozilla Firefox v1.5, use the SSLv2 client hello when initiating a connection even if TLS is used. This is done by sending the TLS version in a SSLv2 client hello. This message uses another header length and a different structure of the message content. It is an easy task to detect this message but to support it requires more code space.

### 7.1.2 Buffers

Buffers are necessary in the TLS layer for intermediate storing of packages between the application and the underlying communication layer. The record size in the TLS protocol cannot be negotiated and can be of any size up to 16 kB. [30]

Memory can either be allocated statically at compile time or dynamically at run time. The advantage of using dynamic memory allocation is that it is possible to only use the space needed for buffer instead of having to use the maximum size. Dynamic allocation can however lead to fragmentation making the program to fail when the allocation does not succeed. One solution to fragmentation is to setup different heaps with pre-allocated fixed sized buffers. A static memory allocation scheme is safer and often recommended on embedded systems.

By using a static memory allocation scheme there is a need to create an incoming buffer of 16 kB if the TLS standard shall be fulfilled. The buffer has to store all incoming data to calculate and verify the message digest before passing it to the application. The static size can be decreased if it is unlikely that any large records will be received. If the target application is a static page web browser only supporting simple GET commands there is no need for a large incoming buffer. A 2 kB buffer may be enough to store any incoming message but if any larger message arrives a dynamic memory solution has to be used or the connection has to be closed. During the handshake phase there is a need to save a lot of temporary information. This information includes client and server random values, all sent and received handshake messages and the master secret. When the handshake is complete there is no need to maintain those in memory. If a large incoming buffer is used, a part of this buffer can be dedicated as a temporary buffer during the handshake phase. This is illustrated in Figure 7.1.

By storing all intermediate values at the end of the incoming buffer the memory overhead in the handshake phase is eliminated. This may result in invalid data if the incoming buffer is to small and no alternative solution exists when the overflow is detected. The size of the temporary data should be estimated in advance to ensure proper function. Another way of saving memory during the
handshake phase is to create a memory pool shared by all connections. If a new
connection is established it will take control of the memory pool and release it
when the handshake phase is complete. Other connections may have to wait until
the memory pool is released. This can be accomplished by protecting the memory
pool with a mutex\(^2\) preventing any sharing violations.

The outgoing buffer, or write buffer, does not have to be as large as 16 kB. Since
the TLS layer controls all sent data a much smaller package size can be used. All
application data packages can be divided and sent as a number of records. The
outgoing buffer has to be large enough to send the largest handshake message.
This is usually the server certificate with an approximate size of 1-2 kB. Reducing
the write buffer size may have impact on the performance by lower throughput
due to larger overhead\(^3\).

One solution to eliminate the whole write buffer is to use only one buffer for
both incoming and outgoing data. If the application can work properly without
simultaneous read and write operations the TLS layer can work with only one
buffer.

If a dynamic memory allocation scheme is used all buffers can be initiated to
a small size and reallocated when needed. There are different strategies when
reallocating a buffer. A common solution is to reallocate the buffer to the exact
size of the current data. If the data is likely to increase a better strategy might be
is increase the buffer with a factor of the current size. Since every new run time
allocation can fail it is important to check that the buffer is valid before use.

When an application data record is sent or received it has to be transformed
between plaintext and ciphertext using a symmetric key algorithm. All symmetric
key algorithms in the TLS specification work either bytewise or on larger blocks
of data. By only storing intermediate values during cryptographic calculations it
is possible to use the same buffer area for input as for output.

7.1.3 Keys and Secrets

Keys for encryption, decryption and message digest operations together with the
secret initiation vector have to be stored within the connection context since they
are unique for each connection. These values will often be used and should be
stored in RAM to increase security and minimize access time. The size of the
keys and other secret values are dependent on the selected cipher suite. A selected
cryptographic algorithm needs space to maintain the internal state. The block
ciphers 3DES and AES need storage of keys and an initiation vector. The key

\( ^2 \)A mutex is an object that allow different threads to shared the same resources. Using a
mutex will ensure that the resource is not accessed simultaneously.

