AARHUS UNIVERSITET

Microservices and DevOps

DevOps and Container Technology

Security 101 – Encryption and TLS

Henrik Bærbak Christensen

AARHUS UNIVERSITET

Security

- Concerned with *ability to protect data and information from unauthorized access while still providing access to people/systems that are authorized*

- Tactic: *Encrypt Data*

- Actually two things
  - **During transport (network)**
  - At rest (databases)

Figure 9.3. Security tactics

AARHUS UNIVERSITET

Confidentiality

By encryption

Symmetric Encryption

- Oldest trick in the book, knowingly used by The Romans
  - 'Attack on Tuesday' to be broadcast, but not understood by enemy

Issue:
How to distribute the encryption key?
The key distribution problem

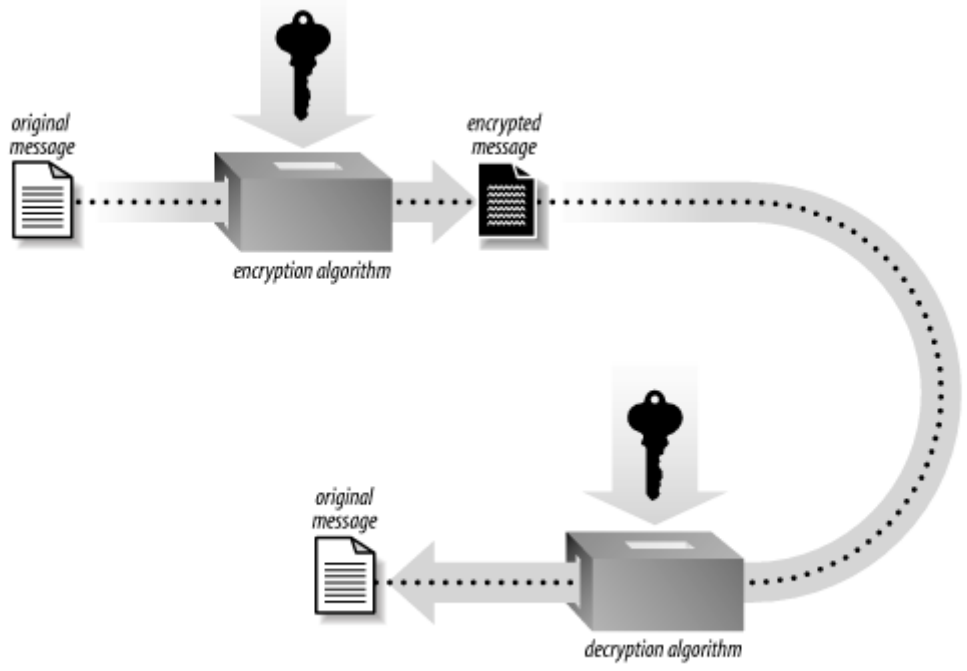


Figure 1-1. Symmetric key cryptography

AARHUS UNIVERSITET

World War II: Enigma

- Enigma was a symmetric encryption
  - A machine, needed a ‘start setup’ of rotators
  - A code book stating the start setup for every day
- Was cracked because
  - The got hold of a machine
  - One of the watched coders used the same setting every day
  - Some messages contained the same string
  - The Brits had Alan Turing ☺ ☺ ☺
- Today? https://www.youtube.com/watch?v=RzWB5jL5RX0

Asymmetric

- User has *two* keys
  - Public + Private
- Public
  - Is public ☺
- Private
  - Is private/secret ☺
- *Coupled* keys
  - *Encrypting* with one allows **decrypting** with the other
    - Pretty cool – allows wonderful stuff

