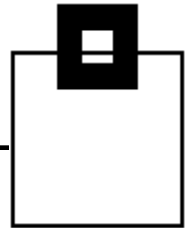
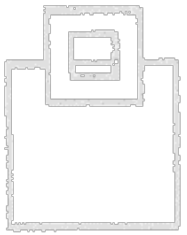
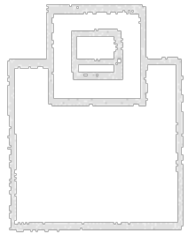


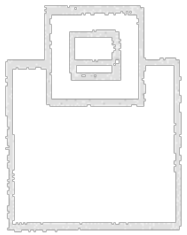
Understand, manage and love certificates in z/OS and USS



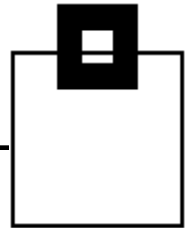


Ulf Heinrich
SEGUS Inc.

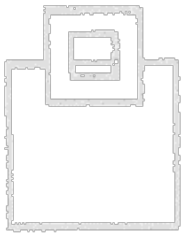
u.heinrich@segus.com

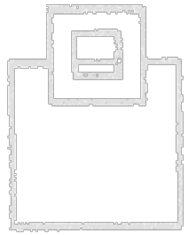


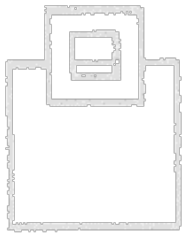
Agenda



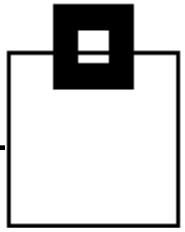
- General basics
  - Where/what are certificates used for?
- How is it used/realized?
- Real examples from the ZOWE ecosystem,
  - as well as z/OSMF, UMS, SQLDI, Db2
- Managing certificates in USS and z/OS
- Analyzing certificate issues



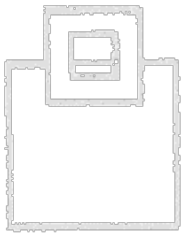




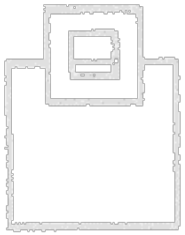
General Basics



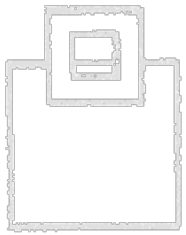
- Digital certificates, or public key certificates, or identity certificates are used to **identify** and **validate** an unknown origin and to **communicate securely** with it



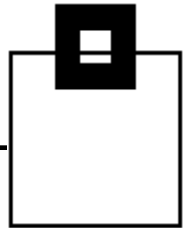
1. It includes information about the owner/subject,
   plus typically a certificate of the entity/issuer
   that has verified the owner/subject



2. It includes a public key that allows asymmetric, one-way encryption
   - The public key is intended to be shared
     - A Public key enables anybody to encrypt content
     - Only the corresponding private key of a public/private key pair can decrypt the content



General Basics

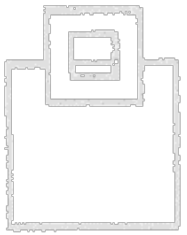


Conclusion:

→ A certificate is an electronic document used to

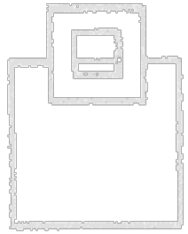
1. prove an identity and
2. to provide a key