\( ^3 \)The smaller the data size, the larger the header and footer overhead.
size is fixed to 168 bits for 3DES but can vary for AES\textsuperscript{4} and the initiation vector is of the same size as the block size. RC4 has a fixed sized internal state. The key-scheduling algorithm for RC4 initiates an array of 256 bytes and two 8-bit index pointers. The internal state of RC4 is almost twice as large as for 3DES or AES.

7.1.4 Certificates

The server sends a digital certificate to the client for identification. This certificate contains the public key used in the asymmetric key exchange. A certificate is usually 1-2 kB in size and the TLS specification supports the use of multiple certificates. Only one certificate is necessary and since it will not be changed during run time it can be stored in constant data space, flash or ROM if available.

Certificates are stored in a description language called Abstract Syntax Notation One (ASN.1)\textsuperscript{5}. Certificates can be created by the tool package in OpenSSL. Care should be taken when creating a certificate so it will not grow unnecessary large. ASN.1 supports much certificate information and extensions to be added into a certificate. By using a configuration when creating the certificate, removing all extensions and minimizing the information, a 1024-bits RSA certificate can be decreased to a size of approximately 400 bytes.

Another saving of the server only approach is that the ASN.1 parsing code can be eliminated. The certificate can be sent in raw data and does not require any preparations on the server side.

If a created certificate is signed by a CA is will usually expire in a few years. If a certificate is to be used in an embedded system where updating the certificate is impossible or unpractical it is possible to create a self signed certificate and significantly increase the lifetime of the certificate.

7.2 Performance Consideration

TLS requires many cryptographic operations that limit the data throughput. This section describes the possible bottlenecks and how the performance can be improved with different techniques.

7.2.1 Cryptography

Asymmetric Ciphers

Asymmetric algorithms are very expensive operations. RSA decryption requires many operations to reveal the plaintext. The following operation is evaluated:

\[ c^d \mod n \] (7.1)

\[ \text{[12]} \] The large amount of operations is a consequence of using large numbers for \( c \), \( d \) and \( n \). The size of these numbers is in the range of 512 to 4096 bits.

\textsuperscript{4}AES currently supports a key size of 128, 192 and 256 bits.

\textsuperscript{5}Standard from 1984 by the International Telecommunication Union.
The most common operation in RSA decryption is modular multiplication. A straightforward solution, referred to as the classical algorithm, is to perform the multiplication followed by a division to calculate the remainder. There are other methods providing a more efficient implementation of modular multiplication. A method called Montgomery reduction can be used to efficiently perform modular exponentiation. Since many reductions are performed with the same modulus in RSA, Barrett reduction can be used to perform reduction without any division. By precomputing a value, right-shifts can be used instead. [21]

An important performance factor is the key length used in RSA. The greater the key length the more computer operations are required for every multi-byte operation. A larger key length also requires more memory for key storage and during calculations to store intermediate results. RSA using 512 bits has been broken and also RSA using 640 bits which is 101 times harder to solve, was broken by 80 2.2GHz Opteron processors in less than half a year. Using 1024-bits, it will take 7 million times longer to break it than when using only 512 bits. RSA Laboratories suggests that 1024 bits are safe until 2020 if using the same methods for breaking RSA. [29]

One way to speed up RSA decryption without decreasing the number of bits in the key length is to use the Chinese Remainder Theorem (CRT). Remembering section 2.1.2, the modulus \( n \) was a product of the large primes \( p \) and \( q \). Using the CRT the problem of computing

\[
c^d \pmod{n}
\]  

(7.2)

can be done by calculating

\[
M_p = c_{p}^{d_{p}} \pmod{p},
M_q = c_{q}^{d_{q}} \pmod{q}
\]  

(7.3)

where

\[
c_{p} = c \pmod{p}, d_{p} = d \pmod{p - 1},
\]

\[
c_{q} = c \pmod{q}, d_{q} = d \pmod{q - 1}.
\]  

(7.4)

and finally computing:

\[
[M_p(q^{-1} \pmod{p})q + M_q(p^{-1} \pmod{q})p] \pmod{n}.
\]  

(7.5)

This requires that the values \( d_p \) and \( d_q \) are calculated in advance and stored in memory. The ideal case gives a speedup by four times. [40]

**Symmetric Ciphers**

Symmetric algorithms are much more frequently used in a session since all application data is encrypted with symmetric algorithms. There are a number of different algorithms with different properties that can be used.

