Cryptography - Lab 7 - MAC and SSL/TLS Handshake

Javier Antxon Garrues Apecechea

Student ID: 24647808

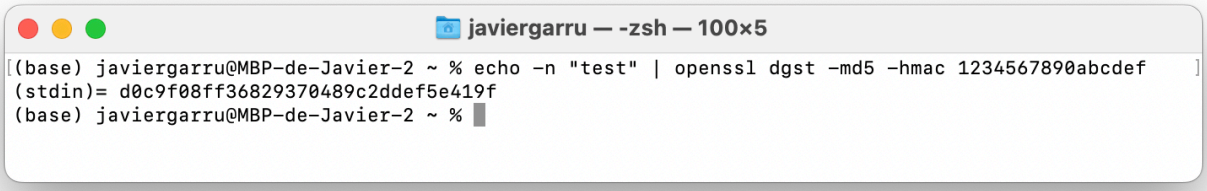
In this lab we will get familiar with one-way hash functions.

Part One: Message Authentication Code (MAC)

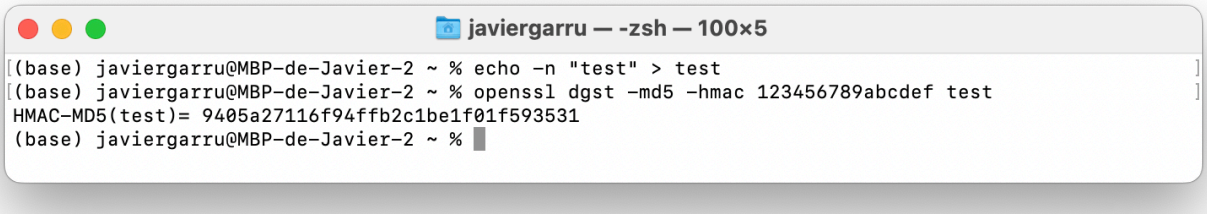
2. GENERATING HMAC USING KEYS

2.1 Practice HMAC:

Without using a file:



Finding hmac of a file:



Question: Compared to hash functions, why MAC can resist man-in-the middle attack?

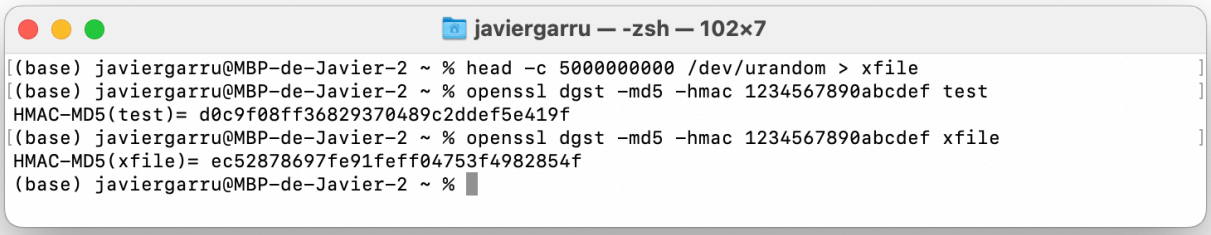
Because it uses a key that is shared by sender and receiver. The adversaty doesn't have access to that key. On the other hand, using Hash the adversary can position himself in the middle and generate the same hash as any of the participants of the communication.

Question: Suppose we have a MAC function with c binary digits MAC code and k binary digits key, what are the levels of effort of the brute-force attacks on the MAC code and the key? What is the best strategy of attackers?

The levels of effort of the brute-force attack on the MAC code is O(c) and the levels of effort of the brute force attack on the key is O(k). The best, and theoretically only, strategy attackers can follow is to try all the possible combinations of key (2^k) and possible plain texts and generate the HMAC of those combinations until finding one that matches the HMAC. In conclusion, the best strategy is brute-force attack. Although the size of c can be smaller than k, the fact that the Hash functions aren't invertible makes impossible going from the 2^c-size space to the plain-text one. Some other strategies of attacking HMAC are here mentioned (but they are mostly based on implementation design mistakes):https://crypto.stackexchange.com/questions/95568/is-there-any-good-attack-model-of-hmac.

2.2 Practice HMAC on files with different sizes

Create a large file and find the hmac:

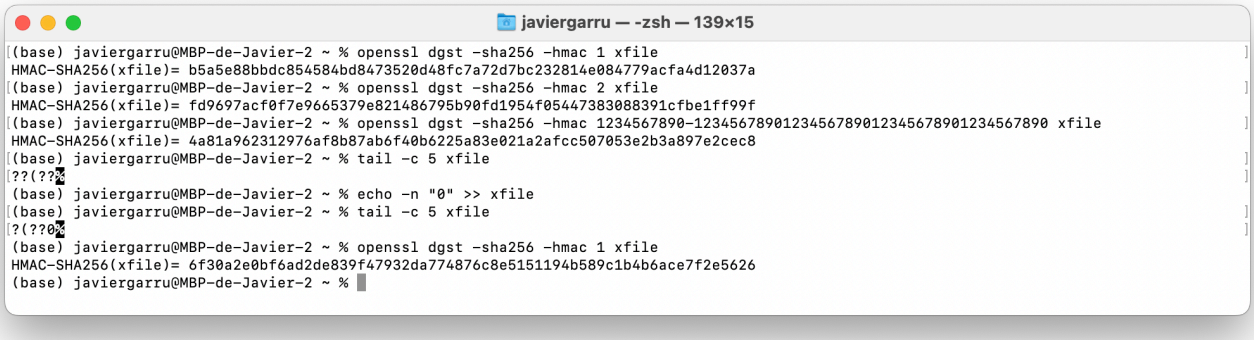


Question: Does the file size affect the length of HMAC values? Why?

It doesn't affect the length as the HMAC is based on a hash function, in this case md5. As we saw in the previous lab, hash functions map inputs to a fixed-length output space in this case 128 bits. Something that it is important to notice is, that althought the size of the output is the same, the time it has taken to compute the second HMAC has been considerably bigger. This is due to the extra computational effort requited.

2.3 Practice HMAC with different keys and file contents

Now we are going to try different keys (1,2 and a really long key). We are also going to add a 0 at the end of the extra large file and see how the HMAC changes after encrypting with the same key:



As we can see, the different keys significantly change the final HMAC of the file as the Hash functions that are being used have the ability of map close input values to really different output values. We can also see how this occurs when the small change occurs in the file instead of in the key.

Question: Will different keys generate different HMAC values on the same file? Is there any limitation of the length of the key, why?

What can you summarise from this task? Which security property does this feature ensure?

Yes, different keys will generate different HMAC values. We have seen an example of that in the previous screenshot and the reason for this is that they are used to modified the input of the hash functions. There are limiations on the size of the keys as it cannot be longer that the block size of the Hash function in which the HMAC algorithm is based. If the key is longer then most algotithms will automatically hash it, then padded until reaching the block size and use the obtained key. An example is seen here https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.hmacsha256.-ctor?view=net-7.0#:~:text=The%20secret%20key%20for%20HMAC,derive%20a%2032%2Dbyte%20key. The reason for this is that the padded key is going to be XORed with the ipad and , in another step, with the opad and they need to build a block of the input to the hash.

