5^{th} Assignment Information Security

Francesco Segala 3521885 Manos Gionanidis 3542068

October 16, 2017

Excercise 25

Break a 64-bit cryptographic key:

As an average a 64-bit cryptographic key is broken after 2^{63} attemps, so this is tha average scenario but as the worst scenario we consider that a cryptographic key is broken is 2^{64} attemps. This means we check all the possible combinations of 64-bit cryptographic keys.

1) Worst case scenario when a key is broken in 2^{64} attempts:

I need to break this key slightly more than three weeks, so i take as an upper bound exactly three weeks. 3 weeks = 1814400 sec $2^x = 1814400 => \log_2 2^x = \log_2 1814400 => x = \log_2 1814400 = 20,7910611$. So x = 21, result: 3 weeks = 2^{21} seconds.

Worst case scenario : attempts / (determine how many calculations i want to break the key) = (the time i want to break the key) $(2^{64})/x = (2^{21}) => x = (2^{43})$ calculations/per second I need to break the cryptographic key in 3 weeks.

A) When you can rent servers for the required amount of time

I assume that we need 1 cycle for one calculation , so one sarver can provide us (2^{38}) cycles/sec so calculations pre second. Now I am going to find how many servers I need.

$(2^{38}) * x = (2^{43}) => x = (2^5)$ server I am going to need . I know that one server costs 250 euros per year, but I only need the servers for 3 weeks . So server's cost for 3 weeks is $x = 250 * (21/365) = 14,38$ euros a server Amount $= 14,38 * (2^5) = 460,16$ euros

B) When you must rent servers using contracts that are valid for a year.

One year = 31556952 seconds = 2^{25} seconds $(2^{64})/x = (2^{25}) => x = 2^{39}$ calculations pre second in order to break the key in one year.

How many servers i want: $2^{38} * x = 2^{39} => x = 2^1 = 2$ servers Amount= $2 * 250 = 500$ euros

2) Average case when the key is broken in 2^{63} attempts:

I need to break this key slightly more than three weeks, so i take as an upper bound exactly three weeks. 3 weeks = 2^{21} (same procedure) seconds.

$(2^{63})/x = 2^{21} => x = 2^{42}$ calculations/per second I need to break the cryptographic key in 3 weeks.

A) When you can rent servers for the required amount of time

I assume that we need 1 cycle for one calculation , so one sarver can provide us 2^{38} cycles/sec so calculations pre second. Now I am going to find how many servers I need.

$(2^{38}) * x = 2^{42} => x = 2^4$ server I am going to need . Amount = $14,38 * (2^4) = 230,08$ euros

B) When you must rent servers using contracts that are valid for a year.

One year = 31556952 seconds = 2^{25} seconds $(2^{63})/x = (2^{25}) => x = (2^{38})$ calculations pre second in order to break the key in one year.

How many servers i want: $(2^{38}) * x = (2^{38}) => x = 2^0 = 1$ servers Amount= $1 * 250 = 250$ euros

Excercise 26

The added value of having a secure shared key K between two endpoints in DH key exchange procedure is that now the attacker cannot perform a MiTM attack against one (or both) of them. Infact if Trudy (the attacker) is able to intercept a message e.g. $g^a modp$ in the comunication she/he can then act as one of the two legitimate point of the comunication. Then he/she send this message to Bob pretending to be Alice , Bob reply with his own part of the secret $g^b modp$, now Trudy can compute the shared secret. If instead of the previous scenario Bob and Alice encrypt their messages with a secure shared key Trudy is no longer able to perform a reply attack against them because he/she cannot come up with the shared secret because of the encryption.

Excercise 27

Is always bad to use the same keypairs for do encryption/decryption and for signing documents. This is true for many reasons, first of all if Trudy in some ways retrieve an encrypted message $P_a\{M\}$ sent by Bob to Alice she then can pretend to be Bob (typical Alice-Bob comunication scenario) and send to Alice a request for signing a document C that is indeed the previous encrypted message. Now Alice dutyfully sign C with her private key and Trudy came up with the plaintext M. The same result (good one for Trudy) can be reached both via the public key encryption of a nonce either via private signing of a plain nonce because of from Alice's point of view the procedure is the same. However this is a bad idea also for another reason : Key management .