RC4 is not only very small in code size; it also has a speed advantage over 3DES and AES. RC4 is a stream cipher and is basically only a few additions, one
7.3 Balance Performance and Memory

swap and one exclusive or operation for every byte of data. This can be written in assembly in an effort to reduce the number of computer instructions. AES was created to be more secure and is also faster than 3DES.

There are techniques to optimize the AES algorithm on 32-bits platforms. An efficient implementation is described in [11] where the data block is transposed to suit the software representation. Figure 7.2 shows how the original data block is transposed. The most expensive step in every AES round, the MixColumns, can now operate on rows instead of columns. And the gain in speed is a result of the computer architecture where each row can be stored as a 32-bit word. [11]

![Figure 7.2. Transposing the data block in AES yields improvements on 32-bits platforms.](image)

Hash Functions

A message digest in the form of a HMAC is added to all application data. The agreed HMAC method is used as frequently as the symmetric algorithms for encryption and decryption. A HMAC using MD5 as hash function is very similar to using SHA-1 and the same functions combined with parameterizations can be used. MD5 and SHA-1 are designed to be quick and should not create a bottleneck.

Clocked Data Processing

The processing time of a data buffer by encryption or decryption and HMAC calculation could be critical. The processing time could be high enough to trigger the watchdog or delay a critical process if the system does not use preemptive scheduling. One solution is to process the buffer in many smaller steps. By scheduling a periodical timer interrupt, the data processing can be clocked and in every clock a pre determined amount of data blocks is processed. This requires more data in the TLS session context to extend the current state.

7.3 Balance Performance and Memory

Storing pre-calculated values and replacing functions with look-up tables could make many algorithms faster. There are also features that could be included to increase the performance.
Session resumption described in Section 3.3 can be used when the same peers want to communicate frequently. If session resumption is used there are some design decisions to make, affecting memory usage and security. For each session stored, space for the session id and master secret is required. To be able to decide what session to override if all memory allocated for session resumption is used, a timestamp could be added. The timestamp will also grant the opportunity to delete old sessions preventing a master secret to be used for a longer time than considered secure. [30] recommends that an upper limit of 24 hours being used. It also points out that stored sessions should not be kept in stable storage.

Both AES and 3DES can be either smaller in code space or faster in terms of higher throughput by adjusting the number and size of pre-calculated tables. If memory is available, they can be entirely table driven. If the tables can be stored in ROM the code size will be much smaller but the ROM access time might decrease the throughput.
Chapter 8

Hardware Support

One way to speed up encryption and decryption without sacrificing security (i.e. decreasing key length or changing to a less secure algorithm) is to utilize special hardware support. There are different types of hardware support that might be used.

There are dedicated hardware devices available supporting different algorithms. These devices often include tamper-resistant memories for secure storage of keys and other secrets. Many such devices are designed for general-purpose computers and are referred to as a Hardware Security Module (HSM). [7] This also removes the static code size for the operations but requires some extra code for the external communication.

Some CPU cores include cryptographic operations. One example of this is the SAM7XC series from Atmel. A selected device from this series will be described in this chapter and later evaluated in the following chapters. Using this type of device, the code size is reduced and only small configurations are necessary.

Another alternative is to add special CPU instructions to the instruction set. One example is to add a multiply-accumulate instruction that combines many steps in an ordinary multiple-precision multiplication [16]. The result of adding new instructions will still keep the processor busy during these critical operations. The code size might get reduced but not to the extent as an external or internal hardware solution.