which is part of the document

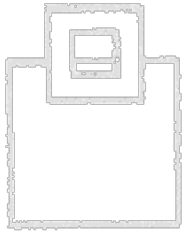


→ Once a certificate is verified to be trustworthy the validity proves

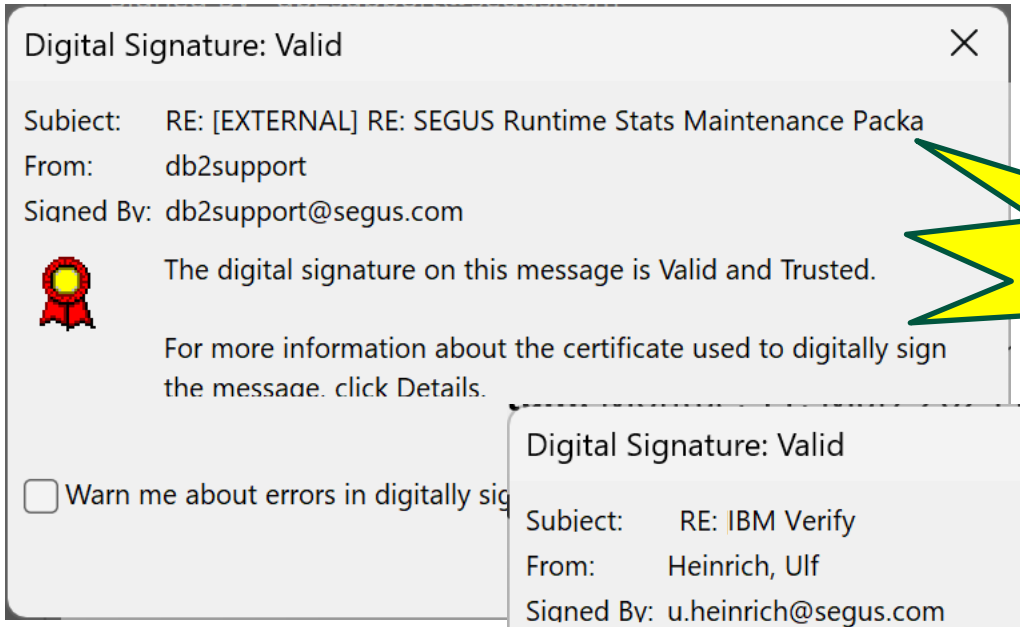
- sender/integrity of an e-mail (S/MIME)
- authenticity of a payment card for transactions (EMV)
- owner/integrity/genuine of apps/binaries (code signing)
- Document, eID, role, …
- device (domain/host/IP) (TLS/SSL)



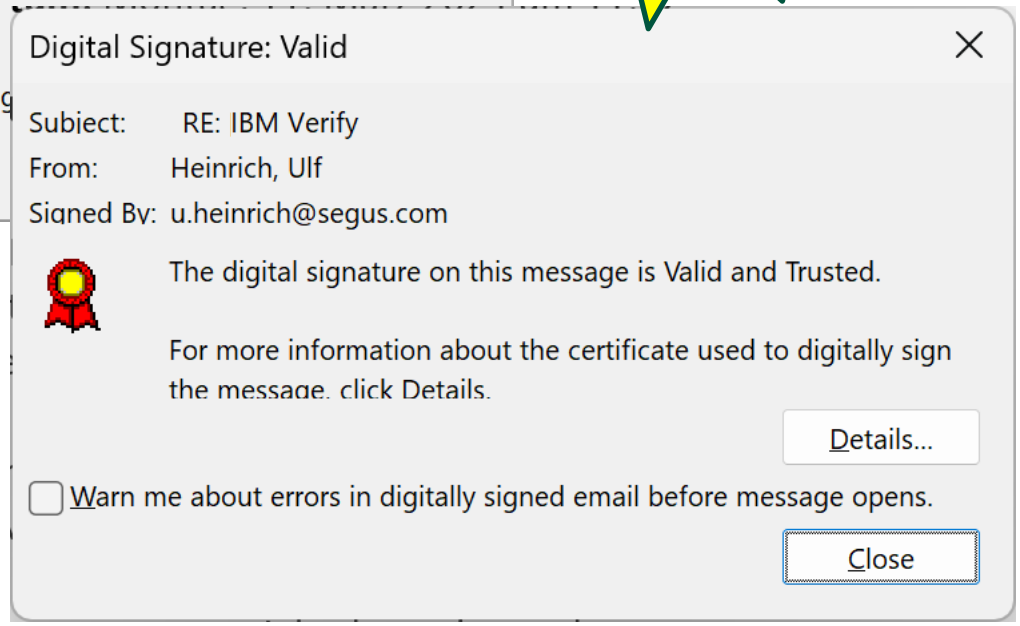
- Further, the public key can be used for secure communication with a
  - Person, or organization (e.g. e-mail, messaging)
  - Device (https, ftps, sftp, ssh, VPN, RDP…)



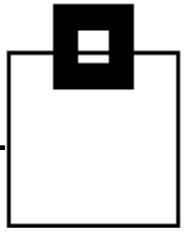
Where/what are certificates used for?



e-mails



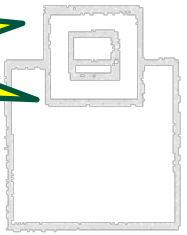
Where/what are certificates used for?