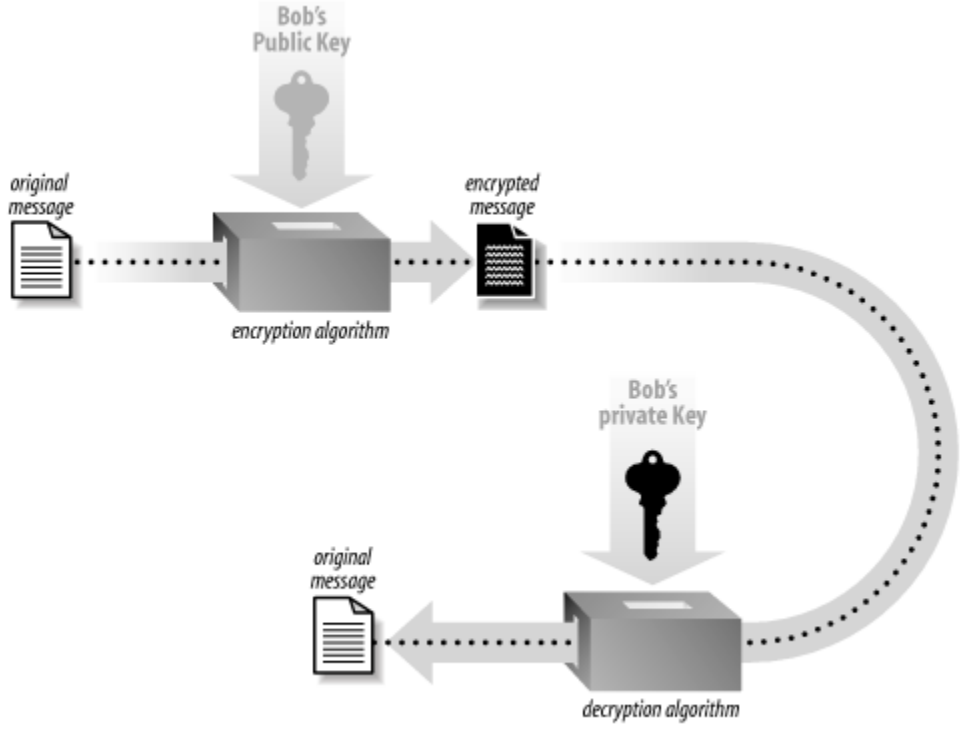


Figure 1-2. Public key cryptography

Asymmetric

- Now, Alice can encrypt the message to Bob using Bob's *public* key…
  - Given that she knows Bob's public key…
- And only Bob can decrypt it, using *private* key
  - As long as he does not share his private key…

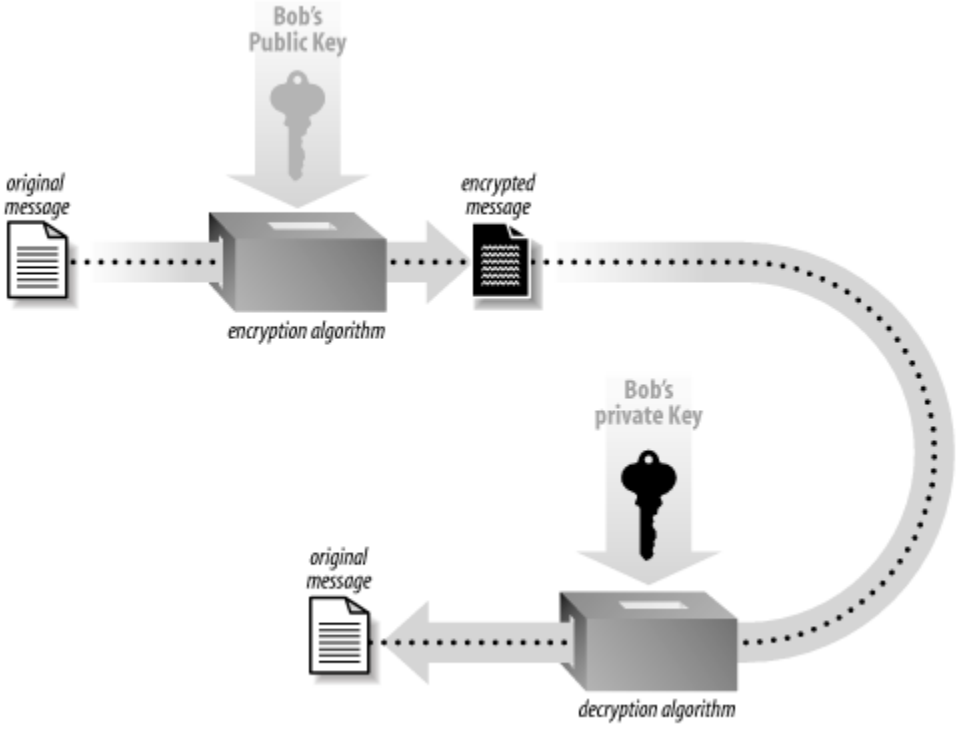


Figure 1-2. Public key cryptography

Example

- I want to transfer kr. 1.000 from my own account to my eldest son
  - And I do not want anyone else to do similar stuff!
- Thus I encrypt that message…
  - Actually an API call on my bank's REST server
- … with my bank's *public key*
- Thus no one but the bank can read that message
  - Ensuring confidentiality

AARHUS UNIVERSITET

Getting Bank's Public Key

- Of course, it does not help, unless I can get my bank's public key…
  - 1) I must get Nykredit's key
  - 2) I must trust that the key is actually from NyKredit
    - I.e. not some attacker wanting to appear to be my bank!
- *The key distribution problem* again
  - Important, but let us look at the protocol for exchanging the message with my bank first…

AARHUS UNIVERSITET

TLS: Transport Layer Security

AARHUS UNIVERSITET

The Confusion

- TLS: Transport Layer Security
- SSL: Secure Sockets Layer

- SSL is the original (Netscape developed) protocol, but vulnerable to attacks and deprecated, superseded by TLS
  - TLS 1.3 / August 2018

- In practice, the term ‘SSL’ is still in widespread use today, but what is meant is actually TLS (or should be!)