8.1 Cryptographic Module Validation Program

The Cryptographic Module Validation Program (CMVP) was established by NIST and allows all vendors to validate their cryptographic modules to the standard FIPS 140-2. The standard defines four levels of the modules security. A higher level requires tamper-resistance and robustness against physical and environmental attacks. Approved security functions applicable to a TLS solution includes AES, DES, 3DES, SHA-1 and RSA. NIST provides a list of all validated modules and the assigned level of security. [25]
8.2 Trusted Platform Module

Trusted Platform Module (TPM) is a specification describing a microcontroller supporting various cryptographic features. It is also the name of any implementation of this specification. The specification was created and is maintained by the Trusted Computing Group (TCG). The TCG is a non-profit organization formed to develop open standards for hardware security technologies. TCG provides design principles and the TPM specification. The following features are required by the current specification [37]:

- **Secure storage**, Security-related information such as keys, passwords and digital certificates can be stored. Access to this storage can be denied if the boot sequence is unexpected.

- **Key generation**, Asymmetric key pairs for RSA can be generated. A random number generator is available to generate other keys and secrets.

- **Cryptographic algorithms**, The TPM must support RSA with key sizes of 512, 768, 1024 and 2048 bits. Other key sizes may be supported. The only mandatory symmetric algorithm is the one-time-pad using XOR. AES may be supported as an alternative algorithm.

- **Checksum generation**, HMAC generation according to [10] using SHA-1 is mandatory. SHA-1 can also be used separately for message digest calculations.

The TPM is not limited to any platform but is today typically used in a PC. The functionality in TPMs may differ as they are produced by different manufacturers. All that is marked with MUST in the specification is an absolute requirement and is supported in every TPM. [37]

8.3 Evaluation Kit Device

The device introduced in section 6.2 on page 24 is not a TPM but does include a few cryptographic features. This device supports both the AES and the 3DES standards. All keys and the initiation vector are stored in write only registers. There are three different ways to encrypt or decrypt a message [8]:

1. **Manual mode**, The data block, keys and the initiation vector is written to certain registers and the data will be processed when a start bit is set manually.

2. **Auto mode**, Requires the same preparations as in manual mode but the processing will start automatically after all registers has been set.

3. **PDC mode**, The Peripheral Data Controller (PDC) can be used to process an entire buffer without interfering with the software. Transmit and receive buffer pointers and the buffer size is instead written to registers. The receive buffer pointer can be equal to the transmit buffer pointer.
The completion of the encryption or decryption can either be polled by reading a status register or configured to trigger an internal interrupt.

This device supports all five modes of operations\(^1\) in AES but is limited to the key length of 128 bits\(^2\). The 3DES supports four modes of operation\(^3\) and can also be run in single DES mode.

---

\(^1\)ECB, CBC, OFB, CFB and CTR.
\(^2\)The C512 device also supports 192 bits and 256 bits.
\(^3\)ECB, CBC, OFB and CFB.
Chapter 9

Results and Analysis

In this chapter the performed measurements will be presented. The results will be analyzed and compared when applicable. The chapter begins with the measurement scope and methods used. The next section contains the size of the TLS library and includes an estimation of the total memory required during open TLS connections. The last section presents the execution times of the implemented cryptographic algorithms. The cryptographic hardware support will be compared with its software counterpart.

9.1 Measurement Scope

The measurements cover both memory usage of the complete application as well as the different parts of the implementation, and the performance of certain algorithms. The complete webserver application was only tested on the reference system and the response times could not be measured. Since the response times are dependent on the performance of the symmetric, asymmetric and HMAC algorithms the times can be estimated by measuring these separately on the target.