Signature keys and encryption keys have different lifetime in terms of back-ups, access control, repudiation, etc. The fallback for a signature key in

case of a catastrophic event is to destroy it to avoid future forgeries, so a signature key does not need to be backed up extensively , and in case of leak pre-existing documents still remain valid and signed by the trustworthy person and just the new ones has to be withdrawed.
Conversely, the fallback for an encryption key is to keep it around to decrypt existing documents, so it needs to be backed up reliably, in case of a leak of an encryption key, the confidentiality of all pre-existing documents is at risk, whereas new documents simply need to be encrypted with a different key.

Excercise 28

Kerberos is n authentication protocol that enables mutual authentication among different terminals in a network.
Kerberos Authentication Examples:

1. A user enters a user ID and password on the client.
2. The client performs a one-way hash on the entered password, which becomes the secret key of the client.
3. The client sends a clear-text message to the KAS, requesting services on behalf of the user. Neither the secret key nor the password is sent to the KAS.
4. The KAS checks to see if the client is in its database. If it is, the KAS sends back the following two messages to the client:
    - Message A: Client/TGS session key encrypted using the private key of the user.
    - Message B: Ticket-Granting Ticket (TGT), encrypted using the private key of the TGS. The TGT includes the client ID, client network address, ticket validity period, and the client/TGS session key.
5. Once the client receives messages A and B, it decrypts message A to obtain the client/TGSsession key. This session key is used for further communications with TGS. The client does not and cannot decrypt the Message B, as it is encrypted using TGS's private key. At this point, the client has enough information to authenticate itself to the TGS.
6. When requesting services, the client sends the following two messages to the TGS:

- Message C: Composed of the encrypted TGT from message B and the ID of the requested service.
- Message D: Authenticator (which is composed of the client ID and the timestamp), encrypted using the client/TGSsession key.

7. Upon receiving messages C and D, the TGS decrypts message D (Authenticator) using the client/TGS session key and sends the following two messages to the client, enabling access to the service corresponding to SAP NetWeaver AS:

   - Message E: Client-to-AS ticket (which includes the client ID, client network address, validity period) encrypted using the AS private key.
   - Message F: Client/AS session key encrypted with the client/TGS session key.

8. Upon receiving messages E and F from TGS, the client has enough information to authenticate itself to the AS. The client connects to the AS and sends the following two messages:

   - Message G: the client-to-AS, encrypted using AS's private key.
   - Message H: a new Authenticator, which includes the client ID, timestamp and is encrypted using client/server session key.

9. The AS decrypts the ticket using its own private key and sends the following message to the client to confirm its true identity and willingness to serve the client:

   - Message I: the timestamp found in the client's recent Authenticator plus 1, encrypted using the client/server session key.

10. The client decrypts the confirmation using its shared key with the server and checks whether the timestamp is correctly updated. If so, then the client can trust the server and can start issuing service requests to the server.

