**Crypto, SSL, Certificates and more**

What is takes to set up a TLS connection on the web (https)

SSL

* Understand what is beneath
* TLS, Transport Layer Security

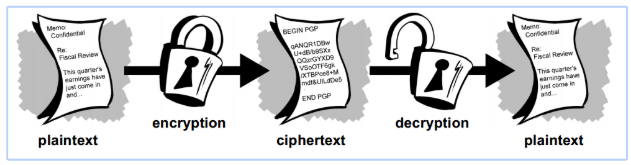
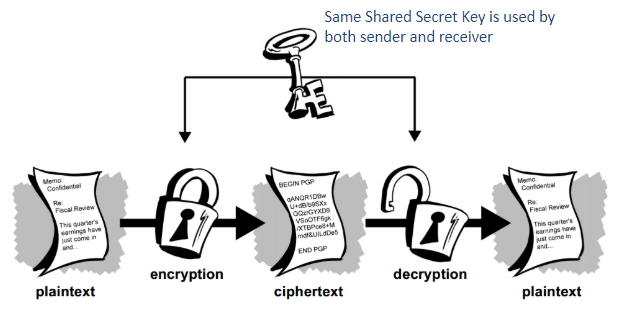
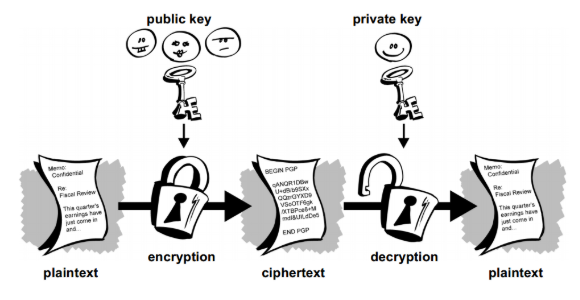
Cryptography

* Confidentiality
  + To send something form a to be, without anyone being able to read it.
* Integrity
  + No one can tamper with it. What we get after decryption is identical to what was sent.
* Non-repudiation
  + The creator/sender cannot deny at a later date, his or her intentions in the creation or transmutation of the information.
* Authentication
  + The sender and receiver can confirm each other’s identity and the origin/destination. Of the information.

Topics

* Symmetric encryption
* Asymmetric encryption
* Hash digests
* Certificate trust hierarchies
* Certificates
* Certificate Authorities
* Hashing
* MAC
* Digital signatures
* Cypher suites

Cryptography

* Basic terms
  + Modstå man in the middle attacks
  + 
* Symmetric encryption
  + Når vi er nået til et punkt hvor der oppe i vores browser står https, er vi i gang med symmetric encryption da det er meget hurtigere.
  + Asymmetric encryption bruges kun ved vores første key exchange
  + 
  + Før vi kan dele vores shared key i begge ender, er vi nødt til at få denne key udleveret gennem en sikker asynkron forbindelse.
  + How do we distribute keys?
    - We must get them over a secure asynchronous connection
  + How do we rotate these keys if they get compromised?
  + Symmetric-key algorithms Cryptography
    - DES (Data Encryption Standard)
    - 3DES (Triple DES)
    - AES (Advanced Encryption Standard)
* Asymmetric encryption
  + To forskellige dele, to keys
  + Public and private key
    - Public key is for encryption, Available to anyone
    - Private key is the only one that can decrypt a message from the corresponding public key.
      * No one should get their hands on this
    - They are mathematically related
  + End to end encryption
  + How do we secure that it is the correct public key that we receive, and not a middle man.
  + 
  + The idea is that you can’t derive one key from the other.
    - Du kan ikke få en public key ud af en private key og omvendt.
  + Public-key algorithm Cryptography
    - RSA (Named after the inventors)
    - Diffie-Hellman (Named after the inventors of Public-key Cryptography)
    - DSA (Digital Signature Algorithm)
  + Efficiency
    - Public key algorithms are much less efficient than symmetric-key algorithms
    - conventional 128-bit key for symmetric use is roughly equivalent to a 3000-bit public key. Størrelserne mellem de to typer af keys er totalt urelateret.
    - Key size is equivalent to time/recourses
    - These technologies are often used together in SSL and TSL

