NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

TTM4135 APPLIED CRYPTOGRAPHY AND NETWORK SECURITY, SPRING 2022

Web Security Lab
Group 1

Martin J. Nilsen Max T. Schau Zaim Imran

April 21, 2022

Contents

1 Introduction **1**

2 Tasks **1**
2.1 Part I: Encrypted email 1
2.2 Part II: Installing a web server and obtaining a certificate 4
2.3 Part III: Examining TLS Traffic 7

3 Cooperation and Experience **8**

1 Introduction

As the lecture period in the subject of TTM4135 Applied Cryptography and Network Security comes to an end, the course transitions over to group based lab work. In the lab we are supposed to dive deeper into the material, getting a more hands on experience of the practical side of cryptography. This report is the final result after working through the lab for the last weeks.

2 Tasks

2.1 Part I: Encrypted email

> **Q1:** Generating a symmetric key k just for encrypting that one message seems like an unnecessarily complicated step. Why does GPG do that, instead of just encrypting the message with p?

A: Symmetric-key encryption is used whenever two parties wants to communicate. They share a secret k which is used for both encryption and decryption. When they have agreed on the secret k, they can use an encryption algorithm E with the secret k to encrypt a message m, such that $c = E(k, m)$. Then, the other part may decrypt the message using a decryption algorithm, such that $m = D(k, c)$ [6]. Public key cryptography, on the other hand, uses both a private and a public key, such that each person involved has a private and a public key. If Alice wants to send a message to Bob, Alice must encrypt message m with Bob's public key. Bob can then decrypt the message with his private key [9].

GPG uses a hybrid cipher, meaning that it will use the public key of the recipient to encrypt the shared symmetric key. When sending a message the message being sent is first encrypted using the shared key, then the shared key is also encrypted using the recipients public key. Both the encrypted message and the encrypted shared key is then combined into one package and sent to the recipient. When the recipient receives the package, the symmetric key is decrypted using the recipients own private key, before using the newly decrypted symmetric key to decrypt the message. By doing this, GPG obtain a session key. This is an advantage, since an attacker can only use the session key once for that one message, and not every other message in the future [10].

> **Q2:** How many bytes does PGP use to store the private signing and public verification keys for each of the two signature types?

A: To find the private signing and public verification we used two different approaches. One approach is to export the keys to a file through the user interface of the GPG Keychain

application, available on Mac. In the details section of each listing, the type of key is also listed. The second approach, available on all operating systems, is to use the command line interface. First you would need to have the fingerprint of the key, which can be acquired by running the following command

```
$ gpg --list-keys
```

For listing the public key, we could then use the following command

```
$ gpg --armor --export [Fingerprint]
```

And for listing the private key, we simply change the last flag as done below

```
$ gpg --armor --export-secret-key [Fingerprint]
```

After acquiring the key size, we could simply count the characters as one character equals one byte. It should also be mentioned that as all of the members had similar results, the group decided to only display the results of one of the members to prevent redundancy.

	RSA		DSA	
	Private	Public	Private	Public
Size in bytes	1813	881	1465	1353

Table 1: Key size RSA, DSA

Q3: How many bytes does PGP use to store the signatures in each of the four cases (both long and short messages)?

A: For solving this task, we would need to sign two files of size 1KB and 1MB, with both RSA and DSA, and check the output size in bytes. Furthermore, we have to keep in mind the type of signing we use. We chose to use the `--detach-sign` flag, which gave us the advantage of storing the signature as a separate file, rather than the message being concatenated to the signature.

```
$ gpg -u [Fingerprint] --output [out.sig] --detach-sign [message.txt]
```

Using this command for all four cases, alternating between two fingerprints and two message lengths, we got the following results.

	Short message	Long message
RSA	310 bytes	310 bytes
DSA	119 bytes	119 bytes

Table 2: Signature size

Q4: How long, on average, does PGP uses for signature generation and verification in each of the four cases?

A: To solve this question, we utilized the `time` command available on unix-based systems, by using the following line of code

```
$ time gpg -u [Fingerprint] --output [out.sig] --detach-sign [message.txt]
```

For the verification we used the command

```
$ time gpg --verify [fileNameD.sig] [message]
```

Yielding the following results

	RSA		**DSA**	
	Short message	Long message	Short message	Long message
Signing time	0.01s	0.02s	0.01s	0.015s
Verification time	0.01s	0.015s	0.01s	0.015s

Q5: Discuss your results for the above three measurements. In particular, how well do they correspond to what you expect from what was studied in the lectures? Include an explanation of how the different elements of the keys are stored (such as modulus, exponents, generators).