The TSL Protocol

- Client get server's *certificate* from server
- Client encrypt *challenge* using server's public key
  - Just a random string
- Server decrypt and send challenge back to client *unencrypted*.
- If they match, client knows it is the right server…
  - The only one able to dechiper…

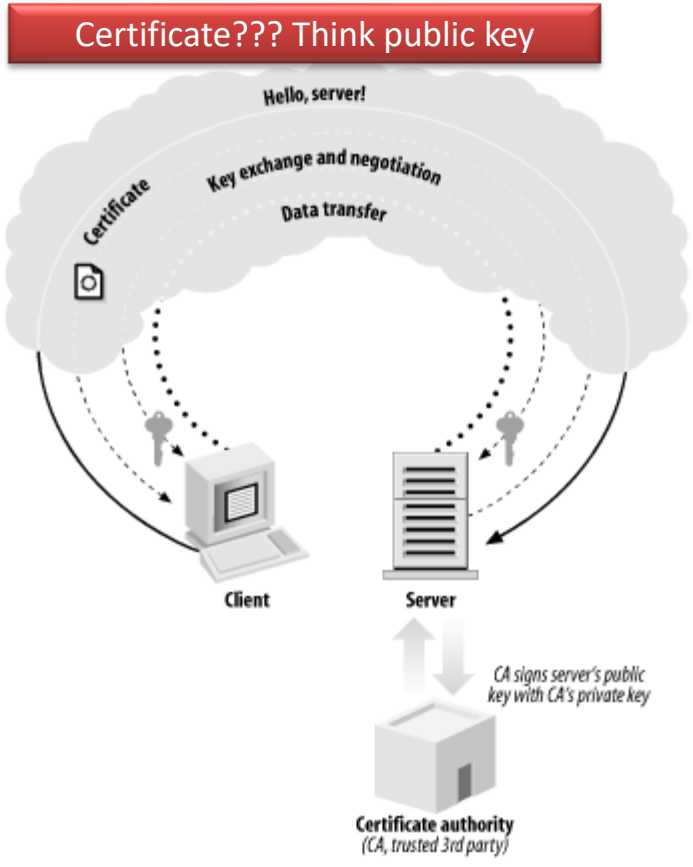


Figure 1-3. An overview of direct communication in SSL

AARHUS UNIVERSITET

Performance

Public key cryptography has a significant drawback, though: it is intolerably slow for large messages. Symmetric key cryptography can usually be done quickly enough to encrypt and decrypt all the network traffic a machine can manage. Public key cryptography is generally limited by the speed of the cryptography, not the bandwidth going into the computer, particularly on server machines that need to handle multiple connections simultaneously.

As a result, most systems that use public key cryptography, SSL included, use it as little as possible. Generally, public key encryption is used to agree on an encryption key for a symmetric algorithm, and then all further encryption is done using the symmetric algorithm. Therefore, public key encryption algorithms are primarily used in key exchange protocols and when non-repudiation is required.

- That is
  - Once I have made the TLS handshake, client and server agree on a *symmetric encryption key*
  - Use that in further encryption of the bulk of messages…

- Still – How do I get the public key? And trust it?
  - The key distribution problem

AARHUS UNIVERSITET

Certificates

Part I: Getting the public key

AARHUS UNIVERSITET

Certificates

- Part I: I need to get that key
- Solution: Certificates
  - **Certificate = public key + organizational information (+..)**
    - That is, it is essentially just an object with some fields ☺
  - A certificate is issued by ‘someone’, is publicly available, as it only contains identification information and the public key.
  - My client/browser fetches it as the first aspect of the TLS protocol
    - ‘get the server’s certificate’
    - So it is part of the TLS protocol itself

Certificate

- Example: Certificate for www.baerbak.com (excerpt)

Man in the middle

- However, ‘man-in-the-middle’ can intercept client’s first request and just spoof it all ☹

Figure 1-4. A man-in-the-middle attack

- That is, an attacker persuades me to hit
  - https://www.nyekredit.dk instead of https://nykredit.dk
  - Just sends the spoof site’s certificate instead ☹
- This is Part II: How to trust the certificate?

AARHUS UNIVERSITET

Certificate Authority

Part II: Can I trust the key?

AARHUS UNIVERSITET

A matter of Trust

- TLS simply reuses the same process that most of us use to gain trust in something
  - If someone that I trust says, that I can trust you, then I will
    - (Get someone to recommend a craftsman, before accepting offer ☺)
  - *Certificate Authorities*
    - A trusted 3rd party that has validated that the certificate is 'the real thing'
    - They say so, by *digitally signing the certificate*
  - *Eh – what is a digital signature?*

Signatures

- A **Digital Signature**
  - The digital equivalent to ‘it is really me’
- Alice needs to sign a message, stating that it is indeed her, that has written it
  - Alice computes a *hash* of the message (a checksum)
  - Alice encrypts the hash with her *secret key -> the signature*
  - Alice adds the signature to the message
- Bob can now
  - Decrypt the hash, using Alice’ public key
  - Compute the hash of the message, and compare with signature
    - If same, it *must be Alice* who signed it

AARHUS UNIVERSITET

Hashes

- Hash function is an *injective* function
  - For any given input you always get the same output
  - However, given the output you can never compute the input
    - It is not a bijection, not an invertible function
  - Near-impossible to find two inputs who provide the same output

Cryptographic hash functions are essentially checksum algorithms with special properties. You pass data to the hash function, and it outputs a fixed-size checksum, often called a *message digest* , or simply digest for short. Passing identical data into the hash function twice will always yield identical results. However, the result gives away no information about the data input to the function. Additionally, it should be practically impossible to find two inputs that produce the same message digest. Generally, when we discuss such functions, we are talking about one-way functions. That is, it should not be possible to take the output and algorithmically reconstruct the input under any circumstances. There are certainly reversible hash functions, but we do not consider such things in the scope of this book.