9.2 Measurement Methods

The application and the TLS library were compiled with gcc with the second level of optimization enabled (the -O2 flag). To measure the total amount of space used by each part of the application the Unix command size was used on all object files. This command lists the size of instructions (text), data structures (data) and uninitialized user data (bss). Only one copy of each part will be used so the static memory usage is the sum of these three components. When calculating the total static memory usage, the size of additional data such as certificates and keys will be added together with the size of the object files. The dynamic memory usage during one open connection will be estimated by calculating the size of the current server and socket contexts.
The execution time on the target was calculated by counting the number of clock cycles every operation consumes. An internal counter was configured to count every 128th clock cycle. The internal clock was configured to 48 MHz. To handle overflow of the 16-bit counter an internal interrupt was used to store the result as 32 bits. The time for encryption and decryption was measured separately for all symmetric algorithms. A buffer of 1 kB of random data was used as input to all symmetric algorithms and HMAC algorithms. Client key exchange data received from web browsers during the test phase of the reference system was used as input when testing the RSA decryption at the target. The output data was validated by comparing it with the output given by OpenSSL using the same input.

9.3 Memory Analysis

The results from the memory measurements is presented in Table 9.1. A small webserver using the TLS library uses approximately 27 kB of memory. To work properly it also need space for a certificate, RSA keys, buffers for incoming and outgoing data and encryption keys. Assume that a connection is established using RSA-1024 for key exchange, AES-128 for data encryption and SHA-1 for HMAC generation. The estimated memory usage is listed in Table 9.2 on page 39. The first part of the table is the server context and is independent of the number of connections. The second part contains one open connection. This will increase by the factor of the number of connections.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA decryption</td>
<td>7444</td>
<td>129</td>
<td>0</td>
<td>7573</td>
</tr>
<tr>
<td>TLS protocol</td>
<td>5372</td>
<td>0</td>
<td>0</td>
<td>5372</td>
</tr>
<tr>
<td>3DES</td>
<td>3938</td>
<td>0</td>
<td>0</td>
<td>3938</td>
</tr>
<tr>
<td>AES</td>
<td>2551</td>
<td>0</td>
<td>0</td>
<td>2551</td>
</tr>
<tr>
<td>RC4</td>
<td>213</td>
<td>0</td>
<td>0</td>
<td>213</td>
</tr>
<tr>
<td>PRF and HMAC</td>
<td>1063</td>
<td>1</td>
<td>0</td>
<td>1064</td>
</tr>
<tr>
<td>MD5</td>
<td>2120</td>
<td>64</td>
<td>0</td>
<td>2184</td>
</tr>
<tr>
<td>SHA-1</td>
<td>1217</td>
<td>0</td>
<td>0</td>
<td>1217</td>
</tr>
<tr>
<td>Total library size</td>
<td>23908</td>
<td>194</td>
<td>0</td>
<td>24102</td>
</tr>
<tr>
<td>Webserver(^1)</td>
<td>2185</td>
<td>12</td>
<td>340</td>
<td>2537</td>
</tr>
</tbody>
</table>

The code size combined with the memory usage for one open connection including the server context is approximately 47 kB in size. This includes the three

\(^1\)The webserver was based on nw\(\text{eb}\) by Nigel Griffiths. It was modified to use \(\text{select()}\) instead of forking child processes. The HTML header was extended to include package length and keep-alive parameters.
symmetric ciphers RC4, 3DES and AES and should be able to accept most incoming connection. If session resumption with the storage of five connections is added, an additional 0.5 kB of space is required. Symmetric ciphers and features such as session resumption were made optional and could be selected on build time.

This library does not fully support SSLv3 and SSLv2. It does however support the use of either the record type of SSLv3 or SSLv2 when receiving the Client Hello message. In this case it was considered necessary to be compatible with a wider range of web browsers. All other incoming records must follow the TLS standard.

The error handling is not fully implemented. It can receive any record using the Alert protocol. The action taken upon receiving an alert is currently to always close the connection. The TLS implementation does not send any alerts. All detected errors are treated as critical by closing the connection without any notice to the other peer. According to the specification it is required to always send a close notification before closing the connection.

