The Transport Layer Security protocol

An overview

Cryptography, Spring 2009

David Volquartz Lebech

June 11, 2009

Contents

1 Introduction

Security is one of the biggest issues on the Internet and an area of enormous ongoing research. When we shop online or log into our web banking account, we want to be sure that no one can pick up sensitive information about us like our credit card number and we want this security to be transparent and ubiquitous whenever we share sensitive information.

Cryptography plays a major role in helping to prevent eavesdropping of personal information. One of the most widespread protocols used for secure communication on the Internet is the Transport Layer Security (TLS) protocol, previously known as the Secure Socket Layer (SSL) protocol. In this paper, I will explore the popular protocol, giving an overview of how it works and what algorithms it uses.

The paper has two main parts. In section 2, I will first describe the TLS protocol itself to provide insight on how the protocol works and how secure it is. In section 3, I will describe two particular algorithms that are often used in TLS. I will not go deep into any specific technical details and I will not supply any implementation. This paper should thus be seen as an introduction to the TLS protocol and not as a technical documentation.

1.1 Motivation

Every time we shop online, log into our web banking account or even check our emails, we are usually using a secure connection to communicate passwords and other information. In most cases, the https protocol is used for secure communication which is easily seen in the beginning of the website address and the typical lock symbol that most web browsers use to represent secure browsing. Https is in fact an implementation of the TLS protocol and TLS is thus a de facto standard for secure communication on the Internet [3, page 314]. For this reason, I think that it is interesting and relevant to explore this topic in more detail because it is greatly related to cryptography.

1.2 Background

The SSL protocol was originally developed by Netscape. The most recent version is 3.0 [6] which was released in 1996. The SSL protocol is now considered obsolete and has since been replaced by the TLS protocol which is currently in version 1.2 [5] and is maintained by the Internet Engineering Task Force (IETF). The goal of this protocol is to provide:

1. Cryptographic security

2. Interoperability
3. Extensibility
4. Relative efficiency

In a cryptographic context, *cryptographic security* is naturally the most interesting and I will only focus on this part of the protocol throughout this paper. I will thus not go into detail with the efficiency of the protocol even though this is sometimes also an interesting aspect from a cryptographic point of view.

2 TLS Protocol description

The TLS protocol consists of two layers which are interesting in each their way because they provide different functionality but they are still related and not independent. I will describe each of them separately. The description is based on the original request for comments (RFC) [5] as well as digested versions found in [3, section 7.6.3] and [11, section 10.7].

2.1 TLS Record protocol

The TLS Record protocol layer is positioned just above the Transport layer of the network which usually uses the TCP protocol. The record layer is responsible for receiving and encrypting messages from layers above it (see section 2.2) so that the messages can be sent with the Transport layer, e.g. the TCP protocol. The environment in which the record protocol operates is called a *connection state*. This state contains the information necessary for the record protocol to process a message from above layers and send it over the wire. This information includes a specification of the currently used compression, encryption and message authentication code (MAC) algorithms.

The record protocol works in the following way:

1. A message M is passed to the record protocol.
2. M is decomposed into smaller $TLSPlainText$ blocks.
3. The blocks are optionally compressed into $TLSCompressed$ blocks.
4. The blocks are encrypted to produce $TLSCiphertext$ blocks. This is done using either a stream cipher, a block cipher or an Authenticated Encryption with Associated Data (AEAD) cipher.
5. The blocks are transferred as e.g. TCP packets.

2.2 TLS Handshake protocol

Above the record protocol layer is the handshake protocol layer. The handshake protocol layer has three sub-protocols. A protocol for alerts, one for changing ciphers and the handshake protocol. The first two are not important in this context.

The handshake protocol is responsible for setting up a secure connection between the client and the server and negotiating the terms of the connection. This includes agreeing on a protocol version, which algorithms to use and a pre-master secret (see section 2.3) as well as passing necessary parameters to the record protocol and an optional authentication of each end of the connection. [5, 3] both provide good descriptions of the actual handshake, i.e. the messages passed back and forth between client and server.

Of particular interest for us is the negotiation of the cryptographic algorithms to be used for a secure connection. The algorithms are collectively gathered in so-called *cipher suites*. More specifically, a cipher suite consists of an algorithm for key exchange, encryption and MAC as well as a pseudorandom function (PRF). I will elaborate a little on cipher suites and how they are negotiated in the next section.