I can summarise that the small changes in the input generate big inputs in the output which ensures that if the message is tampered during the transaction, computing the HMAC with the correspondent key in the other size of the communication will detect those changes. Integrity will therefore be satisfied. The fact that a small change in the key generates a different HMAC also ensures confidentiality as only the user with the correct key can read the message.

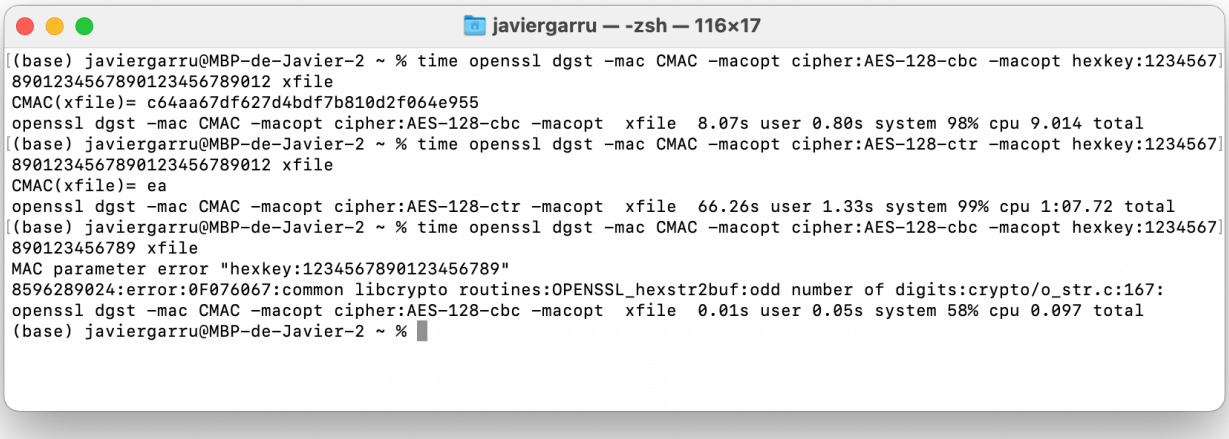
2.4 Try HMAC with different hash algorithms

Question: Why HMAC is compatible with different hash algorithms?

Because the structure of HMAC is indepentend to the underlying Hash algorithm being used. The only effect this algorithm will have is the block size and output size generated but all these hash functions have the necessary properties to work correctly in the scheme.

2.5 Try CMAC

We will try it with AES-128-cbc and AES-128-ctr:



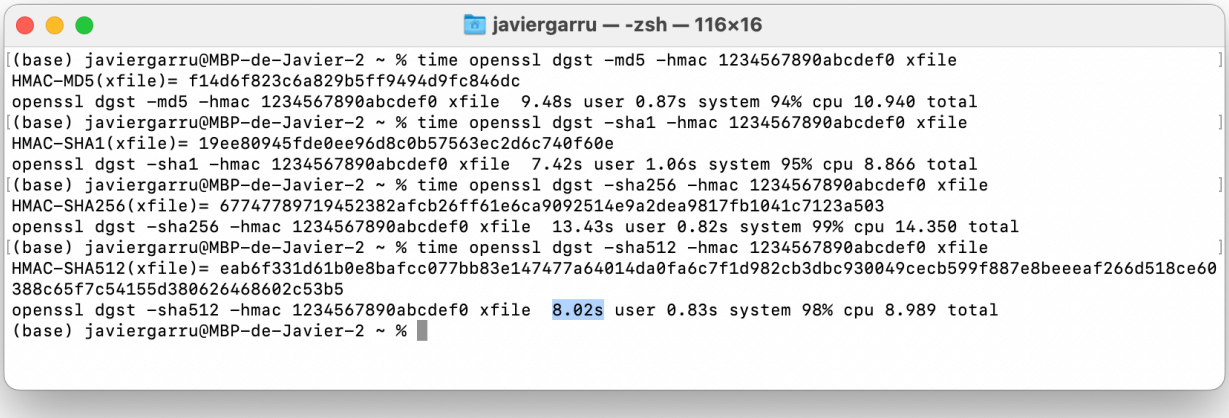
Question: Read the slide of CMAC (page 16) and explain why the last two commands are failed

They failed because the leght of the key used was not the correct one. In the case of AES-128-cbc the key must be 128 bit (32 bytes) and the one used in the last execution was less than that. In the case of AES-128-ctr is because the mode uses counters and it is not chained which makes the final output to be different blocks from which we would have to define a specific function to select the lenght of bits we want to extract. When using CBC is clear that we take the s leftmost bits of the final encrypted value.

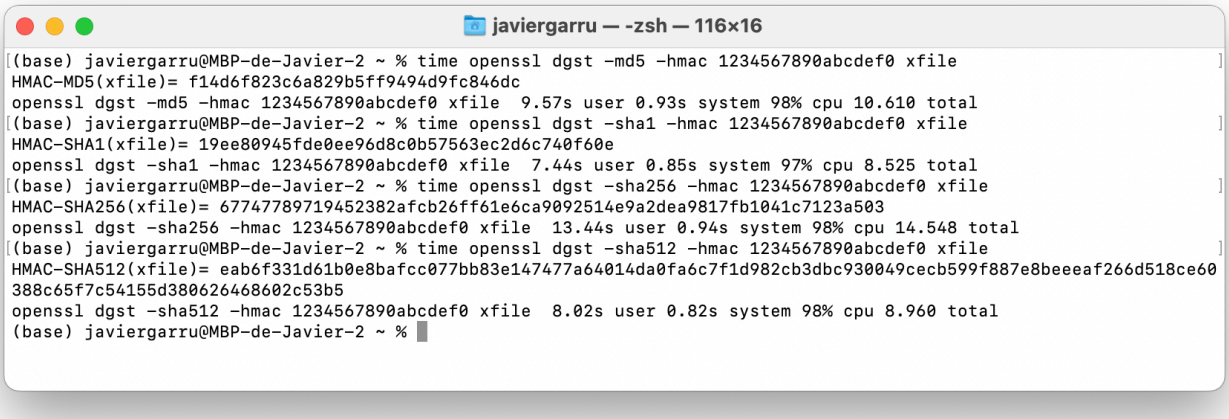
3 EFFICIENCY TEST

3.1 Compare the time efficiency of HMAC functions with different hash functions

First execution:



Second execution:



Third execution:

