NTNU | Norwegian University of Science and Technology

TTM4135 - Applied Cryptography and Network Security

Web Security Lab

Group 42

Emma Lu Eikemo

Kamomilla Madelen Ekrem Berg Godlund

Håvard Solberg Nybøe

March 16, 2023

Introduction

This report will be contain the answers to the questions raised in the cryptography and network security project. The report is divided into three segments, one for each part of the project. The first part is about key signing, encryption, and authentication using GPG, while the second is about setting up a web server and ensuring high network security. Lastly, the third is about the TLS 1.2 protocol with details about cipher suites and the exchange of symmetric keys.

Part 1

> **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: As with RSA public key encryption is usually just a means to create a more secure communications channel to deliver a symmetric key. This is because symmetric keys are much less computationally expensive to use than asymmetric keys. Since we have already authenticated both users, it is more effective to use symmetric key further on.

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

A:

RSA private key	1943 bytes
RSA public key	933 bytes
DSA private key	1801 bytes
DSA public key	1412 bytes

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

A:

RSA short message	310 bytes
RSA long message	310 bytes
DSA short message	119 bytes
DSA long message	119 bytes

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

A:

	RSA	DSA
Short message signature	0.041s	0.023s
Short message verification	0.017s	0.022s
Long message signature	0.035s	0.032s
Long message verification	0.026s	0.031s

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: As expected [1], we notice that the RSA is faster at verification, while DSA is faster at signature, especially on the short messages.

RSA: The prime numbers p and q, can be multiplied to get $p \cdot q = n$ the modulus. To use the Chinese remainder algorithm for decryption and signing, we need to store p and q. Further, it is necessary to store the public and private key.

DSA: This algorithm stores two keys, one public and one private. The one who wishes to be authenticated creates a digital signature, using their private key, which can be decrypted with their public key, and thus proving their identity. In DSA a user may publish their pubic key, but must keep their private key to themself. The values p, q, and g are public and used in both the key generation and verification. A random value k, is chosen and kept secret during the key signing, while the signature in itself consists of two values r and s, which will then be stored by the verifier. During the verification w, u_1, u_2, and v are created as byproducts of the process, but not shared. Only the resulting v is partly shared as it is matched with r in the signature, and the result tells us if the signature was verified or not.

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: If someone downloads your key from the server, they can be reassured about your identity based on the web of trust. If someone in their net already signed your key, they can be more sure of your identity. The main difference between how a certification authority in X509 and a key server in PGP operate is that the CA is responsible for the signing of the certificates, while in PGP it is the users that are responsible. This means that for X509, there should be some official protocol for validation, which should guarantee the certificates they hold have been authenticated. Meanwhile since PGP is based on the web of trust, there is no guarantee that one of your friends is nice, and they can sign a lot of suspicious keys. We still wish to trust those we can though.

Part 2

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

A: The lead manager of the development team at Apache is responsible for signing the new releases, but in general it is either the development teams that generate the keys to do the signings, or the chosen certificate authority that generates so they can be used in the release. By verifying such a signature, you will be sure that the program you are about to download is legit. This can prevent malicious agents from using the name of well known software companies to trick you into installing malware.

Q8: Why did you obtain a certificate from Let's Encrypt instead of generating a self-signed one yourself?

A: Since anyone can generate a self-signed certificate we need certificate authority (CA), which in our case is Let's Encrypt. The CA can provide certificates such that others can check if they really are communicating with the server they think they are communicating with. Also, Let's Encrypt is free, and helps you configure a lot automatically.

Q9: What does Let's Encrypt have to verify, and how do they do it, before they issue a certificate?

A: Let's Encrypt needs to verify that they actually have control over the domain they requests certificates for. This is done by giving a set of tasks they that are designed to check the control over the domain. Let's Encrypt will also provide nonce they must sign to demonstrate they control the key pair. Then they can choose one of the challenges to complete, send information such that Let's Encrypt can verify the challenge, and send it with the signed nonce. When Let's Encrypt has verified the challenge and the nonce, it can issue a certificate.

Q10: What steps does your browser take when verifying the authenticity of a web page served over `https`? Give a high-level answer.

A: The browser needs to verify the web page's certificate and the of certificates of the hierarchy of CAs that provided the web page's certificate.

Q11: 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: It says what kind of algorithm the connection is using. **TLS**: Transport Layer Security. **DHE**: Diffie-Hellman key exchange to find symmetric key for the session. **RSA**: Rivest-Shamir-Adleman, public-key cryptosystem, used to verify the server (and possibly the client) in the connection. **WITH**: self explainable. **AES**: Advanced Encryption Standard for defining how to encrypt the data. **AES**: key size is 128 bit. **CBC**: defines the block cipher mode as Cipher Block Chaining. **SHA**: Secure Hash Algorithms 1, hash function to check if content is altered or not.

Q12: The screenshot 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: The cipher suite is now considered weak [2]. This is both because of the encryption (AES 128 CBC) and the hash (SHA). The Cipher Block Chaining (CBC) algorithms can be vulnerable to timing attacks, and should therefore be avoided. The hash algorithm Secure Hash Algorithm 1 (SHA) has been proven insecure and should not be used.