An attempt to minimize the total memory usage could lead to the exclusion of the ciphers AES and 3DES and limit the RSA keys to 512 bits. The decreased key length of RSA will also affect the certificate size. This can save another 6 - 7 kB of space. The savings is however at the expense of security and compatibility. Supporting only one cipher, and especially not the mandatory 3DES cipher, could make connections fail at the negotiating phase due to there being no matching cipher suite. The 512-bit RSA keys do not generate a significant memory saving but have other positive effects as will be discussed later in this chapter.

Allocating 16 kB for the incoming buffer is necessary to be able to support all incoming records. This buffer size should be decreased to the greatest extent

\footnote{A 1024-bit certificate can be as small as 400 bytes but will quickly grow when adding more information.}

\footnote{The out buffer size is not definite. It could be adapted to better suite the sent data.}

\footnote{Includes connection state, data pointers and status flags.}
possible since it will grow by the number of connected clients. During all tests the specified size of 16 kB was used for compatibility reasons. The outgoing buffer was adapted to the web server application often sending small packages and allocated to 2 kB. Sharing the same buffer for incoming and outgoing records was never evaluated in practice.

It is important to remember that the memory usage will grow when only the web server and the TLS library is compiled. Some standard library functions are used and those are not included in these estimations. This will however not add any overhead if this is a part of another application that is likely to already include the most commonly used functions of the standard library. The actual web material such as HTML pages and possible images could be stored in flash or in constant data space if available.

9.4 Performance Analysis

The performance measurements are divided into three categories. The encryption and decryption times for symmetric algorithms are listed in Table 9.3. The RSA decryption time for different key sizes is presented in Table 9.4 on page 41. Finally, the HMAC generation times are shown in Table 9.5 on page 41.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Software [ms]</th>
<th>Hardware [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES encryption</td>
<td>20.73</td>
<td>0.23</td>
</tr>
<tr>
<td>3DES decryption</td>
<td>20.88</td>
<td>0.23</td>
</tr>
<tr>
<td>AES encryption</td>
<td>17.36</td>
<td>0.15</td>
</tr>
<tr>
<td>AES decryption</td>
<td>27.48</td>
<td>0.15</td>
</tr>
<tr>
<td>ARC4 encryption</td>
<td>1.45</td>
<td>-</td>
</tr>
<tr>
<td>ARC4 decryption</td>
<td>1.45</td>
<td>-</td>
</tr>
</tbody>
</table>

The advantage of using a hardware solution is clearly shown in Table 9.3. The hardware 3DES encryption is more than 90 times faster than its software counterpart. The gain is even higher for AES were the encryption is more than 155 times faster and the decryption is more than 183 times faster. Not only do the hardware support grant a significant speed advantage, it also removes most of the code required to perform the operations.

Note that the software version of 3DES is performing better than AES for decryption in this test. AES is expected to be faster and one explanation is that the current 3DES code is about 1.5 kB larger (see Table 9.1 on page 38) and this is the result of using more lookup-tables. If the same amount of memory was used for AES, improvements in execution speed are expected. The speed advantage of ARC4 is clearly shown when compared to the software versions of the block ciphers. ARC4 is however almost 10 times slower than the fastest hardware version.
All software solutions are standard solutions with few optimizations. ARC4 was based on [19] and modified by removing all modular operations since the key size was fixed to 128 bits. 3DES was derived from the DES algorithm in [28]. AES was basically coded from the description in [23]. The transposed block structure presented in [11] was used but never compared to the default block structure.

Table 9.4. Decryption speed of RSA.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Decryption [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA - 512</td>
<td>1.32</td>
</tr>
<tr>
<td>RSA - 1024</td>
<td>8.67</td>
</tr>
</tbody>
</table>

The RSA decryption performance as presented in Table 9.4 shows that there is a big difference when using different key lengths. Note that the time in his table use the unit seconds. Using 1024 bits would delay the first connection to a server by almost 9 seconds. Such long delay could not only be a problem for the client user but also to the server being occupied for that long to serve one incoming connection. Using only 512 bits the time is significantly decreased and could be considered acceptable for most users.

This shows the importance of adding the support for session resumption. Session resumption can be configured to save all incoming connection if there are only few different client users.