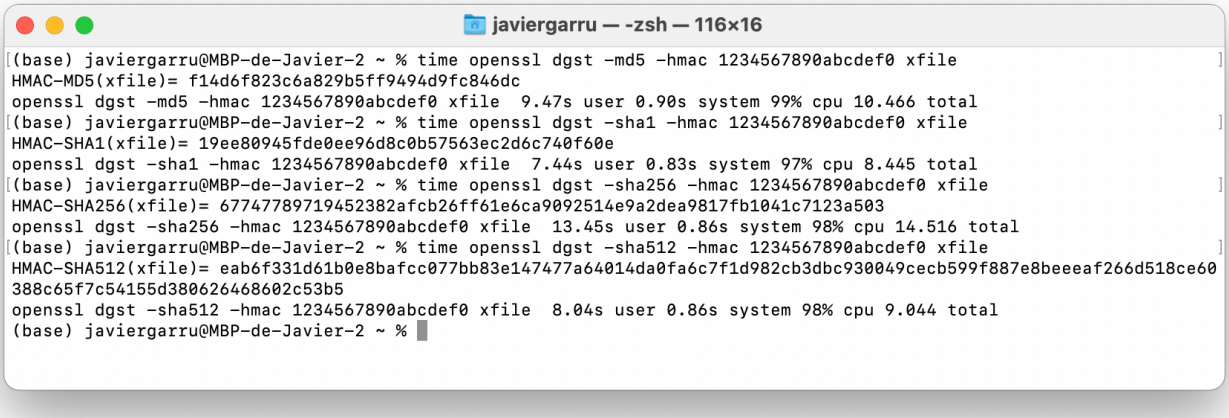


Table:

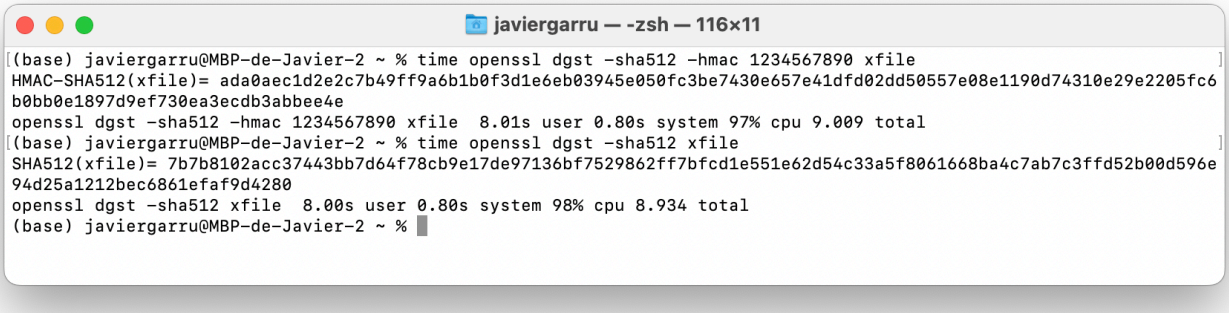
HASH function	First run	Second run	Third run	Average
md5	9.48s	9.57s	9.47s	9.51s
sha1	7.42s	7.44s	7.44s	7.4333s
sha256	13.43s	13.44s	13.45s	13.44s
sha512	8.02s	8.02s	8.04s	8.0267s

Question: How long do these HMAC functions run? Which one is the fastest?

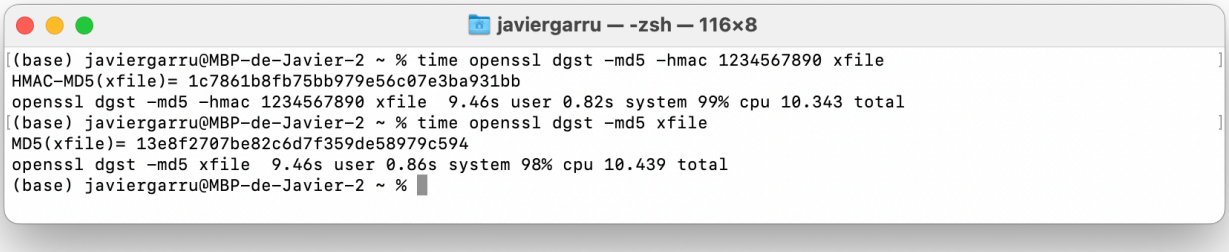
The table shows the answer to both questions. The fasters is the HMAC using hash function sha1. Note that the file had more than 500MB in my case.

3.2 Compare the time efficiency of HMAC functions and corresponding hash functions

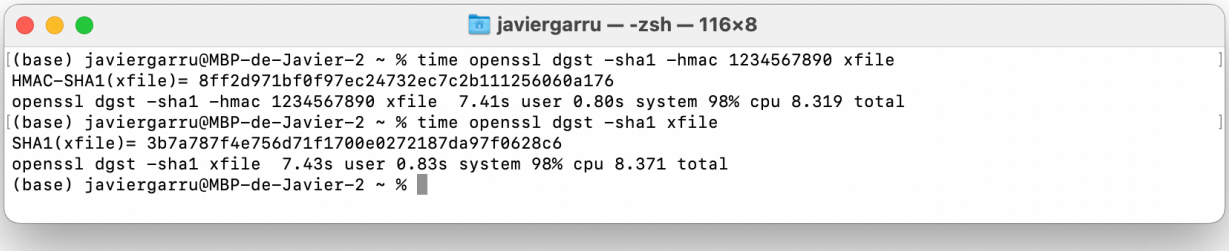
Comparing sha512:



Comparing md5:



Comparing sha1:



Comparing sha256:

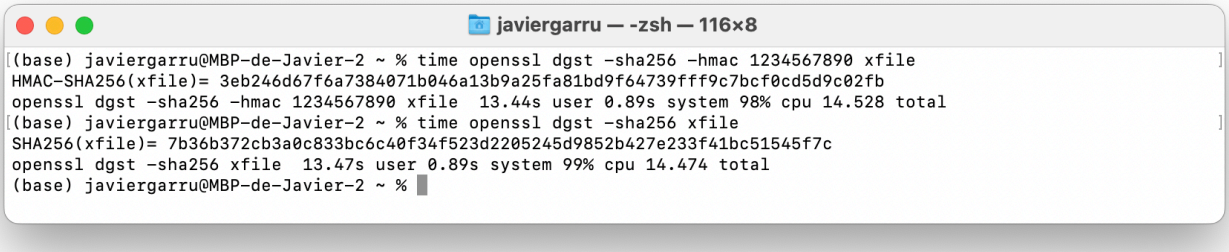


Table:

HASH function	Hash only	HMAC
md5	9.46s	9.46s
sha1	7.43s	7.41s
sha256	13.44s	13.47s
sha512	8.00s	8.01s