A: When it comes to the size of the private and public key of RSA, we initially expected that the private key would take more space compared to the public key, which was in fact confirmed. When generating the public and private key, we started with finding two large distinct prime numbers p and q. Secondly, we calculated $n = pq$, which is used as the modulus for both keys. For the third step, we calculate $\phi(n) = (p-1)(q-1)$. The exponent e must be chosen such that $1 < e < \phi(n)$ and $gcd(e, \phi(n)) = 1$. When e is chosen, we must compute $d = e^{-1} mod\ \phi(n)$ as the fourth step. In the end, the public key will consist of the pair (n, e), while the private key will consist of (p, q, d) [4]. In addition, the public exponent e is often quite small (normally 3 or $2^{16} + 1$), meaning that the public key will consist of a quite small number e and a large number n. As mentioned in the lecture slides, the private key consists of the elements of the public key and more, and thus it makes sense that it is of larger size.

Regarding the size of the public and private key of DSA, the result was pretty much expected as well. DSA stands for Digital Signature Algorithm, and is based on the difficulty of the discrete logarithm problem. We can see from table 1 that both the private and public key of DSA are almost equal in size. This is expected due to the fact that they both are consisting of one single parameter.

From table 2 we can see that the the signature size was the same for both message lengths. This was expected because the `--detach-sign` flag excludes the message from the signature file. We can however see a significant decrease in size from RSA to DSA (310 bytes vs 119 bytes). This is because we are restricting the calculations to a subgroup of Z_p^* for DSA [5]. The length of 310 bytes of the RSA signature makes sense. The size of a RSA signature is equal to the size of the modulus converted to bytes. Thus, as the modulus is 2048 bits, the signature has a size of 256 bytes. However, it seems that GPG stores a bit more, which we assume is because it stores some additional information in the signature. This also seems to be the case for DSA.

The fact that the signing with DSA is quicker compared to RSA makes sense because there are simpler calculations for DSA, and thus it is expected for it to be faster to calculate. It is however expected that the verification process is more time consuming for DSA compared to RSA. We can see from slide 13 and 21 from the lecture slides [5] that the DSA verification process is more complex, and therefore it is more time consuming. This is not the case for our data, which may be due to the setup of our computers or other external reasons.

> **Q6:** What assurances does someone receive who downloads your public key from the server where you have uploaded it? What is the difference between the role of the certification authority in X509 and the key server in PGP, with regard to the security guarantees they give?

A: There are two main ways in which Public Key Cryptography Infrastructure (PKI) is implemented. One using certificates and certificate authorities, described in the X509 standard, and one developed with a goal of not requiring a centralized CA, but instead rely on trust relationships between regular users. With the X509 standard, the certificates are only issued by a professional CA, which provide trust, and is maintained by an organization. With PGP on the other hand, the certificates can be signed by anyone, but is verified with a UserID using the security fault tolerance mechanism *Web of Trust*. This mechanism was designed to compensate for the fact that issuers were not protected or professional, and instead builds trust by utilizing a decentralized trust scheme in a graph structure.

This naturally brings us over to what assurances are given when downloading our public key from the server. The assumption in focus is that a certificate binds a key to a person. The assurance is therefore that the key signer is a real person that has been verified by Web of Trust. With this, we know that the person has an unique UserID, and that the email of this user have been verified.

2.2 Part II: Installing a web server and obtaining a certificate

> **Q7:** Who typically signs a software release like Apache? What do you gain by verifying such a signature?

A: A software release is signed by the creator of the software. This signature is used to verify that the software received is not altered in any way and that the creators indeed are who they claim to be. The software could have been altered with either accidentally via a faulty transmission channel, or intentionally with or without malicious intent. The signature is a way of making sure that the software is in fact what was originally signed, and therefore is not modified.

> **Q8:** Why did you obtain a certificate from Let's Encrypt instead of generating one yourself?