Q13: What restrictions on server TLS versions and cipher suites 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: In order to get obtain an A rating, SSL Labs requires at least TLS version 1.2, and a cipher strength higher than 256 bits [3]. The reason why a lot of popular web servers do not implement this, is probably because they wish to have backwards compatibility for older versions, that require less secure ciphers.

Part 3

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:

```
Client random [32]:
bc6e5ed1845134c88146875ff22972b165cfa6465f989dc908bd12acdcf48919
```

```
GMT Unix Time [4]:
Mar 6, 2070 17:44:01.000000000 W. Europe Standard Time

Server Random [32]:
e226aa3e3698c6ceaf6ec998b502c8f6151a398cd0ea6743827485947db254ff

GMT Unix Time [4]:
Mar 26, 2090 06:12:46.000000000 W. Europe Summer Time
```

The size of the nonces are as expected, as they both are 32 byte, where 4 byte are the timestamp, while the rest of the 28 bytes are the actual nonce.

```
pre master encrypted[256]:
| 63 95 04 7f 39 51 de c1 a1 d4 d5 85 1b fd 0b b5 |
| a6 89 66 6e 67 77 4d 79 69 5e 8c 17 19 07 ec c0 |
| db 45 33 a8 91 3a 2c 28 b1 ef 2d 00 d9 6d 84 94 |
| e3 6b 4b 95 5d e2 97 a5 b0 68 54 08 de 10 52 fc |
| aa 93 67 1d e9 81 c1 7e 4c cd 50 25 ab da 2e 83 |
| 66 de 29 ef 90 3c a1 6e c2 06 f0 28 d9 19 7b 34 |
| b1 68 89 d5 c8 63 85 b5 29 a7 cf e3 ca b4 47 b3 |
| cb c9 36 a5 b1 57 0c a4 69 bf 68 bc 30 27 7d 9c |
| 98 95 15 08 43 f1 a0 73 51 9d 16 89 99 37 fd e9 |
| 4a ce 25 c2 05 68 19 12 bc d3 ee 61 ce ff 26 8c |
| 23 db 42 0b d0 bd 9e 31 31 98 88 a8 97 aa b1 9f |
| c2 db fd 29 44 65 a8 1f ce 06 63 10 00 8d af 65 |
| db 2d 12 bb b3 0c 15 60 ca 8c 40 1b 84 ba 46 7d |
| d7 43 01 5b 10 23 0f fc 49 21 9d 08 c7 f3 8b 26 |
| 64 a5 e1 2b 3e 06 ed 5f e0 ca 57 4f 3c bb 72 0d |
| be 3e eb 89 c2 c7 28 bf c5 bc a5 5b f4 f1 ab fb |
```

The size given for the pre-master secret is as expected considering we are using RSA. The pre-master secret in itself should be 48 bytes, and the master secret we generate later will also be 48 bytes, but when we do RSA, we pad the plain-text with random bits up to the amount possible to encrypt. Since the RSA key of the server is 256 byte, it is natural that the encrypted pre-master secret is also 256 byte long.

> **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 15 shows that this session does not have forward secrecy.

A:

```
pre master secret[48]:
| 03 03 ba a0 57 20 98 99 b2 02 da 0b 99 06 de 97 |
| 41 2c db ce e6 1a 4e 00 f4 0e 45 f8 7c 3a d1 db |
| 9d 5b ab 5d e2 fd 38 80 83 e3 c0 38 4d 17 5e c5 |
```

The size is as expected, considering that the standard size for the pre-master secret is 48 bytes. As mentioned before, the encrypted pre-master secret was longer due to the cipher we used to encrypt it, so it is natural that it goes back down to 48 when it gets decrypted, and we remove the padding. The session does not have forward secrecy, as any actor with access to the servers private key, and a log of the session would be able access all communication happening after the key was negotiated. The master secret is not negotiated again, unless the client ends the session and starts a new one, so this compromises the whole session.

Cooperation and Experience

The lab went mostly well. In part 1 the group we discovered that two of the group members already had experience with GPG. They could then give the last member a quick lesson in how it worked. As a result, that part went well.

Part 2 went well. This is partially because the group members are used to terminals. We also found the instructions from teaching assistants and lab instructions useful.

Part 3 went faster than expected. It was useful that one of the members had Wireshark installed and had experience using it. Further, the group members thought it was fun to see the data in the Wireshark log become decrypted in this part.

It was earlier mentioned that we found help from the teaching assistant very helpful. But unfortunately, one time we went to see them in Sahara. Since we did not see them and an other teaching assistant told us he did not think they were in the room, it ended with us sitting in the same room for over an hour. But we got help in the end!

References

[1] G. for Geeks. "Difference between rsa algorithm and dsa". (), [Online]. Available: https://www.geeksforgeeks.org/difference-between-rsa-algorithm-and-dsa/ (visited on 03/15/2022).

[2] H. C. Rudolph and N. Grundmann. "Weak cipher suite". (), [Online]. Available: https://ciphersuite.info/cs/TLS_DHE_RSA_WITH_AES_128_CBC_SHA/ (visited on 03/09/2023).

[3] S. Labs. "Ssl server rating guide". (), [Online]. Available: https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide (visited on 03/09/2023).