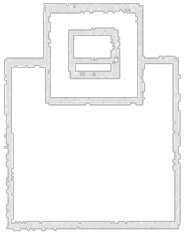


Adobe Acrobat Reader Installer

Verified publisher: Adobe Inc.
File origin: Downloaded from the Internet

Show more details

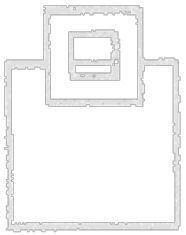




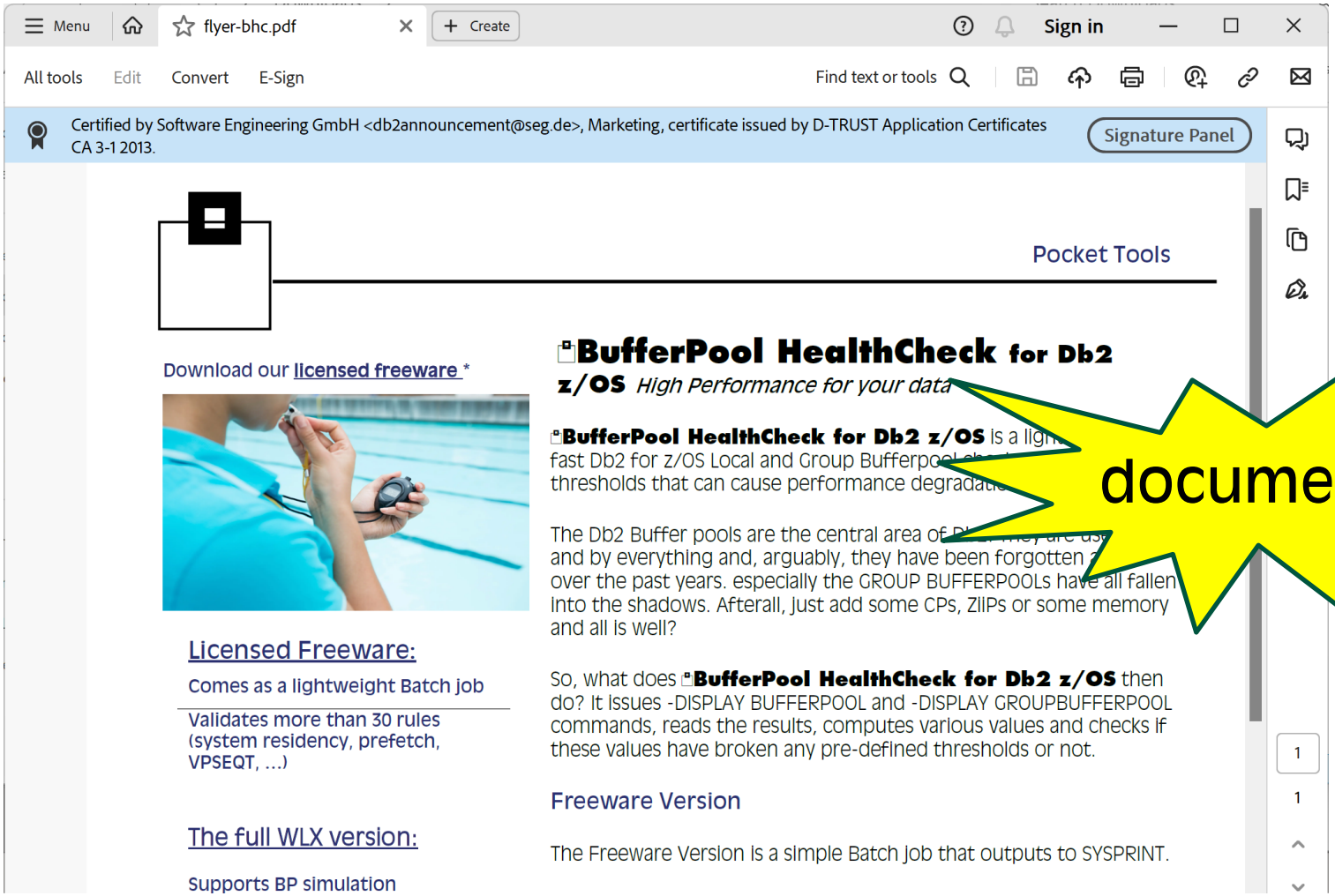
Program signing and verification

This chapter provides information about enabling users to digitally sign programs and enabling RACF® to verify signed programs.