A: This is because Let's Encrypt is a trusted certificate authority while we are not. If someone were to look up our certificate issued by Let's Encrypt to verify our legitimacy, they would go through similar steps as in question 9. They would then need a trusted certificate authority to vouch for our certificate. If we had made this on our own, we would not have anyone vouching for us.

Q9: What steps does your browser take when verifying the authenticity of a web page served over `https`? Give a high-level answer.

A: There are a few steps the browser takes to verify authenticity of a web page. These are:

- The browser receives the web page's certificate. This certificate is signed by a trusted authority (CA). It then checks this certificate/public key by its own list of trusted public keys (certificate authorities). It should also be mentioned that every browser has a bank of these packaged with the browser install
- The domain or IP-address of the web page trying to be accessed is contained within the certificate sent by the web page. The browser then asks the certificate authority if the domain it is connecting to is the same as in the certificate
- As the browsers pack the certificate authorities during installation, one may think what happens if one of these are revoked - will there be a need to get all the users to promptly update their browser? This is not something you can rely on, as some may run software for many years before updating. Therefore, in addition to checking the catalog of certificate authorities, the browser checks the certificate revocation list (CRL). This is a record of digital certificates that have been revoked by the issuing CA before their actual or assigned expiration date, indicating which certificates that should no longer be trusted. This way, we do not solely rely on the list locally stored in the browser, but we check if the certificate is revoked or not by querying the CRL

Q10: Have a look at the screenshot. What does the string `TLS_DHE_RSA_WITH_AES_128_CBC_SHA` in the bottom left say about the encryption? Address all eight parts of the string.

A: The string `TLS_DHE_RSA_WITH_AES_128_CBC_SHA` is the given cipher suit of the protocol. A teardown of the different aspects of the given cipher suit follows.

Transport Layer Security (TLS) is a communication protocol that provides authentication, privacy and data integrity between two applications.

Diffie–Hellman Ephemeral (DHE) is a key exchange method that can be used to securely share a secret using solely public communications [1]

RSA is a public key cryptosystem, with a distinct public and private key. RSA can be used for both encryption/decrpytion and signing/verification. In the DHE_RSA cipher-suite, RSA is used to sign key exchange messages from the server, which are subsequently validated as authentic by the client using the public key [4]

With means that we are using a hybrid cryptosystem combining RSA and AES.

Advanced Encryption Standard (AES) is a specificiation of encrypting block ciphers.

128 described in the string specifies the amount of bits used for the encryption key in AES.

Cipher Block Chaining (CBC) is a technique for chaining blocks toghether. One start with a random initialization vector IV which is we apply a XOR-operation on with the first block of plaintext P_0 before one apply the encryption. Then, this is subsequently used for XOR with the next block of plaintext P_1 and so on [2]

Secure Hash Algorithm (SHA) is a hashing algorithm used in cryptography. It is based on the MDx family with a 160 bit output. [3]

Q11: The screenshot in figure 1 is obviously from a few years ago. Do you think the encryption specified by this string is still secure right now? Motivate your answer.

A: It is clear that the encryption is not secure anymore. First of all, the encryption is using SHA-1 which was proven to have collisions in 2017 [3]. Therefore, there is a chance that two distinct values may get the same hash value. AES-128 is, however, still considered secure. A faster supercomputer would need 1 billion billion years to go through all possible combinations of a 128-bit key [8].

In 2021, an article within the .Net docs, Microsoft argued that decryption with CBC is no longer considered safe. This is because of what is called a padding oracle attack. An oracle is basically information to an attacker on whether their action was correct or not. An attacker may use such a padding oracle to slightly modify the message sent, and repeat the process until the data becomes correct. Thus, an attacker may continue until the message is decrypted [12].

Q12: What is the purpose and format of an SCT certificate field? Why might a certificate with an SCT not appear in any certificate transparency log?

A: The purpose of a Signed Certificate Timestamp (SCT) in the certificate is to indicate when the issuance occurred, how to find it using signature data and which Certificate Transparency log it was recorded in. This makes it possible for the client to verify that the certificate is legit by verifying that the timestamp is correct.