AARHUS UNIVERSITET

Back to the Certificate

- A certificate is actually
  - (public key, organization information, ***issuer's signature***)
- **Certificate Authority** (CA)
  - A trusted 3rd party that
    - Does some background verification of you (like issue a passport)
    - *Issues* the certificate *and adds a digital signature*
  - So you can now use the CA's public key to verify that it indeed did issue that particular certificate
  - *Chain of trust:*
    - I trust you because the CA trusts you – and I trust the CA

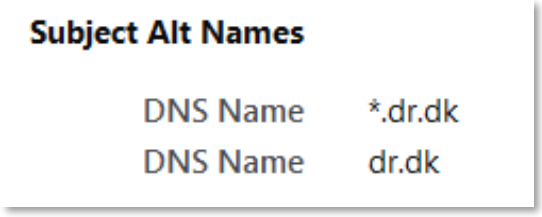
AARHUS UNIVERSITET

So the TLS protocol is…

- Client gets server's certificate
  - Certificate = (public key, organizational info, CA's signature)
- Client extracts the signature from certificate and…
  - Decrypts it using CA's public key to get the hash
  - Computes the hash of the certificate by itself
  - If they match, then the CA must have issued the certificate, and I can trust it…
  - And go on and use the public key to exchange symmetric keys etc.
    - But…

But…

- The certificate also explicitly tells *which DNS names it is issued for*
  - *Subject Alt Names – SAN section*
- Example:
  - www.dr.dk

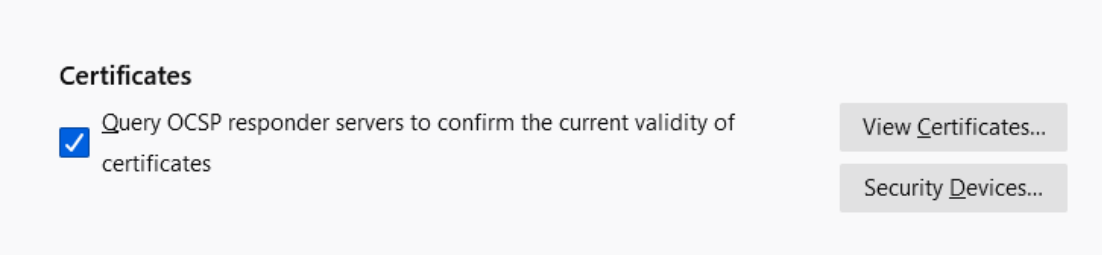


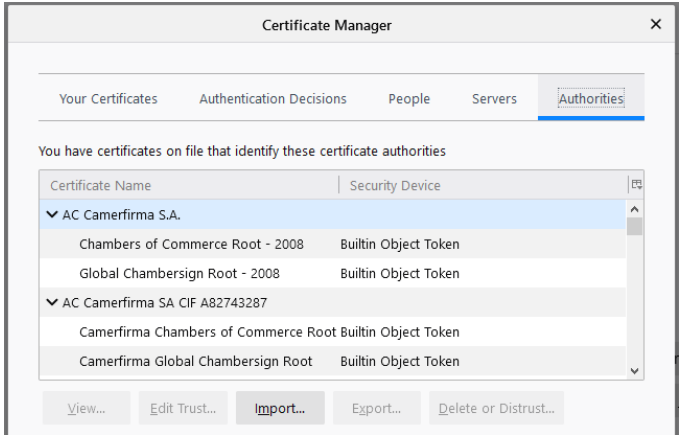
- That is, it is only valid
  for contacting URLs that reside on DNS: *.dr.dk
  - A java server will refuse to start if its hosting DNS is not listed in the SAN section of the certificate

AARHUS UNIVERSITET

Who are the CA's?

- Something is missing here – who do I trust?
  - Some one has to be trusted in the first place! The CAs!
- *It comes built-in with your software.*
- Example: Firefox defaults to trusting these CAs





AARHUS UNIVERSITET

What about applications?