2.3 Overall security procedure

While the two previous sections briefly described the actual parts of the TLS protocol, the overall procedure of interest in the cryptographic context can be summarized as such:

1. A client wants to set up a connection with a server.
2. The server sends its public key to the client and the client authenticates the key. The system is vulnerable at this point because no secret key has been chosen yet. Thus, the client has preferably already received the server's public key from a trusted authority (see section 2.5.1).
3. Using a public key cryptographic algorithm, a *pre-master secret* key is generated and sent. This can happen on both client and server side.
4. Both client and server calculates a master key from the pre-master key.
5. The client and server communicates securely using the master key and a symmetric key cryptographic algorithm as well as an algorithm for message authentication (the MAC algorithm).

2.4 Algorithms in TLS

The TLS protocol does not define any algorithms itself. Rather it uses well-known cryptographic algorithms that are agreed upon between client and server through the cipher suites. The exact cipher suites are defined in [7] and include the algorithms:

Key exchange: RSA, Diffie-Hellman, KRB5, PSK, ECDH, SRP and others.

Encryption: 3DES, AES, IDEA, CAMELLIA, RC4 and others.

MAC: SHA, MD5 and others.

In other words, TLS supports quite a few different algorithms. In section 3 I will describe two of them.

2.5 Security of TLS

Although TLS is generally considered a secure protocol (hence its widespread use on the Internet), there is a few potential threats. In this section, I will cover possible attacks on TLS.

2.5.1 Man-in-the-middle attack

A man-in-the-middle attack refers to a method of active eavesdropping a connection between a client and a server that use public key cryptography. The attack can be summarized like this (using our usual protagonists: Bob, Alice and Eve):

1. Bob wants to send a message to Alice and therefore requests Alice's public key.
2. Alice sends her public key to Bob.
3. Eve intercepts the message and replaces Alice's public key with her own.
4. Bob encrypts his message with Eve's key, thinking it is Alice's key.
5. Eve intercepts Bob's message, decrypts it, encrypts it with Alice's key and sends the message to Alice who will not be suspicious.

This attack is conceptually and practically "easy" to execute. It "only" requires access to the communication channel and the possibility of tampering with messages. Since TLS uses public key cryptography for key exchange, it is potentially a security threat.

According to the protocol description [5], this attack can be prevented if the server is required to supply a certificate of authenticity from a mutually trusted authority (e.g. a certificate authority) as mentioned earlier. On the other hand, TLS also supports totally anonymous sessions and these are "inherently vulnerable" [5, section F.1.1] so these should be avoided for sensitive transactions.

2.5.2 CBC Block Cipher attack

Under certain circumstances it is possible to compromise the TLS protocol when cipher block chaining (CBC) block ciphers are used for encryption. [8] describes an attack that needs to be considered: Earlier versions of TLS used two different error codes for an incorrect MAC record and an incorrectly decrypted ciphertext. The idea of the attack is to exploit the fact that a block cipher pads values to the end of a block to give them a certain size. This could potentially be used by Eve in the following way:

1. Eve picks up two consecutive ciphertexts, makes a guess of the last bit of the plaintext, alters the message, and resends a fake ciphertext.

2. If the guess was wrong, Eve will get a decryption_failed error.

3. If the guess was correct, then the last bit will look like proper padding of the block and Eve will get a bad_record_mac instead. In other words, a small part of the plaintext has been guessed correctly.

The attack has been partly invalidated because the error code decryption_failed is not used in TLS 1.2 anymore. Also, there is no direct access to error codes so the attack is based on the assumptions that the attacker can gain access to at least logfiles or similar information to reveal the error message and that the same message is repeatedly encrypted and sent.

In [2], a timing attack is described that share some of the ideas with the attack just described. The attack exploits the fact that in some implementations of TLS, the MAC is not calculated if the padding of a ciphertext is wrong. The attacker can measure the time it takes to compute a MAC to deduce whether or not the padding was correct. This attack is also based on a lot of assumptions but should still be taken seriously. A way of avoiding the

attack is to just calculate a MAC even though the padding of a ciphertext is wrong, as suggested in [5, 8].