An important factor is the library used for multi-precision integer operations. This implementation used the free library iMath developed by Michael J. Fromberger. Using the GMP\(^5\) instead is expected to improve the decryption times.

Table 9.5. HMAC generation times.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Digest time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>1.00</td>
</tr>
<tr>
<td>SHA-1</td>
<td>3.75</td>
</tr>
</tbody>
</table>

The execution times from HMAC generation according to [10] using the cryptographic hash functions MD5 and SHA-1 are shown in Table 9.5. SHA-1 is much slower but once again this is likely the result of the implementation being smaller. SHA-1 was believed to be slightly slower since it uses a key size of 20 bits where MD5 only use 16 bits.

To estimate the additional delay by using TLS the sum of all time consumed by cryptographic operations can be calculated. The handshake phase on the server

---

\(^5\)GNU Multiple Precision Arithmetic Library, distributed under the GNU LGPL.
side is delayed by the RSA decryption and encryption and decryption of the \textit{finished} message. The \textit{finished} message is fixed to 12 bytes and encryption and HMAC calculations will be almost negligible compared to the RSA decryption. A hardware solution for RSA should decrease the extra delay and also remove the large library for multi-precision integer operations.
Chapter 10

Conclusions and Further Studies

This chapter contains the final conclusions of the implementation. This chapter aims to decide if the resource utilization is good enough for embedded system and what level of security it provides. A short discussion about possible additional work will end this chapter.

10.1 Resource Utilization

The memory usage is possible to control within certain boundaries. Ciphers and some feature such as session resumption and optional handshake messages can be left out to save static code space. There is also a possibility to save constant data on external memory if available.

The poor performance is the largest problem encountered during this thesis work. Even if only one algorithm is performing notably bad it will affect the overall performance significantly. The RSA decryption did not complete within a reasonable time using 1024 bits. Using only 512 bits gave a big improvement but was still rather slow. It is important to find a suitable solution for the asymmetric operations. The general multi-precision integer library that was used in this thesis did not perform well enough. Another important design decision is the buffer management. A lot of memory can be saved if the in and out buffers can be decreased by using a non-conservative method like reallocation or sharing buffers.

TLS itself was found to be applicable to embedded systems. No user interactions are required and by using self-signed certificates, there is no need for any maintenance updates requiring hardware access.

It depends on the underlying system if the selected solution is suitable or not. Is there enough memory available to install and use it and is the overhead generated from cryptographic operation acceptable? If the system itself is not feasible for TLS a hardware extension is perhaps possible. It seems from a quick investigation that hardware modules for encryption are a fast growing market. The evaluated
device performed remarkably well and not only was the performance increased, the code space for the operation was reduced to a few loading of internal registers.

10.2 Security

The implementation supports TLS version 1.0\textsuperscript{1} and offers the security features found in the specification. There are constantly new proposed attacks against different algorithms and also against the TLS protocol. The TLS specification does contain pointers to appropriate actions when a suspected attack is detected but is not complete.

Algorithms such as RC4 and AES do not contain any special constraint according to the TLS specification. The design of the algorithms is a balance not only of memory usage and performance but also of security. A highly optimized algorithm may be vulnerable to sophisticated timing attacks if different inputs result in different computation times.

The required security is highly dependent of the application and the target environment. Is it possible or probable that an attack will occur and by whom? An organized attack by a distributed system requires much higher security to withstand than a single eave-dropper trying to read the current data flow.

10.3 Further Studies

The most important thing to further investigate is the performance of asymmetric algorithms. How can RSA decryption in software be improved without sacrificing a large amount of memory? This thesis did contain some theoretical discussion of possible improvements for large integer operation but never evaluated them. Are there any small, inexpensive RSA hardware modules that easily can be integrated to do the decryption? It would be interesting to compare the hardware solutions with a highly optimized software version.

