NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

TTM4135 APPLIED CRYPTOGRAPHY AND NETWORK SECURITY, SPRING 2021

Web Security Lab
Group 44

Simen Aarnseth Mathias Chunnoo Adrian Langseth Martin Stiles

July 14, 2021

1 Introduction

This is the web security lab report delivery of group 44 in the course Applied Cryptography and Network Security in the spring of 2021. The report consists of answers or solutions to the lab assignment questions. In addition, a section on the group cooperation and the groups subjective experience of the lab is included.

2 Questions

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: Public key encryption is much slower than symmetric key encryption. Hence a symmetric key is generated so that larger messages can be sent without the overhead of public key encryption.

Q2:

1. How many bytes does PGP use to store the private signing and public verification keys for each of the two signature types?
2. How many bytes does PGP use to store the signatures in each of the four cases (both long and short messages)?
3. How long, on average, does PGP uses for signature generation and verification in each of the four cases?
4. 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: 1. When exported, the exported RSA private key file takes 1325 bytes and the public key file takes 628 bytes, while the exported DSA key files takes 1065 bytes for the private key and 982 bytes for the public key.

2. PGP uses 310 bytes to store the private signing and public verification keys with RSA, while it uses 119 Bytes with DSA. PGP uses the the same amount of bytes for long messages as for short messages.

3. The average reported time usage for signature generation and verification is shown in Table 1 and Table 2, respectively. The times are the real run times averaged over 100 runs using, the commands A.1.1 and A.1.2.

4. The RSA private key file should contain n, p, q, e, d and $u = p^{-1} \bmod q$. These values should equate to about $650 - 750$ bytes. The rest of the file consists of about 350 bytes of the key name and a signature for the key itself, and the last couple hundred bytes are used for encryption of the secret part of the key. The file with the public key on the other hand contains about 250 bytes for the values n and e and about 350 bytes for the name of the key and a

	1MB	1KB
RSA	0.155	0.148
DSA	0.153	0.144

Table 1: Time usage for signature generation

	1MB	1KB
RSA	0.021	0.011
DSA	0.031	0.021

Table 2: Time usage for signature verification

signature for the key. For the DSA private key file, it should contain the values p, q, g, x and y which equates to about 832 bytes which takes mode space than the RSA key values. The reason the key file for DSA is smaller is that the signature for the key itself takes about half the space. For the public DSA key file, the kay values p, q, g, and y takes about 800 bytes. Factoring in a signature size of about 150 bytes, this file should be about 400 bytes larger than the RSA public key file, which is what we see.

When it comes to storing the signatures, the signature generated by RSA is the value σ with a size about 256 bytes. Wile the signature generated from DSA consist of the values r and s, both with a size of 32 bytes. Hence the DSA signature file should be about 192 bytes smalled than the RSA one, which is what we see.

As for the timing of generation and verification, the slowest part of the generation for both algorithms are their modular exponentiation, but DSA also has to compute a modular inverse, hence, as we see, both algorithms generate signatures almost at the same speed, but DSA is slightly slower. When it comes to verification, modular exponentiation is the slowest part of both algorithms. RSA uses the exponent e which is quite small, while DSA uses the two bigger exponents U_1 and U_2. Therefore DSA should be a lot slower, which is what we see.

> **Q3:** What assurances does someone receive who downloads your public key from the server where you have uploaded it?

A: To send in a public key to the server, you have to verify it from your own email. By doing this, it links your public key together with your account. This proves that it was the owner of the e-mail address that uploaded the public key to the server, and not anyone else. Further more other people can sign on top of your key verifying that it is indeed yours. This is known as the web of trust.

> **Q4:** Who typically signs a software release like Apache? What do you gain by verifying such a signature?

A: Typically, software published by companies such as Apache are signed by the company's own unique digital signature. This is called code signing[1]. By doing this they guarantee that the code/software has not been altered or corrupted since it was signed. This also ensures the receiver that the software is from Apache and not somebody pretending to be.