- Not all communication is browser based.
- Java
  - Comes with a long list of trusted certificates in its *trustStore*
  - The tool is ‘keytool’
    - Here used to list all trusted certificates
      - Try it. The password is public (why?): ‘changeit’ ☺

```
csdev@m1:~/proj/cave$ keytool -cacerts --list
Enter keystore password:
Keystore type: JKS
Keystore provider: SUN

Your keystore contains 128 entries

cavereg2021, Mar 4, 2021, trustedCertEntry,
Certificate fingerprint (SHA-256): 42:53:CA:C0:D8:5A:23:0B:73:E1:BD:EE:9A:26:07:E1:8D:68:69:2D:A2:84:49:96:1E:EC:B0:A1:76:2C:64:F5
debian:ac_raiz_fnmt-rcm.pem, Sep 23, 2020, trustedCertEntry,
Certificate fingerprint (SHA-256): EB:C5:57:0C:29:01:8C:4D:67:B1:AA:12:7B:AF:12:F7:03:B4:61:1E:BC:17:B7:DA:B5:57:38:94:17:9B:93:FA
debian:accvraiz1.pem, Sep 23, 2020, trustedCertEntry,
Certificate fingerprint (SHA-256): 9A:6E:C0:12:E1:A7:DA:9D:BE:34:19:4D:47:8A:D7:C0:DB:18:22:FB:07:1D:F1:29:81:49:6E:D1:04:38:41:13
debian:actalis_authentication_root_ca.pem, Sep 23, 2020, trustedCertEntry,
Certificate fingerprint (SHA-256): 55:92:60:84:EC:96:3A:64:B9:6E:2A:BE:01:CE:0B:A8:6A:64:FB:FE:BC:C7:AA:B5:AF:C1:55:B3:7F:D7:60:66
debian:affirmtrust_commercial.pem, Sep 23, 2020, trustedCertEntry,
```

AARHUS UNIVERSITET

Java Servers And Clients

Enabling TLS in Java

AARHUS UNIVERSITET

Java Client Code

- Actually surprisingly easy:
  - Just remember the ‘https://’ in the call

```
// The the local running server
response = Unirest.get("https://localhost:7777/hello/Mathilde")
        .asString();
System.out.println("--> Received: " + response.getBody());
```

- Now our HTTP library (here Unirest) will do the TLS handshake, ensure it is talking to the right server, and encrypt further calls.

Java Client Code

- For self-signed certificates you have two options
  - Disable the certificate check (!)

```
// The brute force way to accept any certificates...
// Unirest.config().verifySsl(false);
```

  - Exercise:
    - Is this any better than just using plain HTTP ?

AARHUS UNIVERSITET

Java Client Code

- Or… tell the JVM to trust the certificate
  - Retrieve the certificate using ‘some tool’ or browser
    - Or get the file from the owner (key sharing ☺)
  - Import it into the local machine/Java’s trustStore (using ‘keytool’)
  - Or tell Java to use another local trustStore
    - System Property: **javax.net.ssl.trustStore**
    - Caveat: Now this server is the *only one trusted!*
      - Because the trustStore does not mention the official CA’s !
  - So the better one is:
    - Create a copy of ‘cacerts’, import the certificate into it, and tell Java to use another trustStore than the default one
- **Morale: Much more cumbersome that CA issued ones**

The Server side of it…

- MSDO has a strong ‘server side’ focus
  - A server must present its certificate upon request
    - Same procedure for CA issued as well as self-signed certificates
- How to do that?
  - Put certificate into a *keyStore*
  - Tell library to use that
    Example: SparkJava

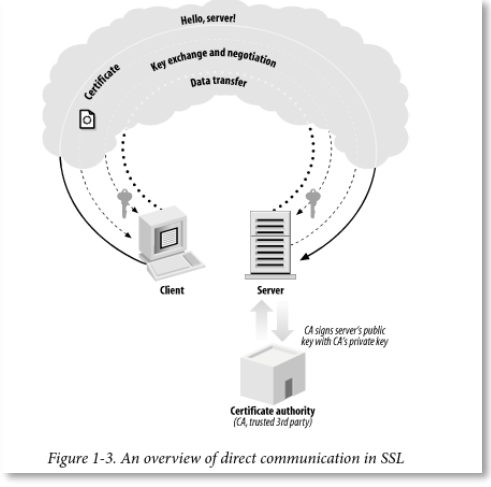


Figure 1-3. An overview of direct communication in SSL

```
// Officially, use the System properties
String keyStoreFile = System.getProperty("javax.net.ssl.keyStore");
String keyStorePwd = System.getProperty("javax.net.ssl.keyStorePassword");

System.out.println( " Using keystore: "+ keyStoreFile);
secure(keyStoreFile, keyStorePwd, null, null);
```

Second parameters are the ‘trustStore’ ones. ‘null’ = use default…

AARHUS UNIVERSITET

Getting Certificates

You do not need the CA, really

AARHUS UNIVERSITET

The two paths

- There are essentially two paths
  - Request a **CA to issue a certificate** for you (production system)
    - Which entails a slow process + a financial cost
      - Some free services, like *Let's Encrypt*
    - And severely limits the DNS
      - You cannot contact https://localhost:7777 if localhost is not mentioned in the certificate's SAN section…
        - » It is about securing access, right
  - Make a **self-signed certificate** (staging system)
    - Which I can do fast and easy, and set up any SAN section
      - Like 'localhost' and '152.143.23.11'
    - Downside
      - Clients have quite a lot of manual work to do…