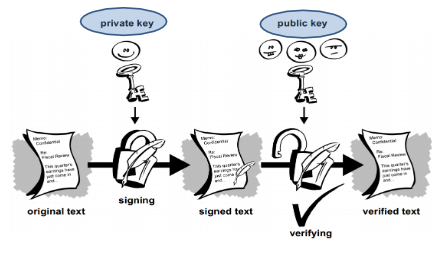
Caesar cipher

* An algorithm is that we do a shift forward in the alphabet.
* And a key for the amount on letter we switch.

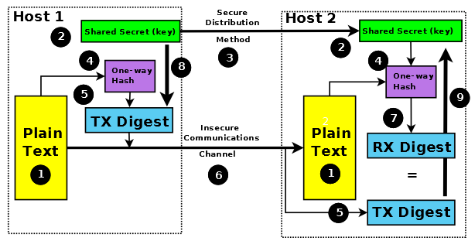
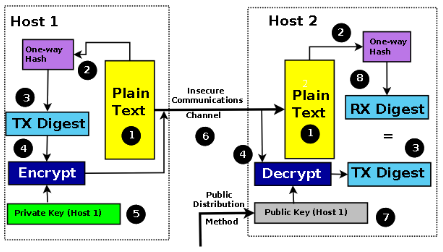
Public distribution

* How do we get the correct public key?
* How does the distribution work?
* Det er derfor at vi når vi starter en ny droplet bliver bet om at skrive yes for om vi stoler på den sendte public key.
* Vi ønsker at undgå man in the middle attacks.
* Jo større nøglen er jo mere sikker, er den, og jo mere kompleks er den, hvilket kræver flere resurser og mere tid.