This chapter also provides instructions for enabling RACF support for Validated Boot for z/OS. Here, you must perform some set-up activities before using Validated Boot for z/OS to sign IPL data. The term *IPL data* includes IPL text and system load modules, such as the system residence volume (SYSRES) contents.With Validated Boot for z/OS, your installation can ensure that its IPL data is intact, untampered-with, and originates from a trusted build-time source. Information about RACF support for Validated Boot for z/OS is provided in IPL data signing for Validated Boot for z/OS.

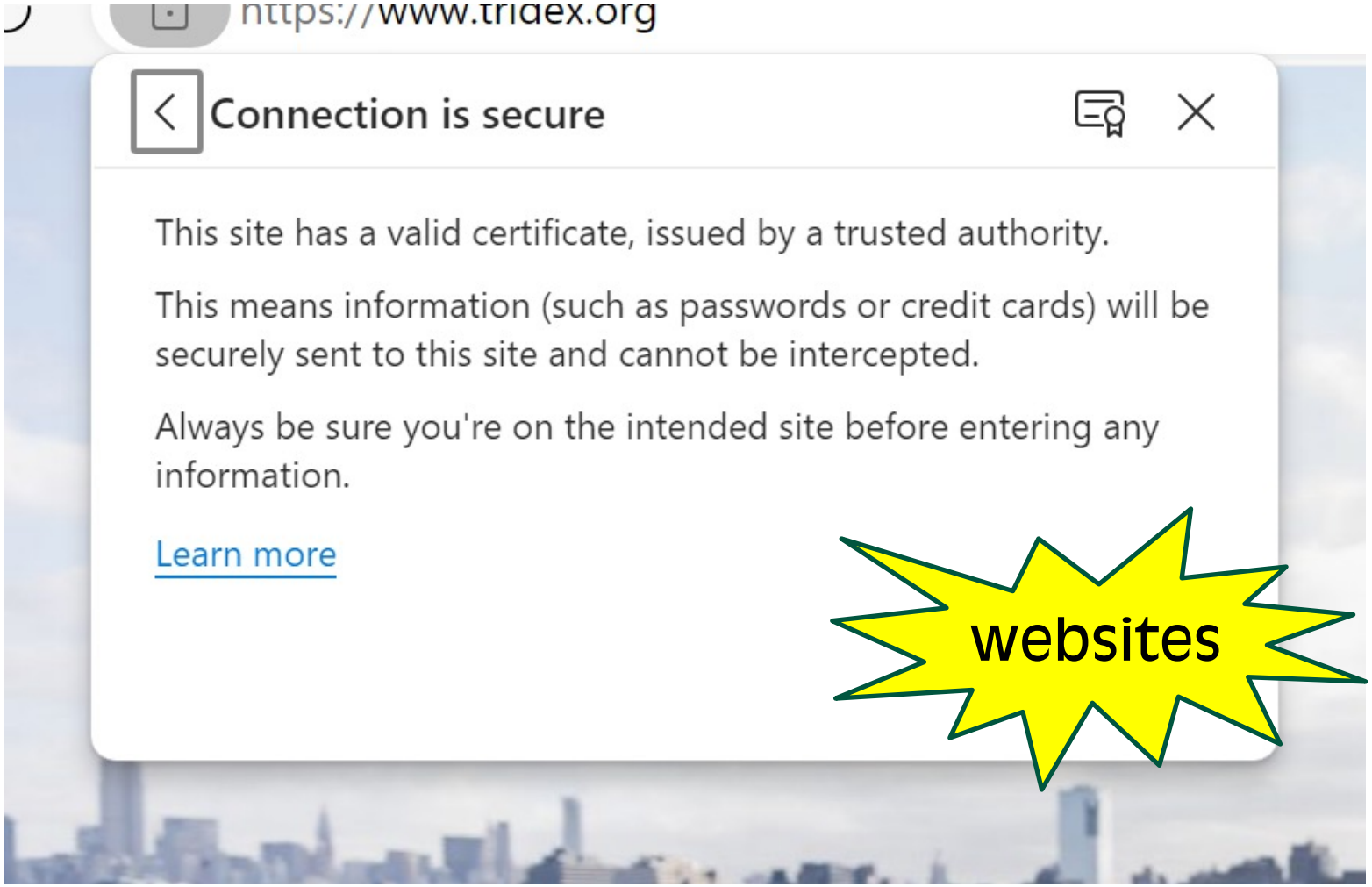


Where/what are certificates used for?