This thesis mainly focuses on TLS version 1.0. Recently, version 1.1 was released and there exists a draft of version 1.2. TLS version 1.1 did not contain any major changes from version 1.0 but version 1.2 introduces new algorithms for HMAC generation and other feature for higher security. How much overhead is created supporting many versions and is TLS developing into a standard that is infeasible to incorporate into smaller systems? Can the standard be more adapted to embedded system? TLS could be extended with optional features such as buffer size negotiation and the ability to let the client do the RSA decryption.

\textsuperscript{1}The mandatory cipher suite using DHE is not supported.
Bibliography


Appendix A

Dictionary

An alphabetized list of explained abbreviations used in this thesis.

**3DES** Triple Data Encryption Standard. Block cipher using the DES algorithm three times.

**AES** Advanced Encryption Standard. Block cipher designed for high security and efficiency to replace 3DES.

**ASN.1** Abstract Syntax Notation One. Standard notation describing data structures.


**ARM** Advanced RISC Machine. RISC processor used in many embedded systems.

**BIS** Bureau of Industry and Security. U.S. agency dealing with security and high technology.

**CA** Certificate Authority. Organization providing signing of digital certificates.

**CBC** Cipher Block Chaining. Block cipher mode. Each block is XOR’ed with the last encrypted block before encrypted. The first block is XOR’ed with the IV.

**CFB** Cipher Feedback. Block cipher mode. The IV is encrypted and XOR’ed with the first block to output one block. The output block is used as feedback to the next encryption acting as the new IV.

**CMVP** Cryptographic Module Validation Program.

**CRT** Chinese Remainder Theorem. Mathematical theorem that can be used to simplify RSA decryption.

**CTR** Counter. Block cipher mode. Transforms a block cipher to a stream cipher.

**DES** Data Encryption Standard. Frequently used block cipher.


EAR  Export Administration Regulations. U.S. export regulations covering commercial items.

ECB  Electronic Codebook. Block cipher mode. All blocks are encrypted without use of any IV or feedbacks.


HTTP  Hypertext Transfer Protocol. Protocol frequently used over the Internet.

HTTPS  HTTP Secure. HTTP using a secure connection.

HSM  Hardware Security Module. External hardware providing security features.

IETF  Internet Engineering Task Force. Organization that develops and provides Internet standards.

IP  Internet Protocol. Low level network protocol with global addressing.

IV  Initiation Vector. A block of data required by some modes in block ciphers.

MAC  Message Authentication Code. A message digest calculated from a message in combination with a secret key.

MD5  Message-Digest algorithm 5. Frequently used message digest algorithm designed by Ronald Rivest.

NIST  National Institute of Standards and Technology. U.S. agency working with different standards.


OFB  Output Feedback. Block cipher mode. Same as CFB but use the encrypted output as next input. This makes encryption and decryption using the same algorithm.

OECD  Organization for Economic Co-operation and Development. International organization promoting free market trading.

OSE  Operating System Embedded. Real-time operating system developed by ENEA.

PDC  Peripheral Data Controller. Data controller transferring data on-chip without processor interference.
**RC4**  Rivest Cipher 4. Frequently used stream cipher.

**RISC**  Reduced Instruction Set Computer. CPU design method providing few but fast computer instructions.

**RFC**  Request for Comments. Documentation of Internet technologies.

**RSA**  Rivest, Shamir and Adleman. Algorithm for public key encryption.

**SHA-1**  Secure Hash Algorithm - Version 1.0. Frequently used message digest algorithm designed by NSA.

**SSL**  Secure Socket Layer. Cryptographic protocol for secure network communications.

**SSLv2**  SSL version 2. The second SSL version.

**SSLv3**  SSL version 3. The third and currently latest version of SSL.

**TCP**  Transmission Control Protocol. Protocol often used in combination with IP create a reliable connection.

**TCG**  Trusted Computing Group. Organization developing the TPM.


**TPM**  Trusted Platform Module. Used both when referring to a specification of a cryptographic hardware module and an implementation of the specification.

**URI**  Uniform Resource Identifier. A short string to identify a resource.

**WA**  Wassenaar Arrangement. Convention dealing with export issues.

Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Bengt Werstén