The format of the SCT is as follows:

- Version: 1
- Log ID: 6F 53 76 AC 31 F0 31 19 D8 99 00 A4 51 15 FF 77 15 1C 11 D9 02 C1 00 29 06 8D B2 08 9A 37 D9 13
- Timestamp: Thu, 17 Mar 2022 09:53:44 UTC
- Signature algorithm: SHA-256 ECDSA
- Signature data: 71 byte : 30 45 02 20 09 05 63 49 AA BD 19 C7 22 AB A3 A1 25 D6 65 A6 7A D9 CC F1 72 DC 23 FC 15 C7 A6 E0 C3 67 A0 DD 02 21 00 98 D9 8F C7 75 3F D3 E0 AF 33 D4 A8 DA 9B C1 D7 01 69 AF 0A 76 1D 77 41 64 F7 5C D1 96 4D 1F AE

For testing purposes, we converted the Log ID from hex to base64 and searched up which log this was, and found that it was called *Sectigo 'Mammoth' CT log*, which corresponds to the field *log operator* containing the value *Sectigo* in our certificate.

According to Certificate Transparencys own report How CT works [7], the certificate needs

a SCT before a CA can log it, but the certificate needs to have been submitted to the log for acquiring the timestamp. So how is this solved? Well, upon submission of a certificate to the log, the log responds with a SCT and a promise of adding the certificate to the log within a time period called the *Maximum Merge Delay*. This is usually 24 hours, with the intent of giving the operators time to fix possible errors before appending the certificate to the log. It is this merge delay we believe is the only possible way of having a certificate with an SCT not placed in a log, as there will be a timespan where it have the timestamp but are yet to be appended.

> **Q13:** What restrictions on server TLS versions and ciphersuites are necessary in order to obtain an A rating at the SSL Labs site? Why do the majority of popular web servers not implement these restrictions?

A: For achieving an A rating in the test at SSL Labs you have to have the following requirements fulfilled. The server have to support forward secrecy, while not supporting either SSL3, TLS 1.0 or TLS 1.1. Furthermore, the server shall not support RC4 and weak DH parameters less than 2048 bits, but support AEAD ciphers. There are more requirements, but these are the ones capping the grade at a B rating. We encountered most of these during the project ourselves, and had to fix them for achieving milestone 2.

The main reason why the majority of popular web servers do not achieve the grade A rating, seems to be because of the support of legacy protocols. Take for example `Facebook.com` and `Amazon.com`, they cap at grade B because of the support of TLS 1.0 and TLS 1.1. This is done as they prioritize availability over security, which is a tradeoff that makes them able to deliver their services to users on older outdated browsers too. If the business model of the company is based on availability, and can handle the tradeoff for a little less secure service, they tend to support these older protocols and will therefore cap their rating at an B. With that being said, there are servers out there with way worse ratings than that!

2.3 Part III: Examining TLS Traffic

> **Q14:** What are the values of the client and server nonces used in the handshake? How many bytes are they? Is this what you expect from the TLS 1.2 specification?
> What is the value of the encrypted pre-master secret in the client key exchange field sent to the server? Is this the size that you would expect given the public key of your server?

A: In each of the handshake we have a 32-byte Random Nonce that contains a 4-byte timestamp and 28-byte randomly generated value.

In Client Hello, Client Nonce:
6e6a4586264eb27f443a47dd81d554e94a1d4eaf576a779e522f5c42a6088746
In the Server Hello, Server Nonce:
5c8c01e8df8a40bcf5ded6262e894d8ccdfabbc0e5929303444f574e47524401

This is as expected since TLS uses a 32-byte random key in the handshake process [13]. Since we are using RSA these random bytes are used to generate the pre-master key and later encrypted with the server public key.
The encrypted pre-master key can be found in the Client Key Exchange and the value of the pre-master key is
2668642b1cabb5b3ca178ffb7ac3e876d63d353feebe5d93bfa18bdcc0a20b64
361caafe29d65ff71c177edb07494f4920ae0d8c898e600613a135bc3bbeca25

c0fad1b434d1b4a69bebad5391602c8159b7be7002669903afe9dd7951614ffd
ec82a4bbdb5838cc95d7c782d62b51417169a578e0451897f61aa4e9af52bf31
a8bbf00d02a329636ee043ae619b8d2c08c98d40fd6843e4a5eb1baec854fd52
a345576df9332a1b4218651d2a03e5282b7aabd3fa610372f3efef0637a0dfa6
27f9921d44e1deb9fba910c5afaa792bea920694ff266421fce6a6007632107e
a5e2bea2093b96a2004da7ba2f1c51352d138e34f893f4b596e019756b610bae
The pre-master key has a length of 256 bytes. Since we are using sha256 with RSA Encryption this is the expected size of the encrypted pre-master key.