documents

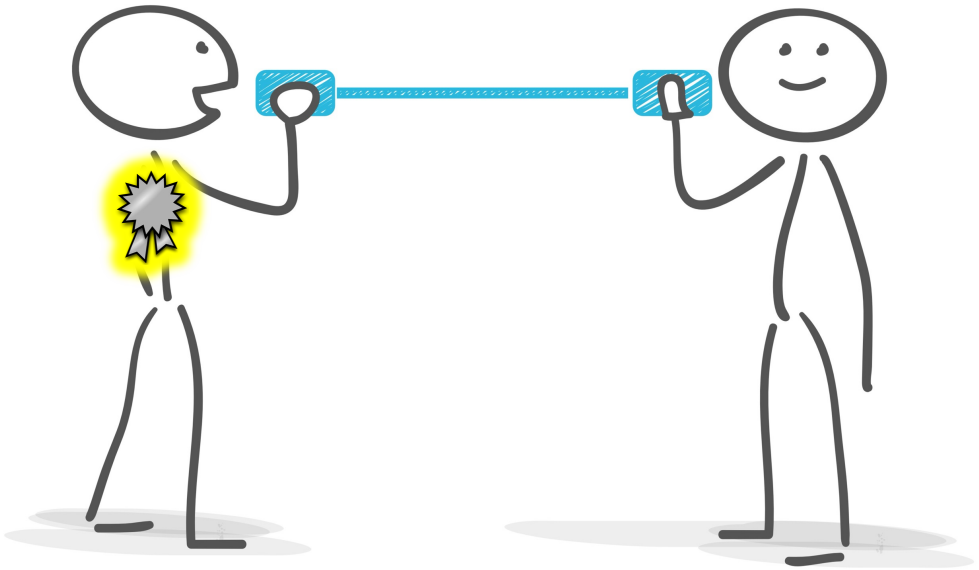
Where/what are certificates used for?



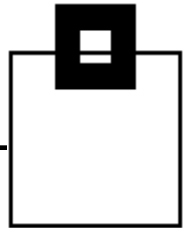
Where/what are certificates used for?

The technology is always the same, but today we focus on secure client – server communication:

1. Assure that a subject is really the one it supposes to be.
2. Assure that the information exchanged isn‘t manipulated.
3. Assure that the communication is treated confidentially.

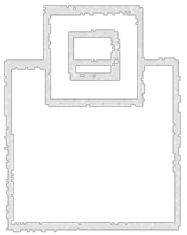


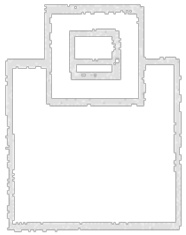
How is it used/realized?

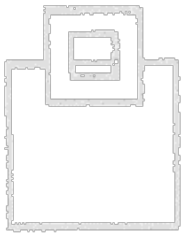


Let's have a closer look at secure client – server communication:

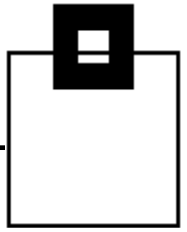
- A standardized process,...
  - 1987 Secure Data Network System (SDNS) project initiated
  - 1996 using SSL 3.0 under governance of the IETF to develop internet-standards
  - since 1999 continuously enhanced as transport layer security (TLS)
- ... that anybody understand/supports
  - Any current client (e.g. browser, desktop, smartphone) and server (e.g. mail, web, database) supports secure communication via the X.509 based mechanisms
    - TLS handshake
    - TLS record





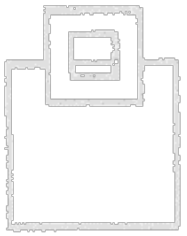


How is it used/realized?