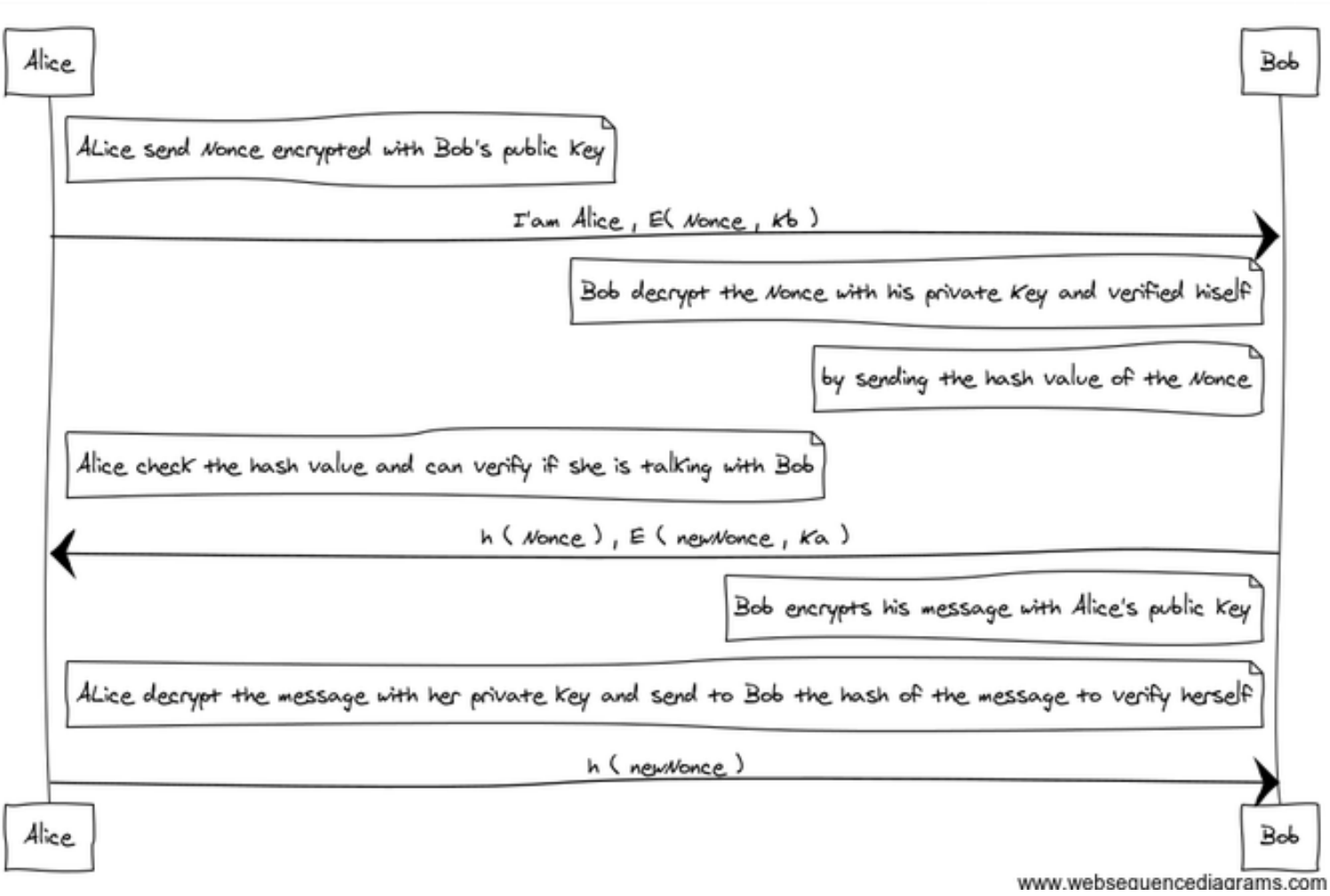
11. The server provides the requested services to the client. It holds additional authentication data, such as the ticket lifetime, and most important, the client's timestamp. When the Kerberos logic on a DC or resource server validates a Kerberos authentication message, it will always check the authenticator's timestamp. If the timestamp is earlier or the same as a previous authenticator, the server-side Kerberos logic will reject the packet because it considers it part of a replay attack and user authentication will fail. The Kerberos server-side logic also compares the timestamp in the authenticator to the local server time. If the timestamp in the authenticator isn't within five minutes of the

time on the server, it will also reject the packet. These five minutes are referred to as the Kerberos time skew.
The Client needs the timestamp in order to complete the above procedure. So this is why the client receives the encrypted timestamp. With timestamp and client's ID composed the authenticator.