AARHUS UNIVERSITET

Self-signed Certificates

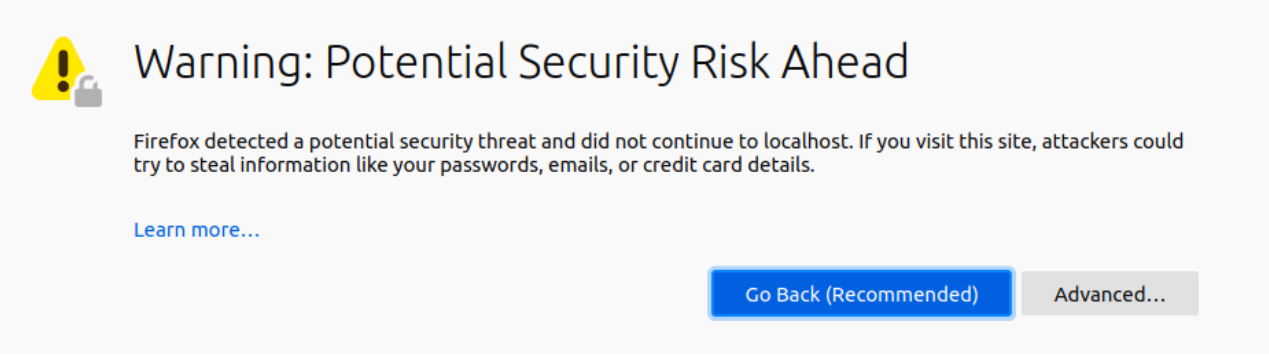
- For staging environments (or if you just want encryption), you may create your own certificates: ‘self-signed’
- In Java world, again ‘keytool’ does the trick

```
keytool -keystore baerbak.jks -alias henrikbaerbak \
  -validity 3600 -genkeypair -keysize 3072 -keyalg RSA\
  -ext SAN=dns:cavereg.baerbak.com,dns:malkia00.home,dns:malkia.st.lab.au.dk,dns:localhost,ip:127.0.0.1
```

- Here
  - Create ‘baerbak.jks’ with a 3072 bit RSA key, valid for 10 years, that will work on the given SAN DNS entries
    - Cavereg.baerbak.com, malkia00.home, 127.0.0.1, …
  - Means I can test my secure server in various settings!

Trusting Self-signed Certs

- For instance in Firefox



- Click ‘advanced…’, review the certificate, and proceed if you feel confident
  - Will add the certificate to the trust manager, so no need the second time around