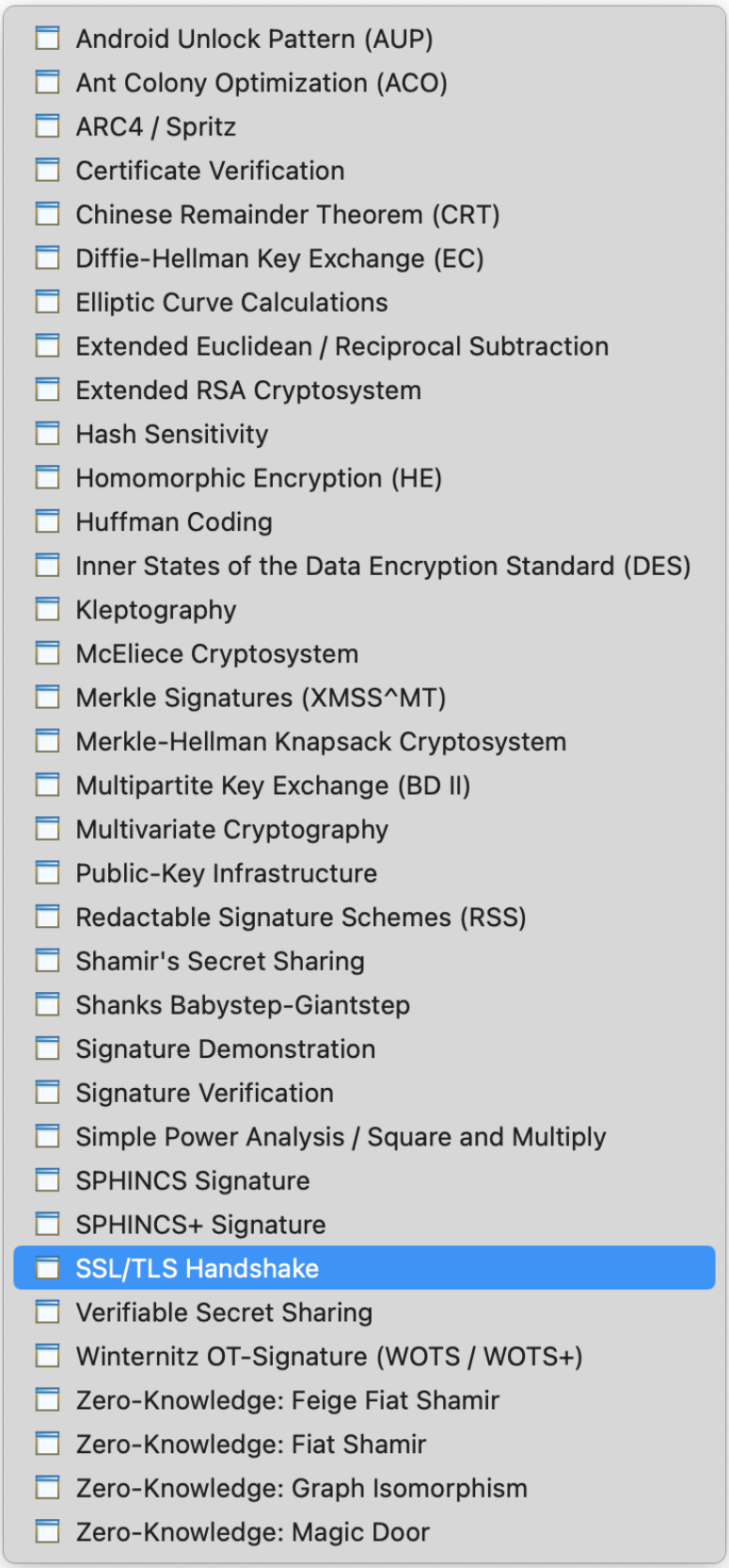
####

Question: Do the running time of HMAC function and corresponding hash function differ much, why?

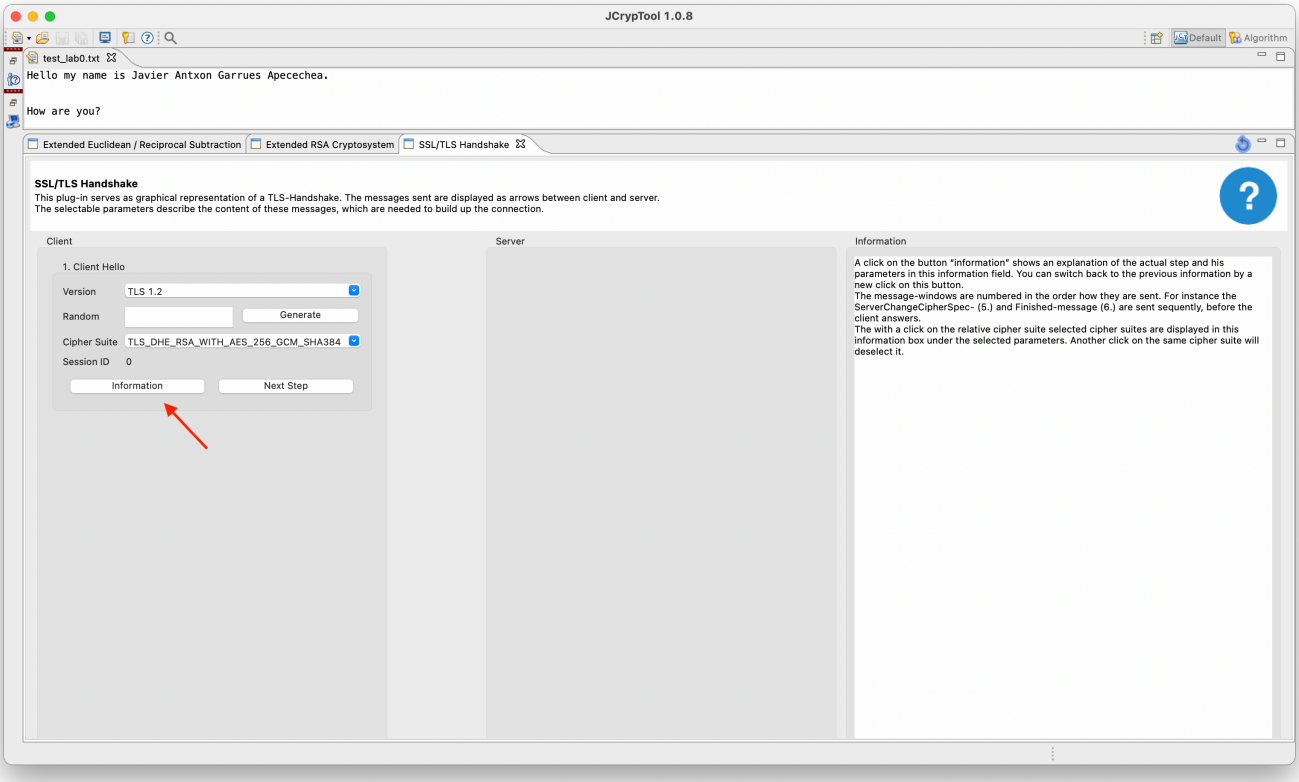
They don't differ much. The reason for this is that the HMAC is implemented in an efficient way such that the hash function is only called once and the rest of operations used in the process are computationally light which means that the time it takes to do them is negligible. Openssl implements different specific modifications that optimize the operations so the time is reduced even more.