Excercise 29

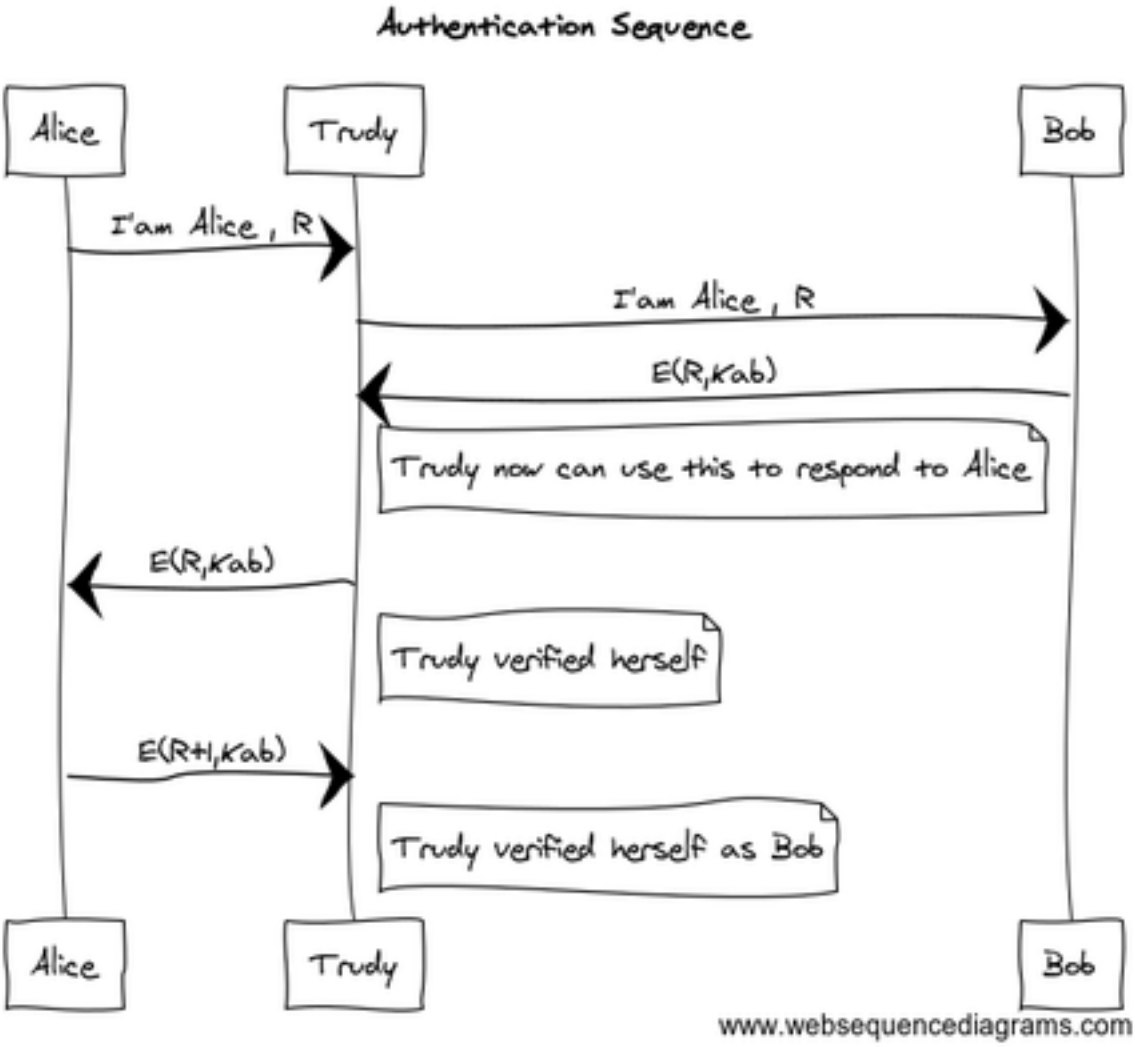
We used PKI(public key infrastructure):
Ka = ALice's public key
Kb = Bob's public key

The logic is that Alice send a message to Bob and a Nonce which is encrypted with Kb , so only Bob can decrypt it. Then Bob send the hash value of the nonce and Alice can check if the hash value is the same. With this way Bob can verify hiself. In addition Bob send a new Nonce encrypted with Ka and Alice following the same steps decrypt the Nonce and send the hash value back to Bob to verify herself. In the beggining the protocol used a shared key. We use PKI and hash function for verification.