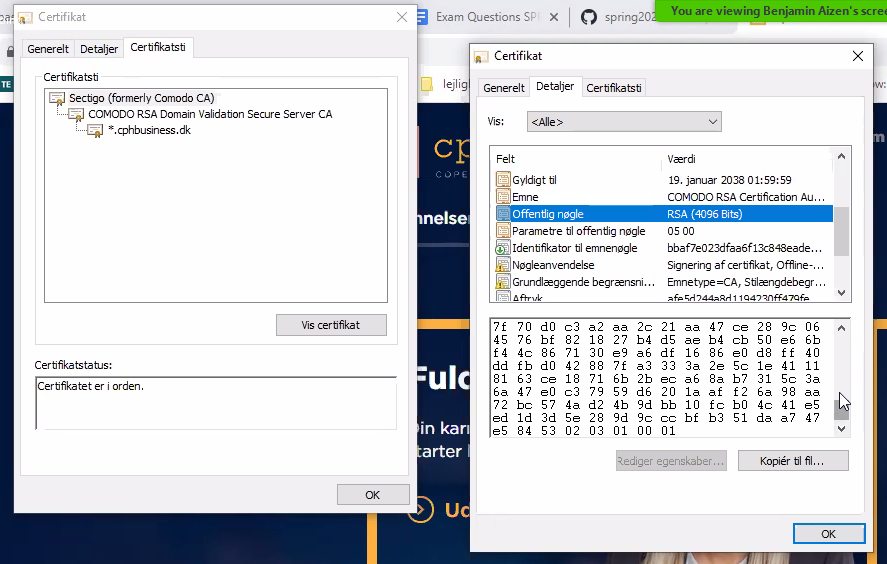
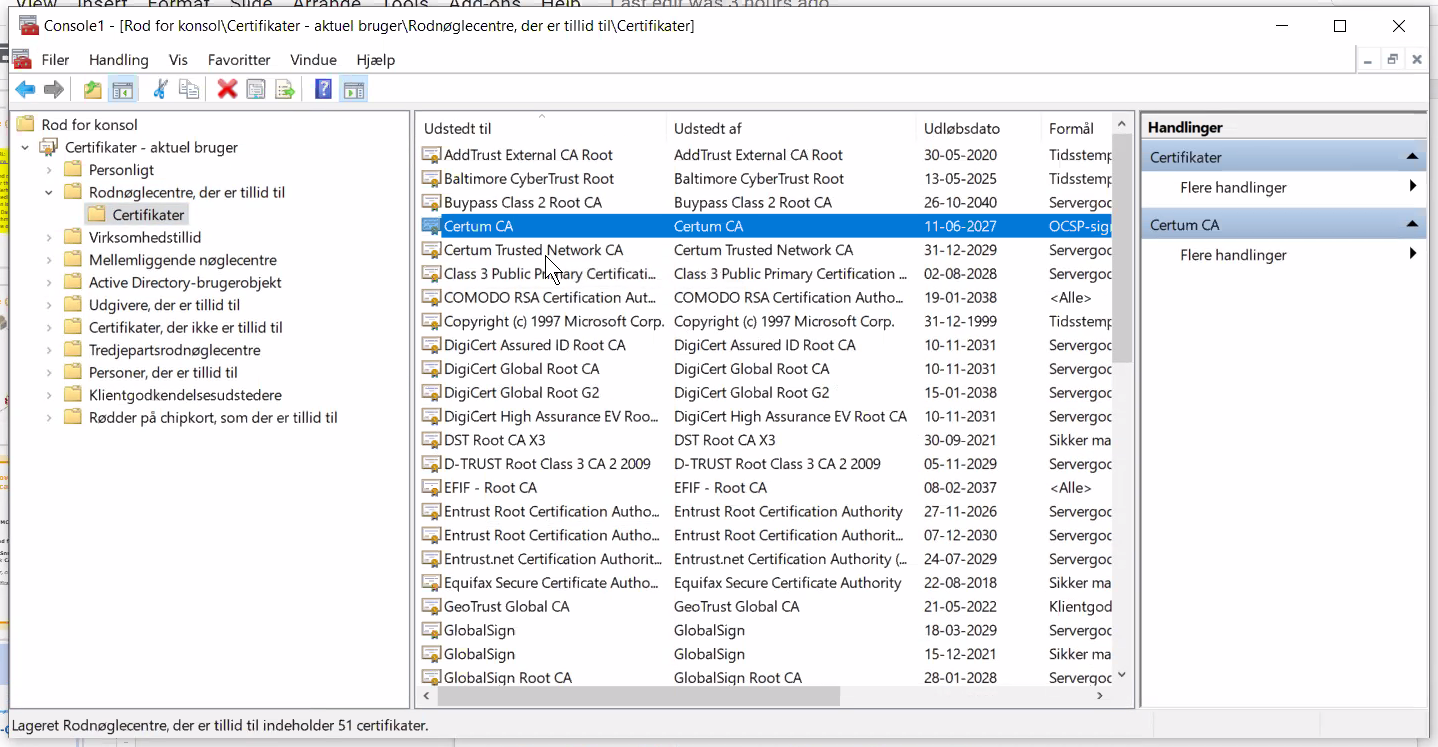
Digital Signatures

* Asynchronous Public key cryptography provides a method for using digital signatures.
* 
* If a message is encrypted with the private key (opposite of what you do if encryption is your goal), anyone with the public key, can decrypt the message. This gives us Digital Signatures.
* Signing a hole document can be slow

Hashing

* NOT ENCHRYPTION
* One-way algorithm.
* Compresses a message to a fixed size.
  + The smaller the hash digest is, the more likely we are to experience unsafe clashes where digests are the same.
* The output of a hash algorithm is called a digest.
* You cannot drive the original message from the hash.
* It is typical to compare hash digests since only the same algorithm on the same message can give the same hash.
* There are different algorithms that produce different digests.
* Cutting information into pieces of sizes which depends on the format (MD5, SHA2-family → SHA224, SHA256, SHA384, SHA512, etc).
* Cryptographic hash functions are used with SSH. These are methods for creating a succinct “signature” or summary of a set of information.
* Use cases
  + Checksums
  + Password hashing
  + Signatures (hashing fingerprints of our messages, and then encrypting them)
* MAC Message Authentication Code
  + Comparing hash digests over possibly insecure connections provides integrity
  + If we add a synchronously shared key to our hash digest, we can compare the digest including the shared key, so both provide integrity and authentication.  
    