Part Two: SSL/TLS Handshake

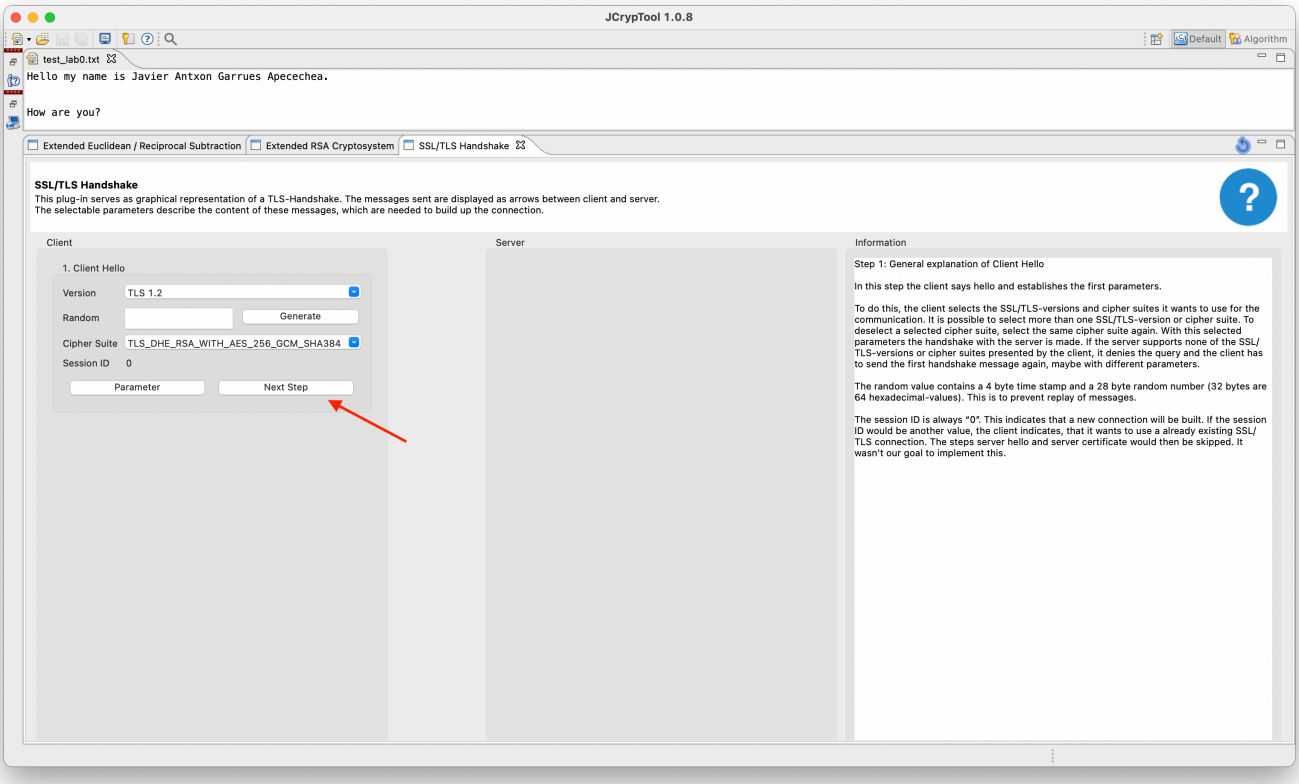
We will use the tool call JCrypTool for the following part of the lab. We are going to select the SSL/TLS handshake option:



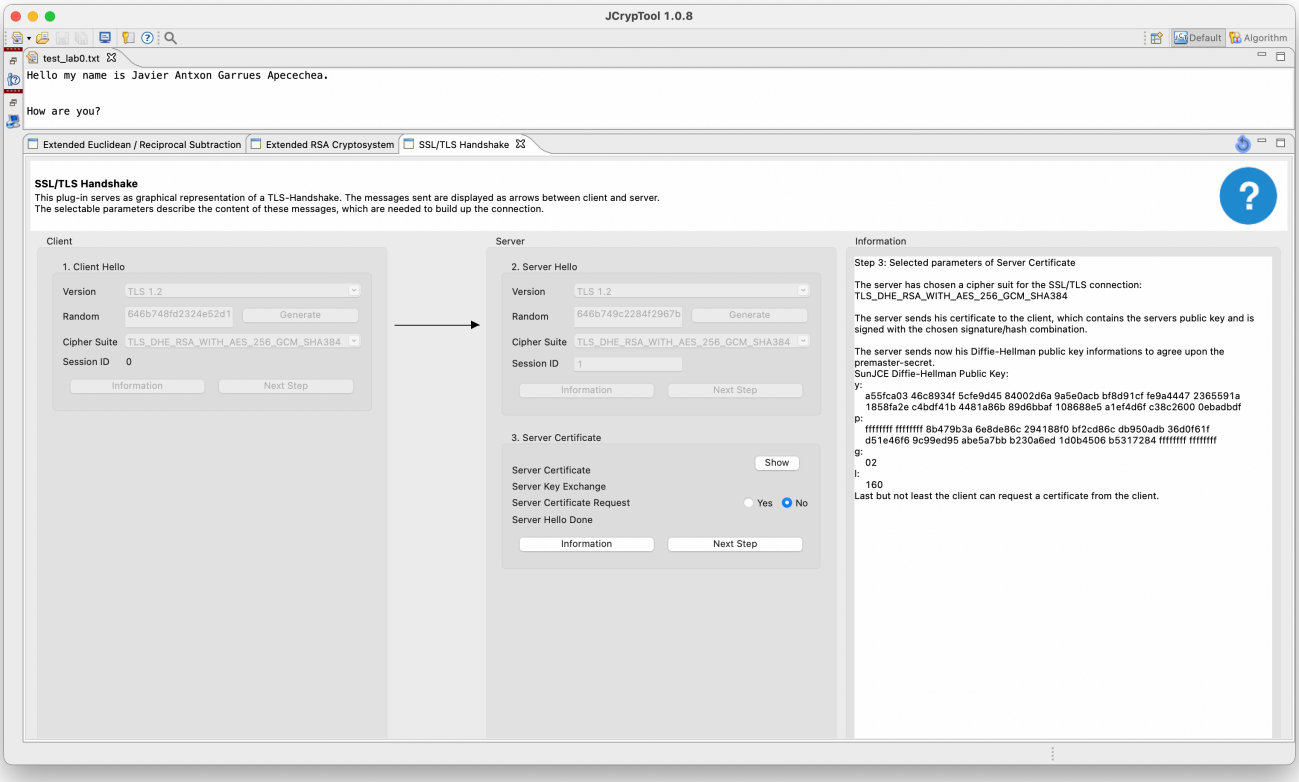
Once there we will see the following window:



We can then click on *Information* and read extra information about the step we are in:



That way we keep going step by step and looking at the information. In the following window we can also click on the *Show* option to see the certificate:



Here we can see what the certificate contains:

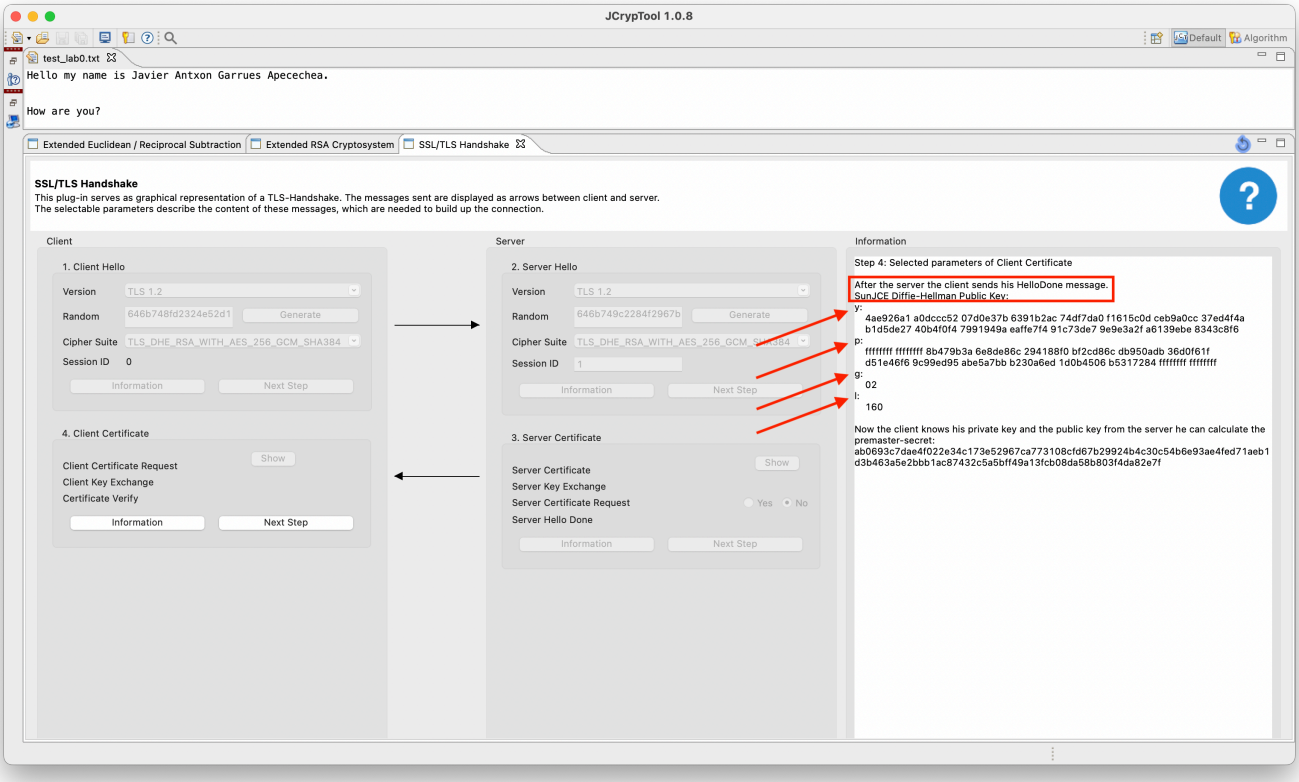
Certificate:
Data:
Version: 3
Serial Number: 1684763805021
Signature Algorithm:
SHA384WITHRSA
Issuer: CN=Test Server
Certificate
Validity
Not Befor: Mon May 22 23:56:45
AEST 2023
Not After: Thu May 23 11:59:59
AEST 2024
Subject: CN=Test Server
Certificate
Subject Public Key Info:
Public Key Algorithm: DH