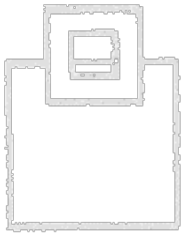


Secure client – server communication starts with a secure connection request, (e.g. **https**, **ftps**, …) and often requires to specify a secure port:

https://s0w1.dus.seg.de:**10443**/zosmf



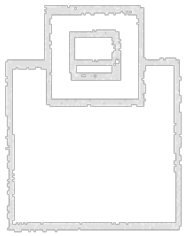
1. Connection request from a client to a server
2. Server replies with its certificate
3. Verification of the replying server and its trustworthiness by the client
4. Connection dependent handshake of the encryption between client and server



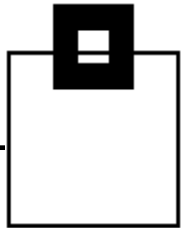
Optionally: Certificate authentication of the client

Verification of the client by the server

5. Start encrypted communication

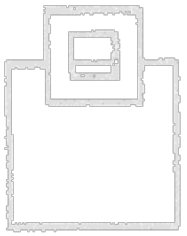


How is it used/realized?

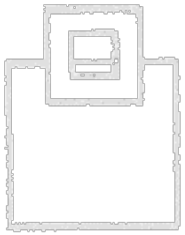


After we've received the certificate (including a key) from a server how is the information verified to guarantee its identity?

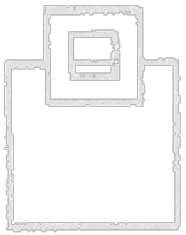
- **A certificate alone does not guarantee the identity shown, nor its trustworthiness!**
  - An identity can only be proved by a trusted entity
  - Trustworthiness can only be judged by the communication partner



- So, how can a client know if the communication partner is safe and trustworthy?
  1. Either the provided certificate is individually categorized trustworthy,
  2. or a superior certificate authority (CA) is trusted that confirms the identity shown (certificate chain)



This is the major concept used throughout X.509-based TLS.



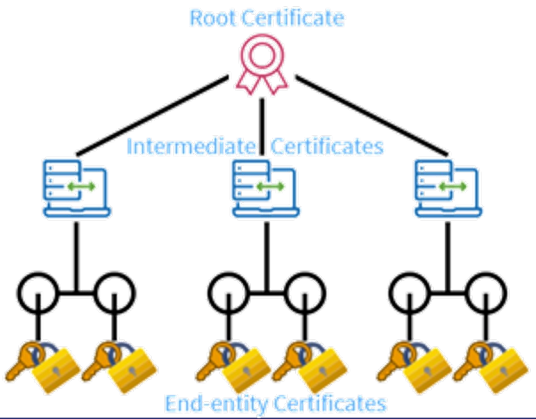
How is it used/realized?

Who is a *superior certificate authority* (CA)?

- Higher instance in a certificate chain of trust (intermediate, or root)
  - Reputable, commonly trusted organizations*
    - May assign limited duties to external identity authorities
  - Companies usually have an “internal” CA to simplify certificate management
- Validates the content of a certificate (signing request - CSR) and can issue/revoke certificates inheriting trustworthiness

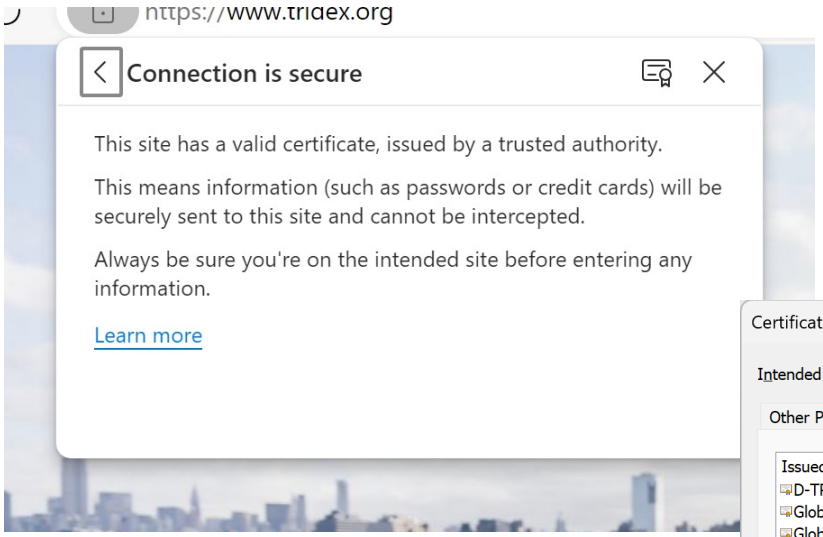
  → **Certificates signed by a trusted CA are automatically trusted!**