3 Cryptographic algorithms

During the term we have treated modern cryptography mainly with focus on public key cryptography in the form of e.g. the RSA and Diffie-Hellman algorithms. We have not talked so much about symmetric key algorithms like AES or MAC algorithms like SHA except for some historical encryption ciphers. The TLS protocol uses three cryptographic algorithms (the cipher suite) to achieve its security. In this final section, I will explore two contemporary and important cryptographic algorithms (AES and SHA) and try to relate them to TLS.

3.1 AES

The Advanced Encryption Standard (AES) algorithm is a symmetric block cipher algorithm that was originally invented by Joan Daemen and Vincent Rijmen. It was selected in 2000 to be the new official encryption standard of the US, replacing the aging Data Encryption Standard (DES) [4, introduction].

The algorithm can process 128 bit (i.e. 16 byte) blocks using a key of either 128, 192 or 256 bit length. The 16 byte block is represented by 4 by 4 matrix and the algorithm works by substitution and permutation of the bytes in the array. This also means that a block of ciphertext has the same length as a block of plaintext [9].

3.1.1 Mathematical foundation

The AES algorithm uses the arithmetic on the finite field $GF(2^8)$ which is nicely described in [11, section 3.11.2]. All elements in this field can be represented as 8-bit bytes. For example the byte:

$$b_7b_6b_5b_4b_3b_2b_1b_0 = 11001100$$

can be described by:

$$1x^7 + 1x^6 + 0x^5 + 0x^4 + 1x^3 + 1x^2 + 0x + 0$$

The addition of two bytes in this field is the exclusive or operation (XOR). In AES, we are working modulo an irreducible polynomial:

$$x^8 + x^4 + x^3 + x + 1$$

which makes multiplication a bit tricky and I will not describe it here. A method for multiplication is described in both [11, section 3.11.2] and [9].

3.1.2 Procedure

The AES algorithm can be described by the following steps (collectively called a *round*) applied to a single 16 byte block [9, 11]:

1. A non-linear byte substitution called the `SubBytes` transformation. In this step each byte is substituted for another byte that is found in a lookuptable called S-Box. The S-Box is constructed by taking the multiplicative inverse over $GF(2^8)$ and applying an affine transform to the bits. The S-Box is shown in appendix A.5. The substitution byte is found according to the hexadecimal number for the byte. A byte with value e.g. 53 will be substituted for the value in the S-Box at row 5, column 3.

2. A cyclical shift of the bytes in a row, applied to the last 3 rows (out of 4) of the block. This is called the `ShiftRow` transformation. More specifically, bytes in row 2 is shifted 1 place to the right, in row 3 2 places and in row 4 3 places.

3. A column-by-column transformation called the `MixColumns` transformation. Each column is considered a polynomial in $GF(2^8)$ and is multiplied with a fixed polynomial $3x^3 + 1x^2 + 1x + 2$ modulo $x^4 + 1$.

4. A round key is derived from the key. This roundkey is a 4 by 4 matrix, i.e. same size as the block. In the `AddRoundKey` transformation, this roundkey is XOR'ed with the block.

In appendix A, I have supplied figures that show the intuition of the above four steps. For a 128 bit key, the encryption with AES is exactly as such:

1. Do an `AddRoundKey` step.

2. Loop the above 4 steps 9 times.

3. Run the above 4 steps 1 time but without step 3, i.e. the `MixColumns` step.

In each iteration of the algorithm just outlined, a new round key is used. A so-called key schedule is created from the initial secret key. The secret key is also represented as a 4 by 4 matrix and via a key expansion procedure,

extra columns are added for a total of 44 columns. The key schedule is thus a 4 by 44 matrix. The round key is selected from this matrix. I will not go into detail with the procedure of key expansion here.

3.1.3 Security and use in TLS

There seems to be only one successful attempt at breaking the AES algorithm, namely timing (or side-channel) attacks, one of them which is described in [1]. The attack is based on the basic assumption that lookup times in an array depend on the array index (i.e. in the S-Box). Furthermore, if this is correct then the execution time for the whole algorithm is variable and in correspondance with the table lookups which, in turn, will leak information about the key.

The attack can be avoided with the correct implementation of AES, i.e. constant-time table lookups. And as with a lot of other attacks, it is still based on assumptions that only hold under certain conditions, for example a server leaking too mich information. It is thus reasonable to conclude that AES is a secure algorithm which probably explains why it is often used in the TLS protocol.