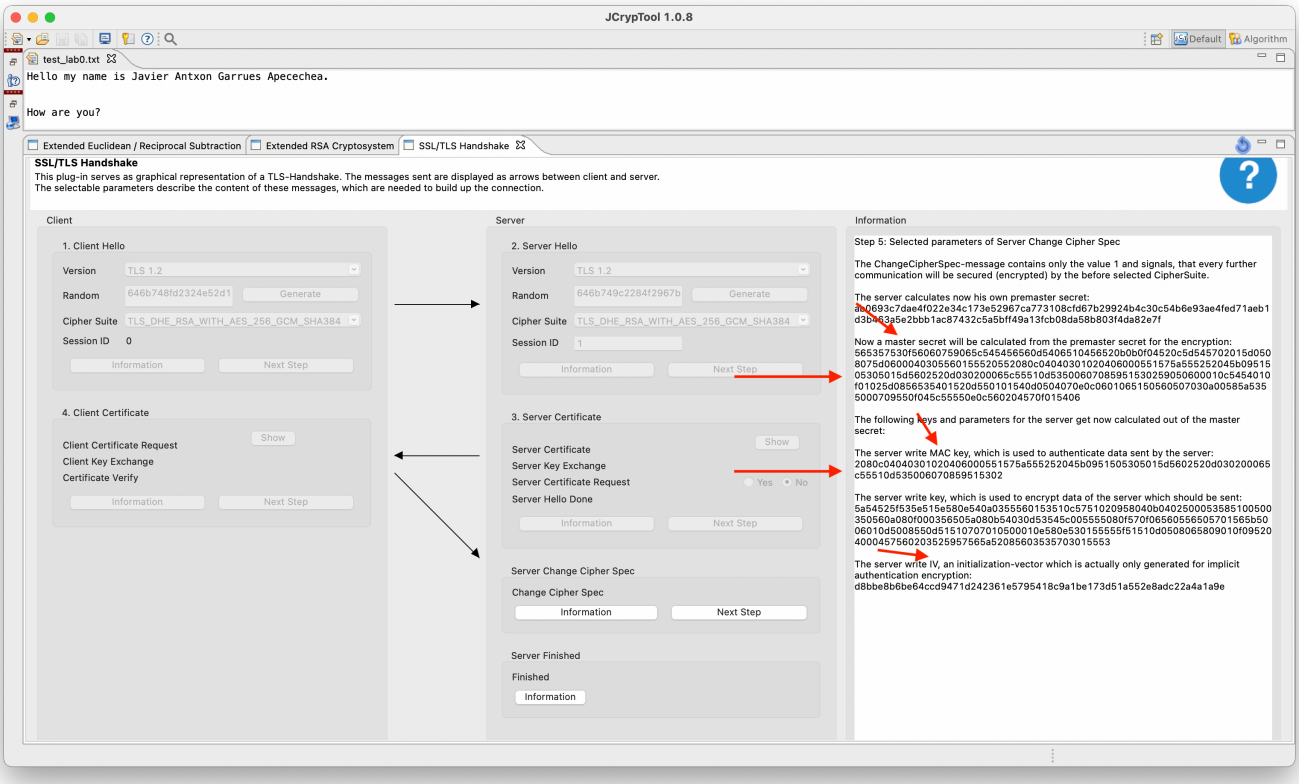
OK

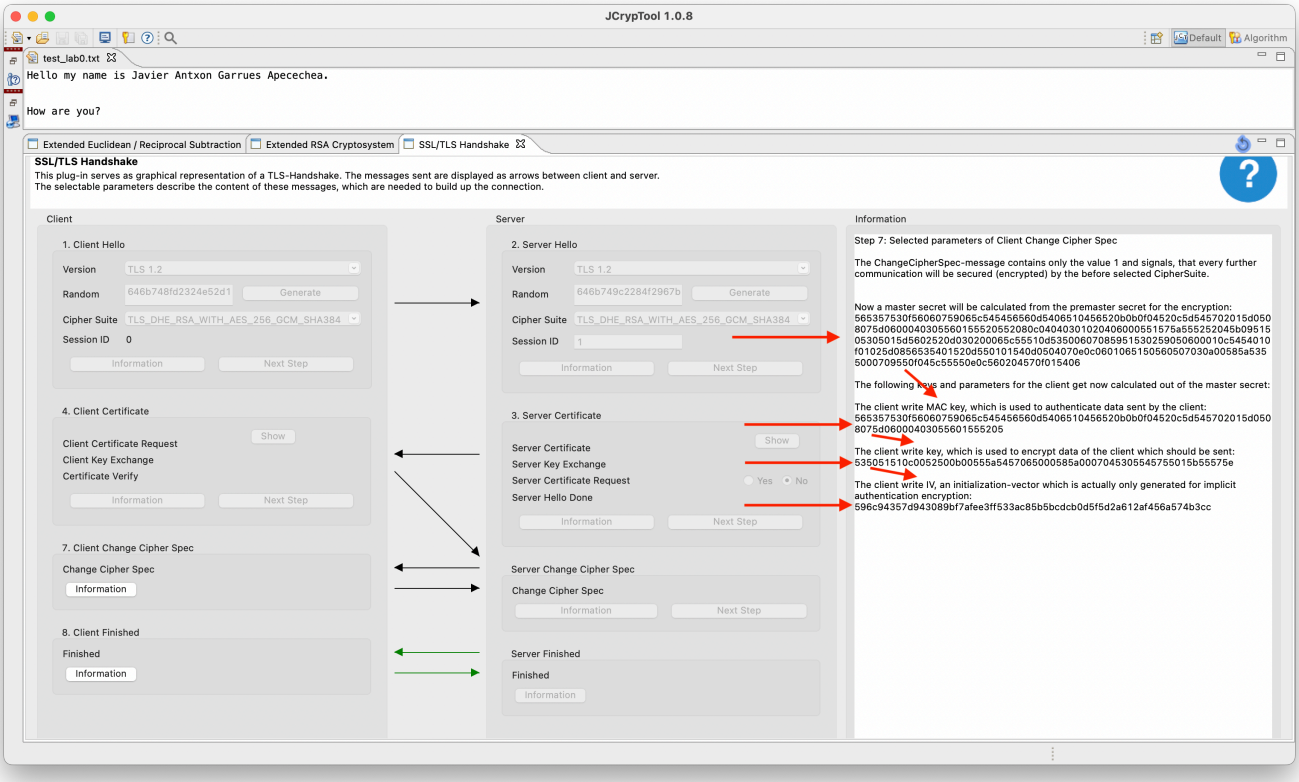
Public Key Algorithm: DH
a5:5f:ca:03:46:c8:93:4f:5c:fe:9d
:45:84:00:2d:
6a:9a:5e:0a:cb:bf:8d:91:cf:fe:9a:
44:47:23:65:
59:1a:18:58:fa:2e:c4:bd:f4:1b:4
4:81:a8:6b:89:
d6:bb:af:10:86:88:e5:a1:ef:4d:6f
:c3:8c:26:00:
0e:ba:db:df
Signature Algorithmen:
SHA384WITHRSA
b1:d3:85:c2:4d:6b:0f:27:04:fe:7
9:dd:d0:ff:b5:
8c:c7:d6:18:cc:4c:9e:2b:44:ef:a
5:37:7f:eb:88:
be:8e:9d:4d:82:c3:c4:5a:14:fb:1

OK

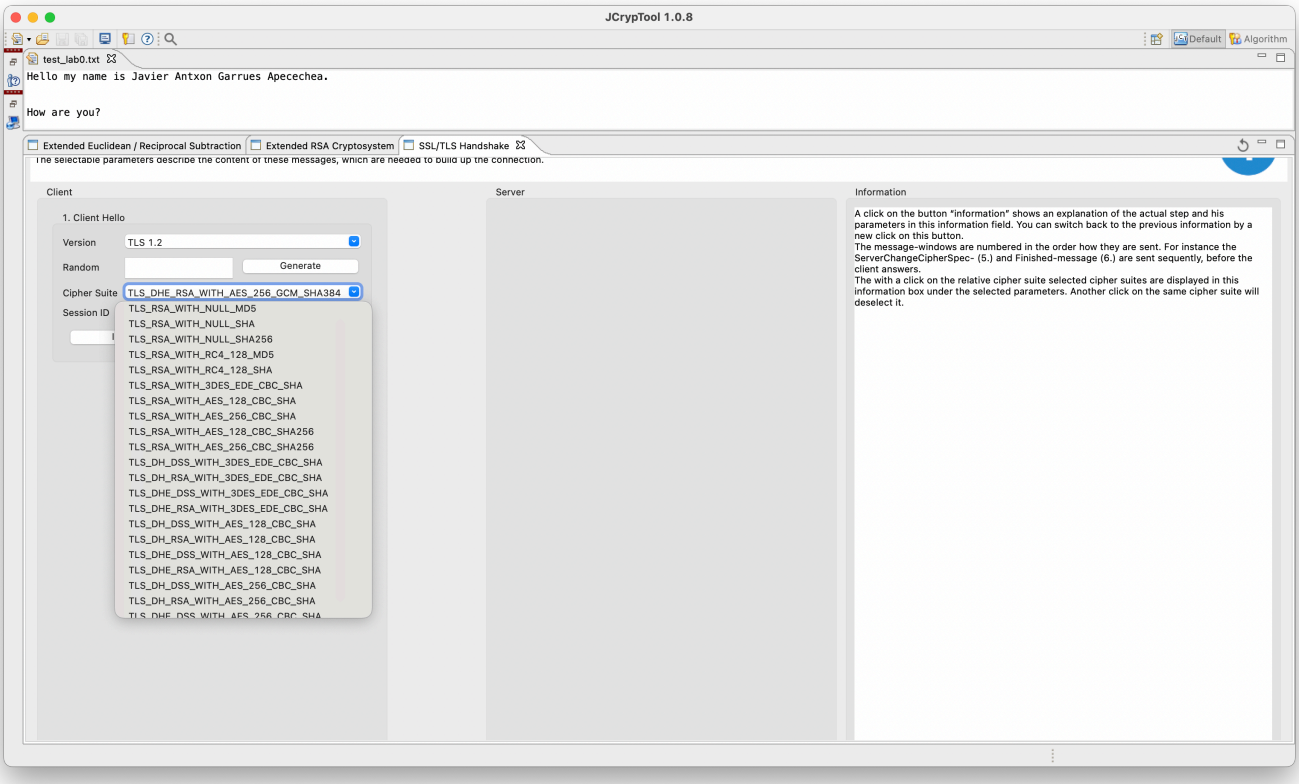
The following screenshots show the final steps:


JCrypTool 1.0.8
test_lab0.txt
Hello my name is Javier Antxon Garrues Apecechea.
How are you?
Extended Euclidean / Reciprocal Subtraction
Extended RSA Cryptosystem
SSL/TLS Handshake
SSL/TLS Handshake
This plug-in serves as graphical representation of a TLS-Handshake. The messages sent are displayed as arrows between client and server.
The selectable parameters describe the content of these messages, which are needed to build up the connection.
Client
1. Client Hello
Version
TLS 1.2
Random
646b748fd2324e52d1
Generate
Cipher Suite
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
Session ID
0
Information
Next Step
4. Client Certificate
Client Certificate Request
Client Key Exchange
Certificate Verify
Show
Information
Next Step
Server
2. Server Hello
Version
TLS 1.2
Random
646b749c2284f2967b
Generate
Cipher Suite
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
Session ID
1
Information
Next Step
3. Server Certificate
Server Certificate
Server Key Exchange
Server Certificate Request
Server Hello Done
Show
Yes
No
Information
Next Step
Information
Step 4: Selected parameters of Client Certificate
After the server the client sends his HelloDone message.
SunJCE Diffie-Hellman Public Key:
y:
4ae926a1 a0dccc52 07d0e37b 6391b2ac 74df7da0 f1615c0d ceb9a0cc 37ed4f4a
b1d5de27 40b4f0f4 7991949a eaffe7f4 91c73de7 9e9e3a2f a6139ebe 8343c8f6
p:
ffffffff ffffffff 8b479b3a 6e8de86c 294188f0 bf2cd86c db950adb 36d0f61f
d51e46f6 9c99ed95 abe5a7bb b230a6ed 1d0b4506 b5317284 ffffffff ffffffff
g:
02
l:
160
Now the client knows his private key and the public key from the server he can calculate the premaster-secret:
ab0693c7dae4f022e34c173e52967ca773108cfd67b29924b4c30c54b6e93ae4fed71aeb1d3b463a5e2bbb1ac87432c5a5bff49a13fcb08da58b803f4da82e7f


JCrypTool 1.0.8
test_lab0.txt
Hello my name is Javier Antxon Garrues Apecechea.
How are you?
Extended Euclidean / Reciprocal Subtraction
Extended RSA Cryptosystem
SSL/TLS Handshake
SSL/TLS Handshake
This plug-in serves as graphical representation of a TLS-Handshake. The messages sent are displayed as arrows between client and server.
The selectable parameters describe the content of these messages, which are needed to build up the connection.
Client
1. Client Hello
Version
TLS 1.2
Random
646b748fd2324e52d1
Generate
Cipher Suite
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
Session ID
0
Information
Next Step
4. Client Certificate
Client Certificate Request
Client Key Exchange
Certificate Verify
Show
Information
Next Step
Server
2. Server Hello
Version
TLS 1.2
Random
646b749c2284f2967b
Generate
Cipher Suite
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
Session ID
1
Information
Next Step
3. Server Certificate
Server Certificate
Server Key Exchange
Server Certificate Request
Server Hello Done
Show
Yes
No
Information
Next Step
Server Change Cipher Spec
Change Cipher Spec
Information
Next Step
Server Finished
Finished
Information
Information
Step 5: Selected parameters of Server Change Cipher Spec
The ChangeCipherSpec-message contains only the value 1 and signals, that every further communication will be secured (encrypted) by the before selected CipherSuite.
The server calculates now his own premaster secret:
ab0693c7dae4f022e34c173e52967ca773108cfd67b29924b4c30c54b6e93ae4fed71aeb1d3b463a5e2bbb1ac87432c5a5bff49a13fcb08da58b803f4da82e7f
Now a master secret will be calculated from the premaster secret for the encryption:
565357530f56060759065c545456560d5406510456520b0b0f04520c5d545702015d0508075d060004030556015552052052080c0404030102040600055157 5a555252045b0951505305015d5602520d030200065c55510d535006070859515302590506000 10c5454010f01025d0856535401520d550101540d0504070e0c0601065150560507030a00585a5355000709550f045c55550e0c560204570f015406
The following keys and parameters for the server get now calculated out of the master secret:
The server write MAC key, which is used to authenticate data sent by the server:
2080c0404030102040600055157 5a555252045b0951505305015d5602520d030200065c55510d535006070859515302
The server write key, which is used to encrypt data of the server which should be sent:
5a54525f535e515e580e540a0355560153510c57510209580 40b040250005358510050035056 0a080f000356505a080b54030d53545c005555080f570f0656055650570156 5b5006010d5008550d5151070701050001 0e580e530155555f51510d0508065809010f0952040004575020352595756 5a5208560353570301555 3
The server write IV, an initialization-vector which is actually only generated for implicit authentication encryption:
d8bbe8b6be64ccd9471d242361e5795418c9a1be173d51a552e8adc22a4a1a9e



Reseting the process we can come back and select the desired Cipher suite:



Question: Which cipher suite do you prefer? Explain the cipher suite and give your reason choosing the cipher suite.

I would prefer the TLS_DH_RSA_WITH_AES_128_GCM_SHA256. This cipher suite uses Diffie Hellmand key exchange which is less secure that the ECDH but computationally lighter. This would enable to intialize many connections faster from machines with low computational capabilities. RSA will be chosen due to its simplicity and strength for the authentication of the server and generating the public key. I would choose AES instead of DES as I can choose the 128-bit version which is more poweful than the standard DES. This would be used for encryption.The Galois/ Counter Mode uses counters and can be easily paralelized which is something good when doing several operations simultaneously. At last, the underlying hash function is SHA256 instead of SHA384. GCM and SHA will be used to obtain the message tag/digest and ensure data integrity. The choice of SHA256 is based on the fact that is more widely supported (for now). This would ensure that most times the guessed cipher suite by the client is supported by the server.

Part Three: Summary

In summary, in this lab we have learnt the functioning of HMAC functions and the influence the use of different hash functions have on them. Furthermore we have studied their performance in terms of time and compare the results. At last, we have studied how TLS/SSL handshake protocol works, the different cipher suites available and their meanings.