  + If we use an asynchronous Public-key and we trust the public key provider, we can be sure that the public-private key pair has not been repopulated by a man in the middle. This provides Non-Repopulation. Now we can encrypt our hash digest, communicate it, decrypt it, and then compare the digest to a new hash of the message content, all while the key distribution is split by a matching private public key pair, that can not be switched by repopulation attempts.  
    
  + We can only achieve this if we trust the public key distributor.

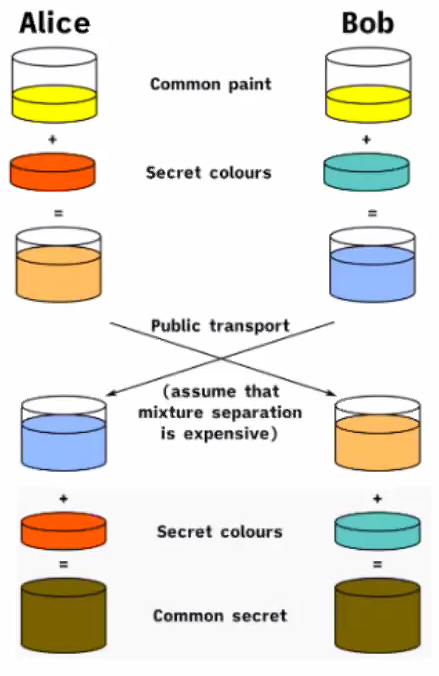
Digital certificates

* All sites we communicate with has a certificate that tells us who they are, to let our machine trust them.  
  
* Root certificate
* Intermediate certificate
* Hvis man går ind under brugercertifikater, kan man se en liste af de kendte certifikater som vi stoler på ved denne maskine.  
  

Cipher suites

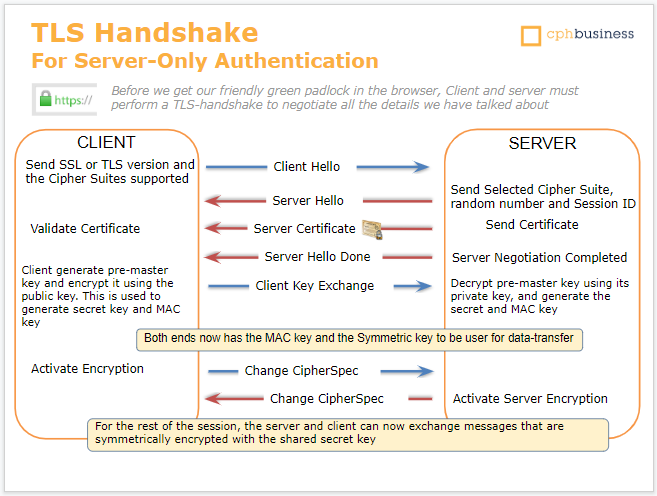
* En made hvorpå browseren sender sikkerheds præferencer ud til serveren, så vi kan være sikker på at vi laver en sikker forbindelse. En algoritme der bliver brug til at finde ud af hvordan vi vil udveksle vores key.
* 
* Example key exchange algorithms: RSA, DH, ECDH, ECDHE;
* Example authentication algorithms: RSA, DSA, ECDSA;
* Example bulk encryption algorithms: AES, 3DES, CAMELLIA;
* Example MAC algorithms: SHA, MD5