3.2 SHA

The Secure Hash Algorithm (SHA) belongs to a group of algorithms known as *hash functions*. A hash function converts a message into a *message digest* of a fixed length which is similar to a ciphertext. The hash function should have the following properties [11, section 8.1]:

1. One-way. Contrary to a symmetric key algorithm, a hash function is meant to be a one-way function. It should thus be very hard (or impossible) to reverse the hash function and find the original message for a given message digest.

2. Fast. The hash function should be able to calculate a message digest fast.

3. Collision-free. The hash function should avoid hashing different messages to the same value.

SHA is an official US standard called Secure Hash Standard and the latest version of the standard definition can be found in [10]. There are several different SHA hash functions available based on the standard. In this section, I will describe the SHA-1 function. The description will be based on [10,

section 6.1] and [11, section 8.3] and I will not reference these further in this section.

3.2.1 Mathematical foundation

The mathematical background needed to understand SHA is not particularly difficult. The basic tools needed are actually just the usually defined bitwise operators $\wedge$ (and), $\vee$ (or), $\oplus$ (addition) and $\neg$ (negation) as well as $X \hookleftarrow r$ (shift X r bits to the left) and $+$ which is addition modulo 2^{32}. We then define 80 functions f_t like this:

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & \text{if } 0 \leq t \leq 19 \\ B \oplus C \oplus D & \text{if } 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{if } 40 \leq t \leq 59 \\ B \oplus C \oplus D & \text{if } 60 \leq t \leq 79 \end{cases}$$

and 80 constants K_t:

$$K_t = \begin{cases} \texttt{5a827999} & \text{if } 0 \leq t \leq 19 \\ \texttt{6ed9eba1} & \text{if } 20 \leq t \leq 39 \\ \texttt{8f1bbcdc} & \text{if } 40 \leq t \leq 59 \\ \texttt{ca62c1d6} & \text{if } 60 \leq t \leq 79 \end{cases}$$

3.2.2 Procedure

The SHA-1 algorithm can be described by the following steps, given a message m:

1. Preprocessing: Pad m with a 1 bit and several 0 bits if necessary and split the message into n 512 bit blocks m_i where $i = 0, \ldots, n-1$. Set initial hash values H_0 through H_4. I will leave out the details for this step (i.e. the concrete values).

2. For each message block m_i, do the following:

   (a) Define a message schedule W_i, each of which has 32 bits. Let $m_i = W_0 || W_1 || \cdots || W_{15}$, i.e. m_i is the concatenation of the first 16 message schedules. For $t = 16, \ldots, 79$ we have:

   $$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \hookleftarrow 1$$

   (b) Let $a = H_0$, $b = H_1$, $c = H_2$, $d = H_3$ and $e = H_4$.

   (c) For each t between 0 and 79, do the following:

i. $T = a \hookleftarrow 5 + f_t(b, c, d) + e + K_t + W_t$

ii. $e = d$

iii. $d = c$

iv. $c = (b \hookleftarrow 30)$

v. $b = a$

vi. $a = T$

(d) Let $H_0 = H_0 + a$, $H_1 = H_1 + b$, $H_2 = H_2 + c$, $H_3 = H_3 + d$ and $H_4 = H_4 + e$.

3. Output the message digest $H_0||H_1||H_2||H_3||H_4$.

3.2.3 Security and use in TLS

In TLS, keyed MAC algorithms are used for message integrity [5, section 5]. In particular a certain type of hash functions, keyed hash functions, are used which are called HMAC. SHA can be used as a keyed hash function in a special HMAC-SHA version and is thus not used directly since the SHA itself does not use a key.

The SHA algorithms are generally considered safe. This means that there is no method for finding a collision or reversing the hash process and SHA is thus widely used in TLS and on the Internet in general. However, it seems that recent research has shown an improvement in finding collisions in the SHA-1 algorithm. They are still not efficient enough to be practical but they are faster than brute-force. One method is described in [12] but describing the method is beyond the scope of this paper. For most application, SHA-1 still offers great security as a secure hash function but it will be interesting to see how long this will last.