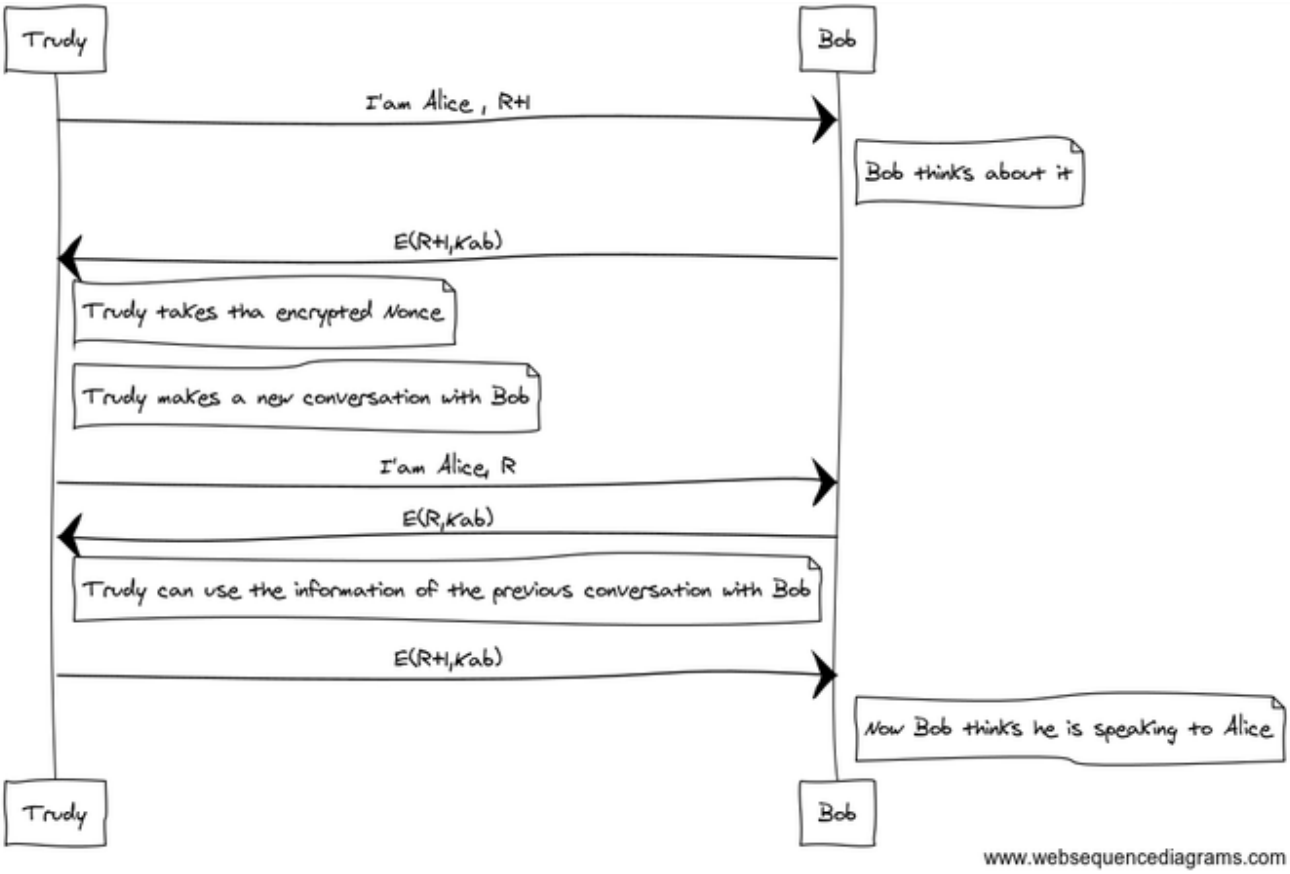
Excercise 30

Attack #1:
Trudy makes Mim(Man in the middle) attack, just forwarding the messages.
Steps:
Alice send to Bob 'I'am Alice with a Nonce, but this message goes first to Trudy and she forward the message to Bob. Bob replies with the encrypted Nonce with the shared key , now Trudy can use this and answer as Bob . Alice answers to Trudy the encrypted Nonce+1 and verifies herself to Trudy . Now Trudy knows R(Nonce) and E(R+1,Kab) so she can use them to initialize a new conversation with Alice/Bob with the same Nonce.



Attack #2:

Trudy initialize a conversation with Bob pretenting to be Alice , and sending a Nonce(R+1), Bob respone E(R+1,Kab) and encrypt the Nonce with the shared key. Now Trudy can start a new conversation with Nonce(R) , and she has already her answer ready E(R+1,Kab).



Exceercise 31

Under a security point of view is never ok to use ftp protcol to transfer file from/to a server. This is because the whole data exchange are plaintext so everyone can sniff packets and access to reservate data as password , username , shared files etc.
If we want to set a theoretical scenario under wich the ftp protocol is ok for our purposes then probably that would be when you need to upload/-download files (not critical files however) in a private network, in this case for the attacker is harder to gain access to critical data because first he has to break in the network (we suppose that the private network is secure). So basically is "ok" to use ftp when the mean of transport is cryptographically secure so no one can get (at least understand) your data. But is strongly discouraged to use it in any cases.
This because even if you share encrypted files you are still a possible victim of MiTM attack, or in a worst case scenario the Attacker can break in remotely to yours machine and inject malicious sofware.