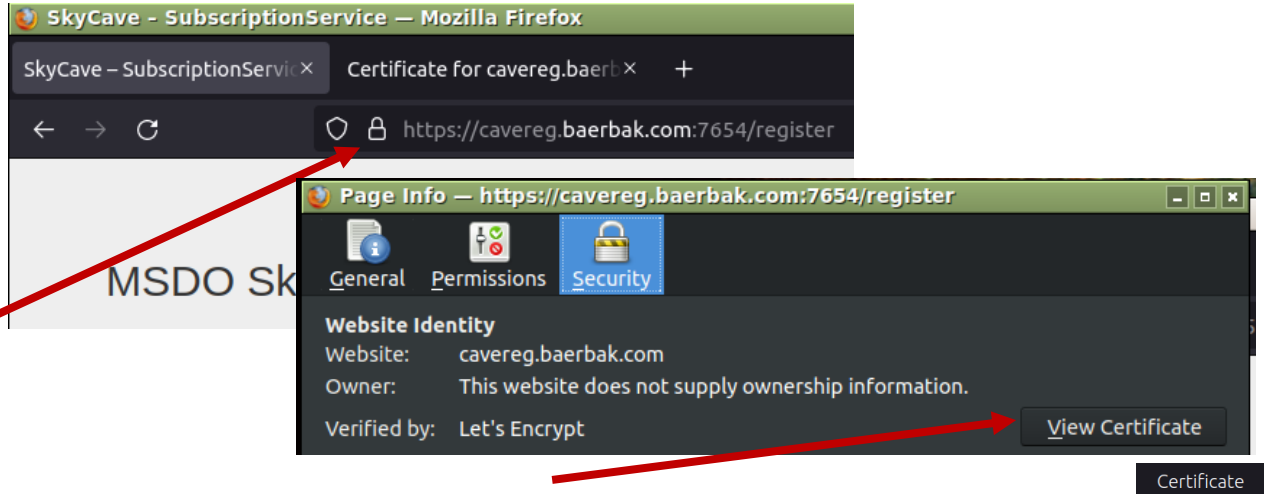
AARHUS UNIVERSITET

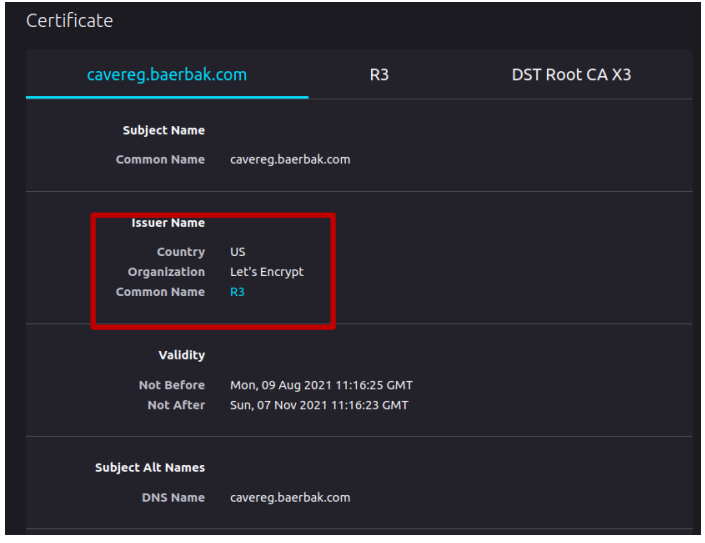
Let's Encrypt

- Original Plan for MSDO for SubscriptionService
  - Make a self signed certificate
    - Creates hell for all of you as you need to enter it into each and every machine's trustStore ☹
- Finally gave in, and used Let's Encrypt, a free service
  - Turned out to be somewhat *easier than anticipated*…
  - Costless certificate, but
    - Expires after 3 months; have to get a new one and install in keyStore
    - Only SAN is on that given machine
    - Only guaranty is that 'you hit the right machine'

AARHUS UNIVERSITET

To See It...



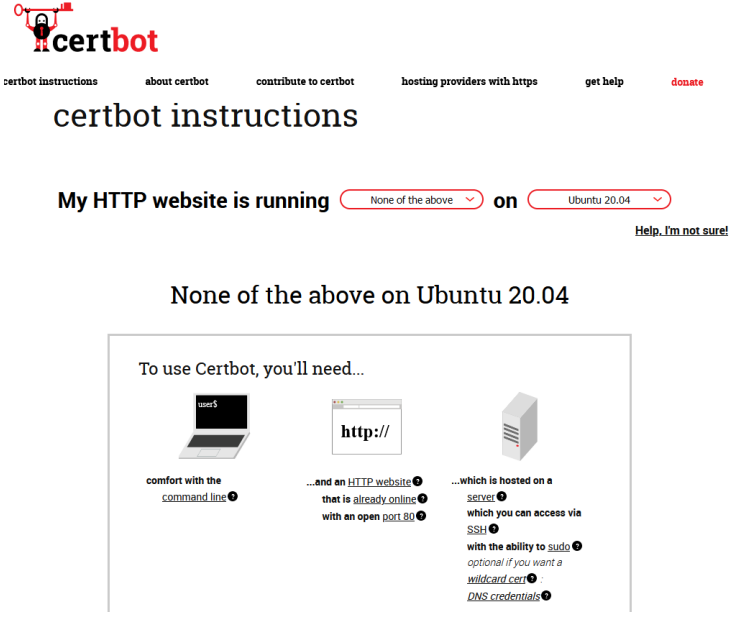


AARHUS UNIVERSITET

Let's Encrypt

- Uses a tool 'certbot', which comes with a guide.
- Install and then

```
$ sudo certbot certonly --standalone
```



- … will copy a certificate to your machine, which

8. Install your certificate

You'll need to install your new certificate in the configuration file for your webserver.

  - That is, use the 'keytool' to convert to a *keyStore* and tell HTTP lib to secure() using it.
    Jobs done… ***Again in three months time…***