> **Q15:** What was the value of the pre-master secret in the session that you captured? Is this the size (in bytes) that you expect from the TLS specification? Explain how Task 16 shows that this session does not have forward secrecy.

A: We found the master secret in the `SSLKEYLOGFILE` of length 46 bytes, with the value
`d06f4cc3dbc6b3e3f429ec3a6f939ece134f82c71f0e2a90`
`a073ba12d0240cca55f4378e8961f6a31a86f7bbb1262afd`
This is the expected length as mentioned in [11]. This session is not forward secrecy since the TLS_RSA_WITH_AES_128_CBC_SHA will use the previously negotiated keys and not generate one for each session. If an attacker has access to the master key they will have full access to any communication between the user and the server going back to when the key was generated. If a system has full forward secrecy it will create a session based master secret that is only able to decrypt messages sent in that session

3 Cooperation and Experience

Originally, the group consisted of four members, but Ole Jonas went to Japan on a student exchange program.
We have felt during the lab that the cooperation have gone quite well, much because of the former projects we have worked on together and the relation we have to one another. We agreed early on to work every Thursday in the given period, ensuring that we were able to complete all the milestones within the deadline. This worked out well, and we were able to collaborate on and discuss all the different aspect of the assignment as we were all working at the same time. We were all present physically when working with part 1, but on the day we were to proceed with the next parts, Max was home sick. The rest of the group took responsibility on completing milestone 2 and 3 that day, while Max had the responsibility of completing the questions from part 1 for the report. For the subsequent days, all of the members of the team were present, meaning that we could all collaborate on the remaining questions of the assignment. Thus, we ensured that we agreed on our answers on the different questions. We collectively feel that this has been a team effort.

This project have been both useful and challenging in different ways. Firstly, we found the practical approach of the lab quite motivating. Getting more of a hands on experience of the theory we have learned in the former months have made it easier to gain a better understanding of the curriculum. Secondly, we think that the tasks were the right amount of a challenge, where we did have to do some research and proceed to try (and fail) for getting the correct answers, but we also felt that the milestones were achievable to complete within the time frame given. Lastly, we are left with a feeling that the lab were interesting to work with, and we had both fun and felt mastery along the way. We had a great experience with the lab overall, and feel better suited for the exam to come.

References

1. Boyd, C. A. Lecture 11 - Public Key Cryptosystems based on Discrete Logarithms 2022.
2. Boyd, C. A. Lecture 6 - Modes of Operation and Random Numbers 2022.
3. Boyd, C. A. Lecture 7 - Hash functions, MACs and authenticated encryption 2022.
4. Boyd, C. A. Lecture 9 - RSA 2022.
5. De Kock, B. Lecture 12 - Digtal Signatures and Certificates 2022.
6. Delfs, H. & Knebl, H. in Introduction to Cryptography 11–31 (Springer, 2007).
7. How CT works `https://certificate.transparency.dev/howctworks/` (Apr. 1, 2022).
8. How Safe is AES Encryption? `https://www.kryptall.com/index.php/2015-09-24-06-28-54/how-safe-is-safe-is-aes-encryption-safe` (Mar. 24, 2022).
9. Public key cryptography `https://www.ibm.com/docs/en/ztpf/1.1.0.14?topic=concepts-public-key-cryptography` (Mar. 10, 2022).
10. The GNU Privacy Handbook `https://www.gnupg.org/gph/en/manual.html#AEN111` (Mar. 10, 2022).
11. The Transport Layer Security (TLS) Protocol Version 1.2 `https://www.rfc-editor.org/rfc/rfc5246#page-64` (Mar. 24, 2022).
12. Timing vulnerabilities with CBC-mode symmetric decryption using padding `https://docs.microsoft.com/en-us/dotnet/standard/security/vulnerabilities-cbc-mode` (Mar. 24, 2022).
13. TLS Handshake `https://wiki.osdev.org/TLS_Handshake#Handshake_Overview` (Mar. 24, 2022).