Excercise 32

Q: how can Trudy, working at the postal service, gain access to the box's contents without Alice or Bob noticing it?
Attack :
Alice send the box **b** to Bob with his lock , let's say A[b]. Now trudy intercept this box and sends back the box with also her own lock T[A[b]] to Alice. Since Alice wait for a box with 2 locks she did not notice nothing suspicious and remove her lock from the Box, so now the box has only Trudy's lock T[B]. Now Trudy can open the Box and do his malicious things (i.e. copy or modify the content). Then Trudy lock the box again T[b] and send it to Bob that is waiting for a locked Box. Bob do not know that the lock he recieved is from Trudy so he lock the box with his own lock B[T[b]] and send it back to Alice. Finally Trudy recieve the Box and remove his own lock and send it back again to Bob that now can access the content of the Box.
Neither Alice nor Bob knows about Trudy's access to the content of the Box.

Alice —— A[b] ——> Trudy
Alice<—— T[A[b]]—— Trudy
Alice —— T[b] ——> Trudy
Trudy reads the content of b
Trudy —— T[b] ——> Bob
Trudy <—— B[T[b]] —— Bob
Trudy —— B[b] ——> Bob **Q:** What must be done to prevent Trudy from gaining access to the box's contents?

Bob or/and Alice has to verify that the first box comes from the right sender. but How to do this?
Alice and Bob have to add to the previous protocol a secret. Infact if Bob and Alice share a secret let's say K_{ab} (they can gain this secret in several ways) then they can use the box for writing their authentication protocol. The new protocol work in this way:
Alice write on the box a nonce R_A , then Bob writes on the same box E(Bob,R_A,K_{ab}), R_B. Alice finally writes back to Bob (on the box) E(Alice,R_B,$K_a b$). This protocol prevent Trudy's access to the content of the box because she cannot reply the right message on the box since she don't know the shared key.

Excercise 33

What is required to do proper authentication?
To do proper authentication is required that Alice (so the user, me) has

to verify and check the signature of the certificate. So when a popup or just a warning comes up Alice must not ignore it and take the right precautions.Not skip it.
meaning of doing authientication:
In technology terms, it refers to a client (web browser or client application) authenticating themselves to a server (website or server application) and that server also authenticating itself to the client through verifying the public key certificate/digital certificate issued by the trusted Certificate Authorities (CAs).
Is https://security.rug.nl properly authenticating itself? How and why if so? the answer is yes. Now we explain why and how:
Why: security.rug.nl authentication is checked by the openSSL tool in Linux whch turn a OK code.
How: In SSL authentication, the client is presented with a servers certificate, the client computer might try to match the servers CA against the clients list of trusted CAs. If the issuing CA is trusted, the client will verify that the certificate is authentic and has not been tampered with. This is done by 9 main steps. I found a nice explanation in www.codeproject.com/Articles/326574/An-Introduction-to-Mutual-SSL-Authentication , so the main steps of how a server auth himself to a client are:

1. Client sends ClientHello message proposing SSL options.
2. Server responds with ServerHello message selecting the SSL options.
3. Server sends Certificate message, which contains the server's certificate.
4. Server concludes its part of the negotiation with ServerHelloDone message.
5. Client sends session key information (encrypted with server's public key) in ClientKeyExchange message.
6. Client sends ChangeCipherSpec message to activate the negotiated options for all future messages it will send.
7. Client sends Finished message to let the server check the newly activated options.
8. Server sends ChangeCipherSpec message to activate the negotiated options for all future messages it will send.
9. Server sends Finished message to let the client check the newly activated options.

Q:Can you find an important difference between the certificates used by https://security.rug.nl and https://www.icce.rug.nl?

Either https://security.rug.nl and https://www.icce.rug.nl has verifiable and reliable certificates but we noticed a difference between them, infact connecting at security.rug.nl show up that this server has a 4096 bit public key , conversely icce.rug.nl server has a 2048 bit length key. This reflect in a different length of the certificates themselfs.

Q: Can you find a rug web-site whose https connection does not properly authenticate itself?
Unfortunately (or fortunately for RUG) we cannot find an https site that do not authenticate himself, we just noticed that http://rug.nl do not uses HTTPS and is vulnerable to a BEAST attack.