> **Q5:** Why did you obtain a certificate from Let's Encrypt instead of generating one yourself?

A: The point of a certificate is to verify the identity of the domain owner. For this to make sense both the client connecting to the domain and the owner of the domain has to trust whoever issued the certificate. Let's encrypt is a widely trusted certificate authority and hence clients are likely to trust their certificates. Had we on the other hand generated our own certificate clients would not be likely to trust it as anyone can generate a certificate for a domain.

> **Q6:** What steps does your browser take when verifying the authenticity of a web page served over `https`? Give a high-level answer.

A: 1. The web browser downloads the certificate of the web server. From the certificate it can extract the server's public key.

2. Most web browsers come pre-installed with the public keys of major CAs. The web browser use these public keys to verify that the web servers certificate was signed by one of the major CAs.

3. The certificate contains the domain name and/or IP address of the web server. Before connecting, the web browser checks with the CA that the address in the certificate has an open connection.

4.The web browser generates a shared symmetric key. This key will be used to encrypt all the HTTP traffic on the connection. The web browser will encrypt the shared symmetric key with the public key belonging to the web server. It will then send this as a message to the server. Since the server is the only one with the private key, nobody else will be able to view the symmetric key.

> **Q7:** 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 first part determines that the protocol is Transport Layer Security. The second part describes the key exchange as being Diffie-Hellman Ephemeral. The third part denotes the Authentication mechanism and shows that RSA is used for the handshake. WITH does not indicate much, but tells that the RSA is done WITH the following encryption. The fifth through seventh part correspond to the encryption with the fifth denoting the session cipher as AES with a with a key size of 128 (as indicated in part 6) in a Cipher Block Chaining mode (as indicated in part 7). The last part denotes the hash function as SHA 1.

> **Q8:** 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 SHA 1 was proven insecure in 2017 and should therefore not be used. The encryption specified by this string is considered weak. In addition to this, the CBC is vulnerable to BEAST attacks and could/should therefore be changed to GCM.

> **Q9:** 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: When a CA issues a certificate, it will record this on a log which is publicly available on a server. This server contains a log of all certificates issued. The certificate has several different fields containing information about it. These are Log ID, Issue Date, Hash Algorithm, Signature Algorithm and Signature Data. It is possible to delay the time before an SCT is logged by specifying the Maximum Merge Delay(MMD). This delay can at most be 24 hours.

> **Q10:** 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: A server cannot support neither TLS 1.1 nor 1.0. If they do, the grade is capped to B. Therefore, to get an A rating, a server must support TLS $\geq$ 1.2, and nothing below. Furthermore, AEAD and Forward secrecy must be supported to attain a A rating. Implementing these restrictions will exclude a lot of older devices from using the servers. Therefore, the popular web servers do not restrict their users to TLS $\geq$ 1.2, in an attempt to not exclude potential customers.

> **Q11:** 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:

The value of the client and server nonces can be seen in appendix A.2 and A.3 respectively. The nonces are both 32 bytes and consists of a 4 byte unix timestamp and a random 28 byte number. This is what we would expect from the TLS 1.2 specification. The value of the encrypted pre-master secret can be seen in appendix A.4 and it has a length of 256 bytes. The public key sent by the server has a length of 257 bytes, which indeed would create a 256 byte encrypted pre-master secret when using RSA.

> **Q12:** 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: As discussed on blackboard forum, we have only figured out the value of the master secret.
The value of the master secret can be seen as a 96 digit hexadecimal value in appendix A.5. This hexadecimal value represents an 48 byte value which is what you always would expect from the TLS 1.2 specification. We can see in the *SLLKEYLOG* file that the same master secret have been used in all transactions with the server. This means the session does not have forward secrecy. This is because if the master secret of one of the later messages where to be found out, one could use it to decrypt massages sent sent earlier on.