AARHUS UNIVERSITET

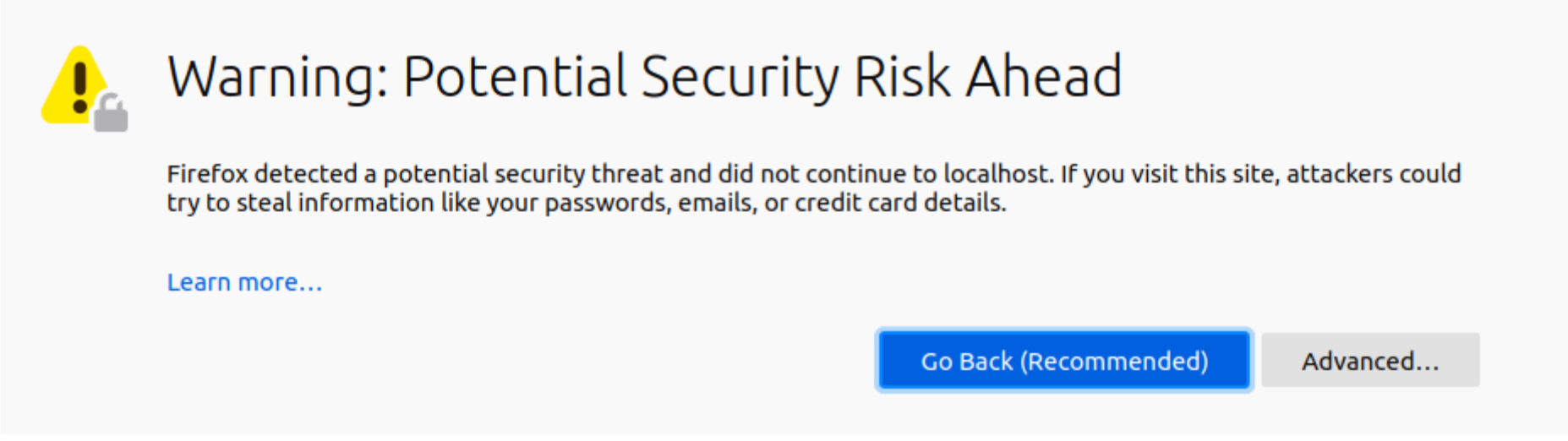
Extras

Importing PEM into TrustStore

AARHUS UNIVERSITET

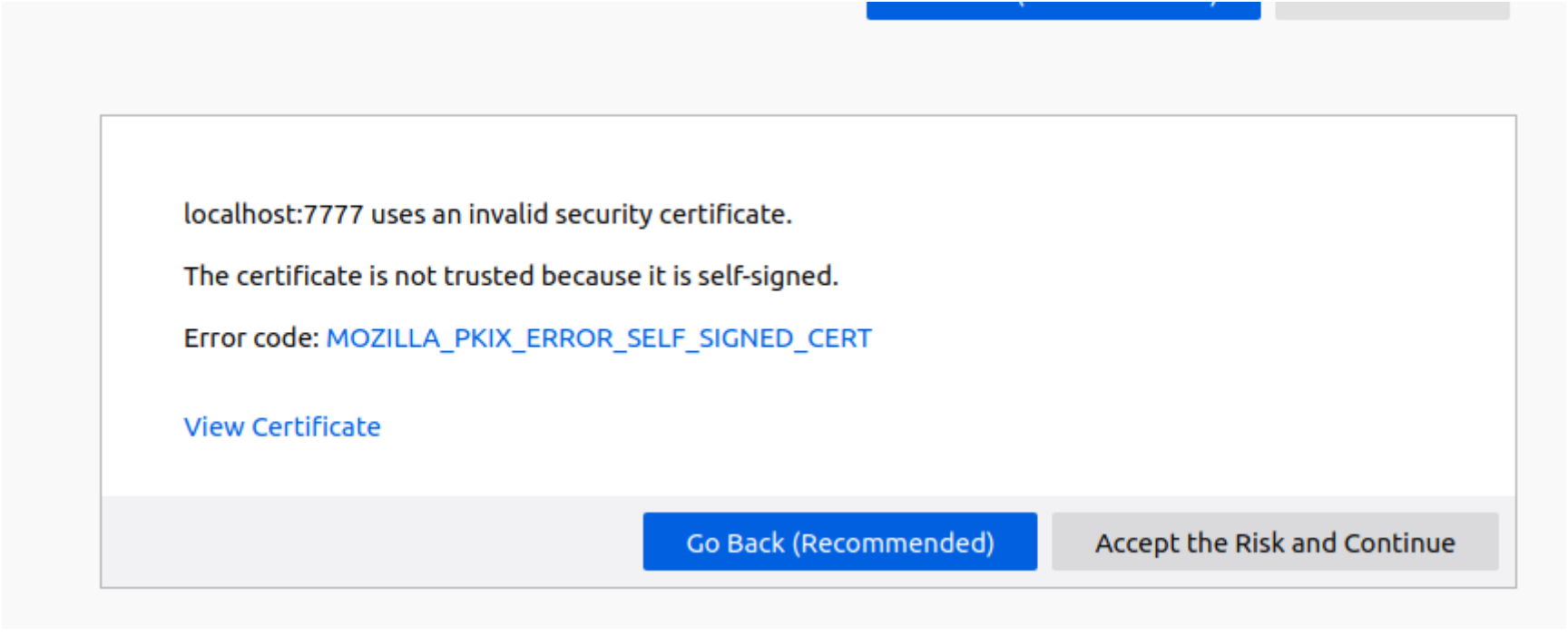
1. Browse to the site

- You will be warned



- Click ‘advanced…’

2. Get it



- Click ‘view certificate’…

AARHUS UNIVERSITET

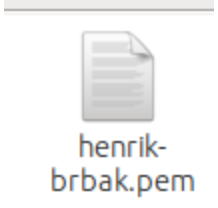
3. Verify and Download

- Verify that it is indeed ‘me’

Miscellaneous

Serial Number	23:1F:0F:32
Signature Algorithm	SHA-256 with RSA Encryption
Version	3
Download	PEM (cert) PEM (chain)

- And download the ‘PEM (cert)’ file



AARHUS UNIVERSITET

4. Trust it

- Now tell Java to trust that certificate by importing it into the ‘trust store’
  - You do that as super user ‘sudo’

```
sudo keytool -cacerts -importcert \
  -file ~/Downloads/henrik-brbak.pem \
  -alias msdo2021
```

  - The alias is just a call name for the certificate
  - Remember, the default password of the truststore is ‘changeit’
  - You will be asked to confirm by typing ‘yes’

AARHUS UNIVERSITET

5. Onwards

- Now you can make clients that just use ‘https’ calls to websites that uses that signature.

- To remove trust, you can

  – Sudo keytool –cacerts –delete –alias msdo2021