3.3 Relationship of TLS algorithms

In section 2, I briefly described the TLS protocol and the cipher suites that contain the three important cryptographic algorithms that together form the backbone of the security in TLS. As should be apparent from the above descriptions, the three algorithms (public/symmetric key and hash function) are different in their design but share the same basic idea about security, namely altering a message so that it becomes unreadable.

In TLS, the hash function is primarily used for message integrity purposes. Since the process is irreversible, a hash function cannot be used for encryption. It would not make sense to encrypt without being able to decrypt. The public-key algorithm is only used for the secure exchange of a

key to be used by the symmetric key algorithm. This makes sense because public key algorithms are significantly slower than symmetric key algorithm [11, section 1.1.2]. It seems logical that TLS uses three different types of algorithms because they have each their strengths and weaknesses, thereby improving the overall security of the protocol.

4 Conclusion

In this paper, I have described the Transport Layer Security (TLS) protocol. I have given an overview about the cryptographic security of TLS and how this security is established. Furthermore, I have described two popular algorithms in more details, namely the symmetric key algorithm AES and the secure hash function SHA-1.

TLS can generally be considered a secure protocol. It is a hybrid cryptosystem, meaning that it uses both symmetric and public key algorithms to achieve its security. However, the security does not only come from the protocol itself. A quote from the RFC for TLS [5]:

> The system is only as strong as the weakest key exchange and authentication algorithm supported, and only trustworthy cryptographic functions should be used.

In other words, the continued improvement of cryptographic methods and algorithms is the most important way of keeping TLS secure as well as well-implemented version of the protocol on servers and in browsers. This is very important since TLS is a de facto standard for secure communications on the Internet.

One of the unexpected outcomes of my research for this paper was to realize that seemingly advanced protocol and algorithmic descriptions (i.e. the RFC and FIPS documents) are surprisingly much easier to read than I initially thought. This might sound like a trivial fact but indeed I feel that this is one of the most valuable lessons I have learned from this project. Although there is definitely more advanced protocols and algorithms out there, TLS is certainly not the least advanced of them all.

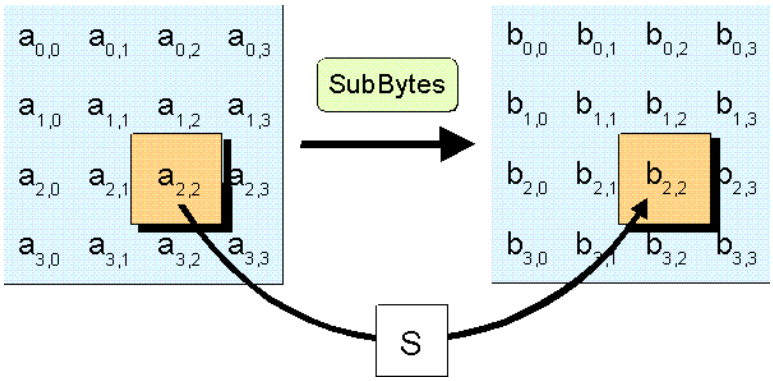
During the term, we have been mostly concerned with modern public key cryptography. By looking into TLS, I have gained valuable knowledge about other contemporary methods for cryptographic security, particularly on the Internet, which will benefit me in future projects.

A AES

All figures and tables in this section are supplied only for quick reference. I have not produced them myself but all have clear source reference.