3 Cooperation and Experience

3.1 Group responsibilities

Although the lab is a group assignment and all group members participated in each aspect, the group divided responsibilities amongst themselves to increase efficiency and avoid the situation where everyone spectated a single member doing the implementation. Martin and Mathias were identified to have the highest proficiency in practical cryptography and therefore took the lead on the tasks themselves. Simen and Adrian concluded that the lab report should be completed in parallel to the lab tasks and took it upon themselves to document and describe the task solutions as they were generated. As stated previously, but repeated for clarity: All members took part in both parts (and all subparts) of the assignment, but certain members took leadership and the accompanying responsibility of the described sections.

3.2 COVID and Digital

A cryptography lab as this one does not require lab spots an availability of larger equipment, such as lab work in the traditional sciences might. Despite this, the possibility of meeting and completing the work in close cooperation with each other is necessary for an efficient lab environment. This was not always possible, due to the restrictions of COVID. Furthermore, NTNU has slashed the capacities of group rooms, and removed the students' access, we were not always able to find spaces to cooperate. Fortunately, the group were not totally hampered by the restrictions, as the group managed to cooperate through the use of Discord. One member shared his screen for the rest of the group and discussion was possible on the actions continuously. Although digital will never be a fully qualified alternative to a physical lab, employment of solid tools has greatly mitigated the negative effect.

3.3 Lab sentiment

The group found the lab to be quite interesting, but also at times very challenging. Throughout the lab, we found ourselves to be increasingly dependent upon the proficiency of a single member. This understandably escalated throughout the lab as the tasks became harder. Such a situation was addressed quite late, and with the little time available for group work, the group did not make any large-scale attempt to remedy this, such as redoing all tasks. Rather the group discussed the tasks and the actions taken to complete them in such a way for all members to gain a good understanding of the tasks and solutions. The little time available was not a fault of the lab, as three weeks proved to be an appropriate amount for such an assignment, but rather a product of finding time where four members of four different specializations and with separate projects running parallel to the lab, where all were available to work together. Despite this, the lab was very useful for the academic development in the course, and proved to be an effective practical experience of the second part of the lab.

References

[1] Microsoft. Introduction to code signing.

Appendix A

A.1 Code for task 2.3

A.1.1 Timing signature generation

```
time for i in {1..100}; do gpg -u key_id --output /dev/null --detach-sign
message.txt; done
```

A.1.2 Timing signature verification

```
time for i in 1..100; do gpg --verify signature.sig message.txt; done
```

A.2 Client nonce for task 15

```
57fb36c569ad885cff580f0eb6e705073f91743cb9d8e2ef339c21ee9a900534
```

A.3 Server nonce for task 15

```
2ae5c4c12acc3ab029f73f2f3a0bd46b0039e774d4e6b1a8444f574e47524401
```

A.4 Encrypted Pre-master secret for task 15

```
90702969aa450d7bd0ac6bc354172d4810aacf3e5875582652360be68274abc
1a77959b54b3bca9e6daad88798807ab2c6dc9aaf8a06e4692b790b6b7772db
83a36ded6dd2fff5c5239943c9a9aea453f5625c3ce54accce838d48794c25e
2a88dce8541d8443812a872fe817b24e4e5aaaaba2f2902c33fa49688c97c69
42a53250c63bd49abf2d4e752e4fcba3386b5aa1364129ee90399c10392c27d
28e5c4c2277e496de56a3d7a8b94af8eb3983e3a55f497cd4ee7e0d0a52fd25
ce65d3557009a14496bceaf1c2d2abfe760da9173fae81b2bbb4d38033a5ddf
688ac31dec210e70d797159de112c91ada5b565e2e037a41bc8f77c8a47062e
5dd02218
```

A.5 Pre-master secret (task 16)

```
3c1ed14d3ad7e35b5e11a7769b6bb1a2d2c45fdb2115658b8a0376f20004f39
dd13742358c2b894238c568b43d080196
```