Diffie Hellman key exchange

* Ved at man sender noget frem som alle kan se, kan vores partier generere en shared secret som kun de kan se. En måde at sende data over et public Network
  + Først vælger vi en fælles farve som alle kan se.
  + Derefter vælges der 2 individuelle farver
  + Farverne i hver side lægges sammen
  + De to nye farver overføres til den anden kommunikative parti
  + Derefter ligges de originale selvvalgte farver sammen med de overførte
  + Dette danner den samme farve i begge sider
  + Af denne grund kan de nu snakke sammen sikkert og på samme niveau som kun de kender.
  + Vi har skabt en hemmelighed i begge ender. En fælles shared secret.
* 

Hvad skal der grundlæggende til for at få en sikker forbindelse

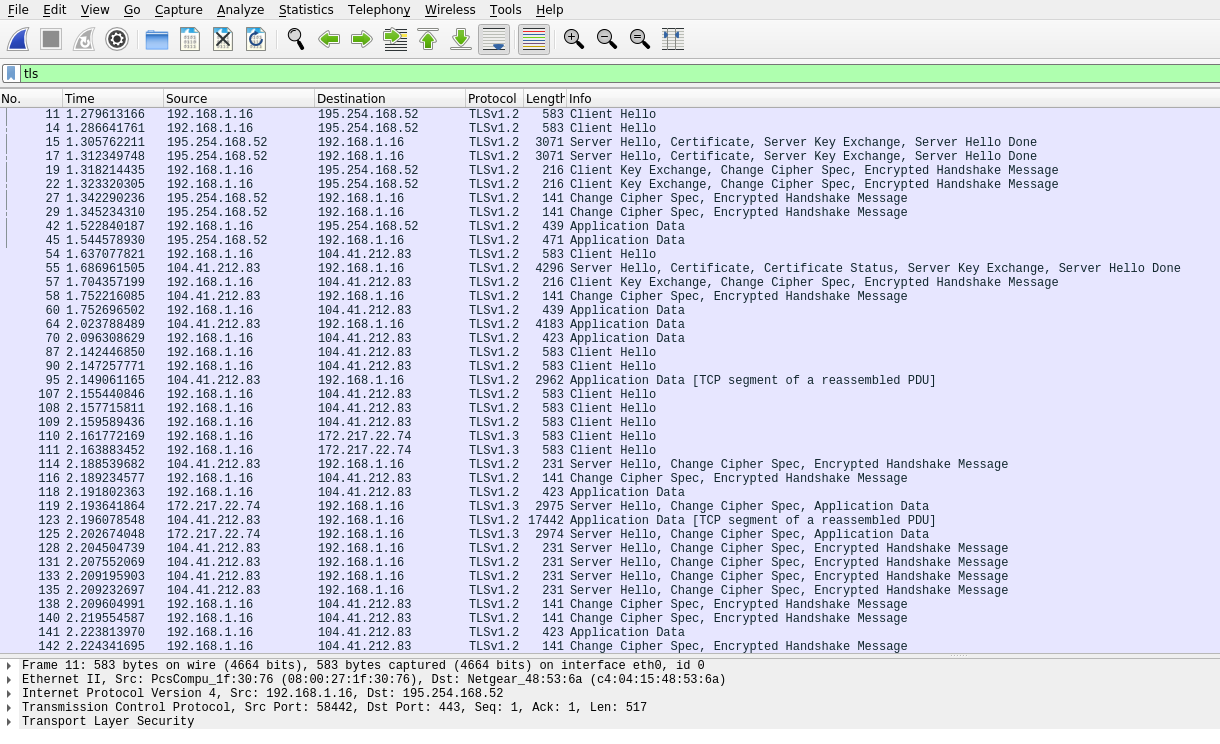
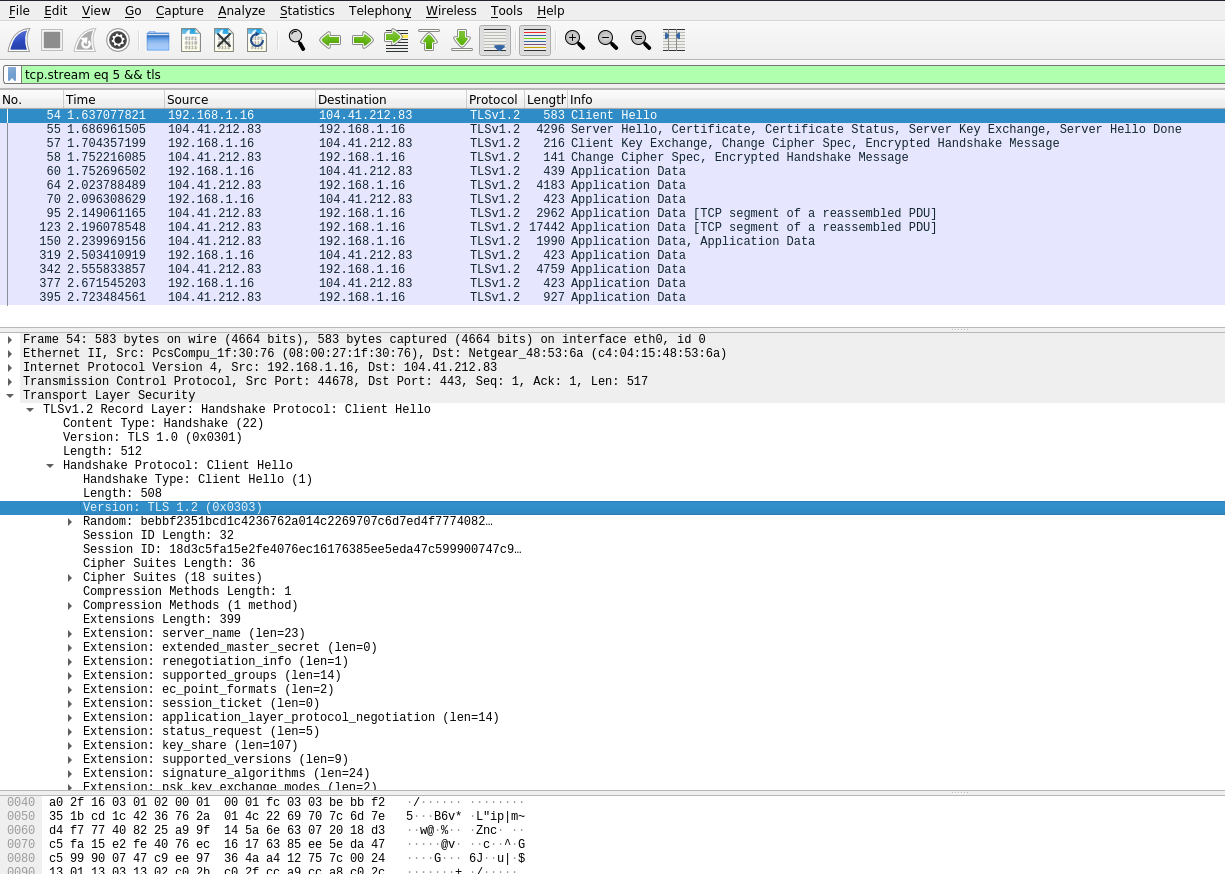
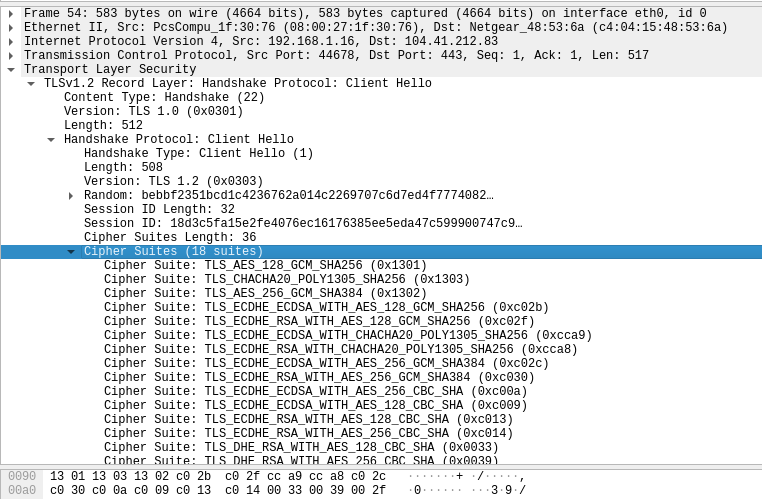
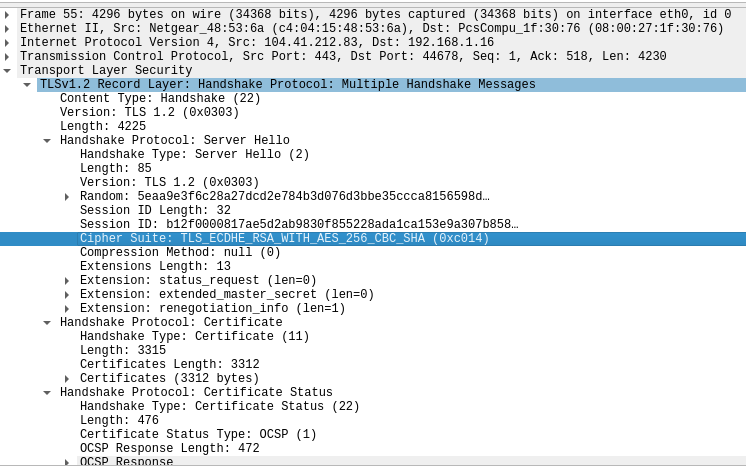
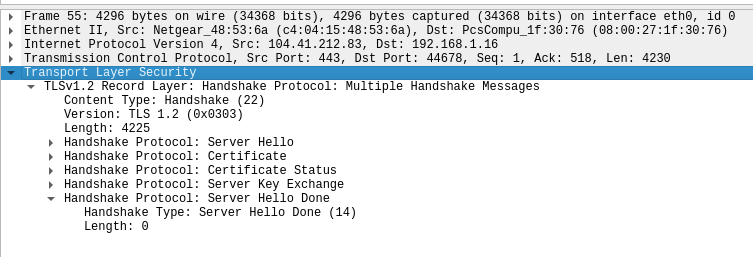
Når vi ser hængelåsen, er alt det her lavet!