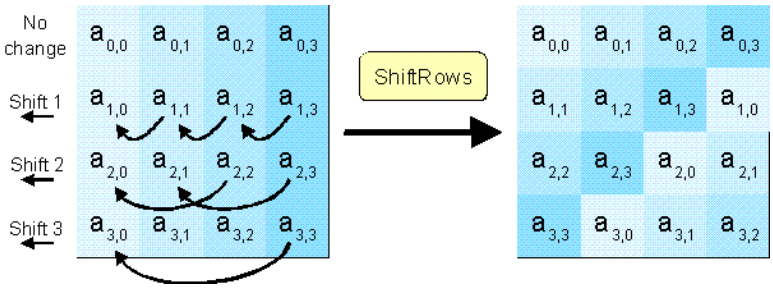
A.1 SubBytes

Source: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard



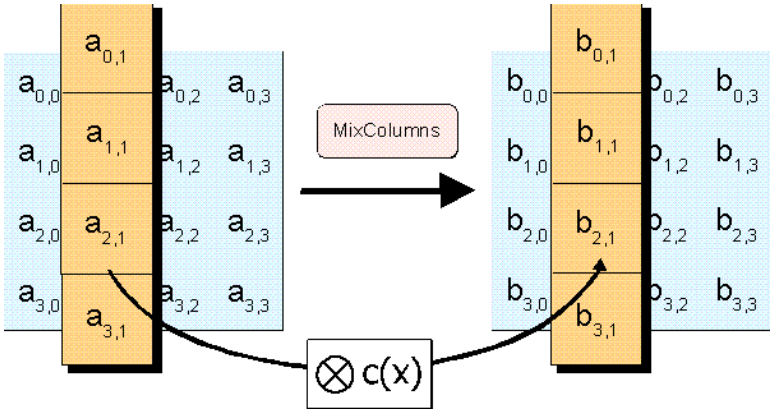
A.2 ShiftRows

Source: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard



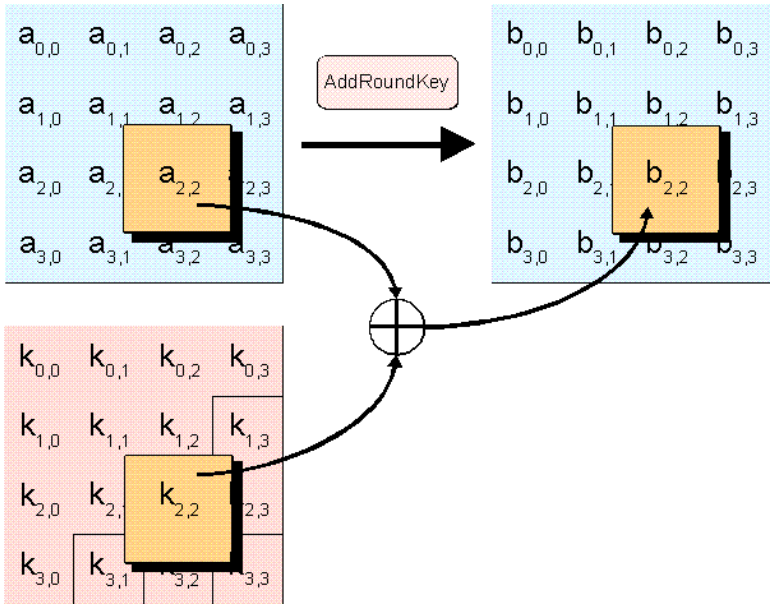
A.3 MixColumns

Source: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard



A.4 AddRoundKey

Source: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard



A.5 S-Box for AES

Source: http://en.wikipedia.org/wiki/Rijndael_S-box

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

References

[1] Daniel J. Bernstein. Cache-timing attacks on AES. `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`, 2005.

[2] Brice Canvel, Alain Hiltgen, Serge Vaudenay, and Martin Vuagnoux. Password interception in a SSL/TLS channel. In *Advances in Cryptology - CRYPTO 2003*, volume 2729, 2003.

[3] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems Concepts and Design.* Addison-Wesley, 4th edition, 2005.

[4] Joan Daemen and Vincent Rijmen. *The design of Rijndael, AES - The Advanced Encryption Standard.* Springer-Verlag, 2002.

[5] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol, Version 1.2*, August 2008. `http://tools.ietf.org/html/rfc5246`.

[6] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL protocol, version 3.0. Technical report, Netscape Communications Corp., November 1996. `http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00`.

[7] IANA. *Transport Layer Security (TLS) Parameters.* `http://www.iana.org/assignments/tls-parameters/`.

[8] B. Möller. Security of CBC ciphersuites in SSL/TLS: Problems and countermeasures. `http://www.openssl.org/~bodo/tls-cbc.txt`.

[9] National Institute of Standards and Technology (NIST). *FIPS 197: Advanced Encryption Standard (AES)*, 2001.

[10] National Institute of Standards and Technology (NIST). *FIPS 180: Secure Hash Standard (SHS)*, 2008.

[11] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory.* Pearson Prentice Hall, 2nd edition, 2006.

[12] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *In Proceedings of Crypto*, pages 17–36. Springer, 2005.