* Client sender en besked til serveren vedrørende SSL-version og supportede cipher suites.
* Serveren vælger en cipher suite og generere et sessionid der så sendes til klienten
* Serveren sender derefter et certifikat.
* Klienten validerer derefter certifikatet fra serveren om mod sin egen liste af accepterede certifikater.
* Serveren har nu completed sin negotiation.
* Klienten kan nu generere en key og kryptere den med en asynkron public key for at generere en secret key og MAC key, hvorefter der bliver udført en Key-Exchange med serveren, som så selv kan dekrypter denne key og bruge dets private key til at generere den samme Secret og MAC-key. Altså en synchronized Shared key.
* Encryption kan nu aktiveres ved at ændre Cipher-Spec fra både klienten og serveren side.
* Fortsat kommunikation vil nu forgå over encryption med en synchronized shared key.  
  

Det er vigtigt at huske

* Efter application layer, før vi når TCP transportlaget, rammer vi SSL/TLS-laget.

Wireshark example

* Hvordan vi selv kan lave vores eget certifikat:
* 
* Vores handshake og Diffie-Hellman exchange:  
  
* VI sender hvilke Cipher-Suite som vi kan forstå til serveren, så den kan vælge:
* Nu har serveren valgt en Cipher-Suite:
* 
* Her kan vi se hvad der bliver udvekslet over et handshake mellem server og klient:  
  
* Data nedenfor er krypteret og kan derfor ikke umiddelbart læses:  
  