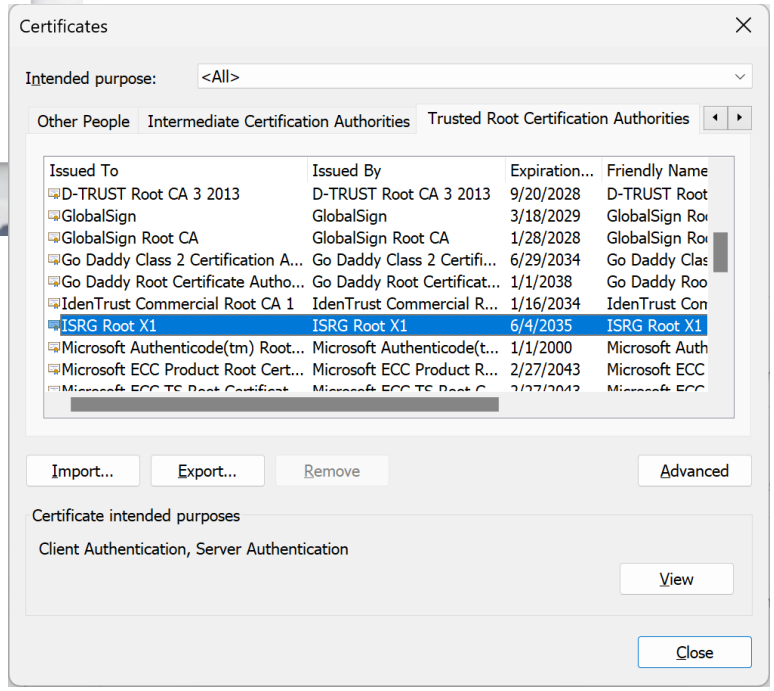
*The Certification Authority Browser Forum (CA/Browser Forum) is a voluntary gathering of certificate Issuers and suppliers of internet browser software and other applications that use certificates.



How is it used/realized?

Who is a *superior certificate authority* (CA)?

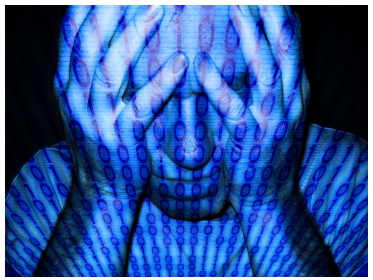




How is it used/realized?

Besides the verification of an identity we want to initiate the secure connection, but

- Client and server may not know each others yet
- Communicating securely requires that both parties are able to encrypt and to decrypt the information sent/received



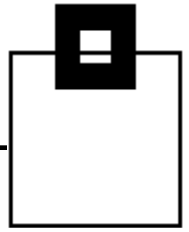
BUT:

- Without a common (symmetric) encryption key, no encryption!
- If they'd negotiate a key to start encryption, it would need to be unencrypted and someone else on the network could use a network sniffer, steal the key and compromise the encryption

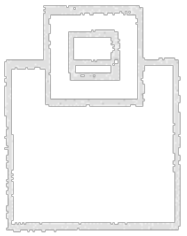
The solution:

→ Client and server negotiate the symmetric encryption key using asymmetric encryption

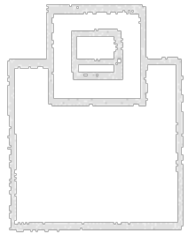
How is it used/realized?



- TLS encryption is based on X.509 certificates that identify the owner and provide the public key from a public/private key pair
- The public key coming with this certificate can be used to initiate asymmetric encrypted communication
  - Therefore, the *public* key provided along with the certificate at connection request is used by the recipient to check integrity and create and return an encrypted pre-master-key
  - The encrypted pre-master-key can only be decrypted with the appropriate *private* key, which is then used for the further encryption

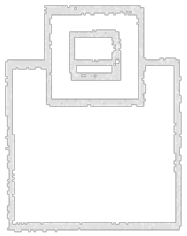


  → Public key can encrypt, but only private key can decrypt (asymmetric encryption)



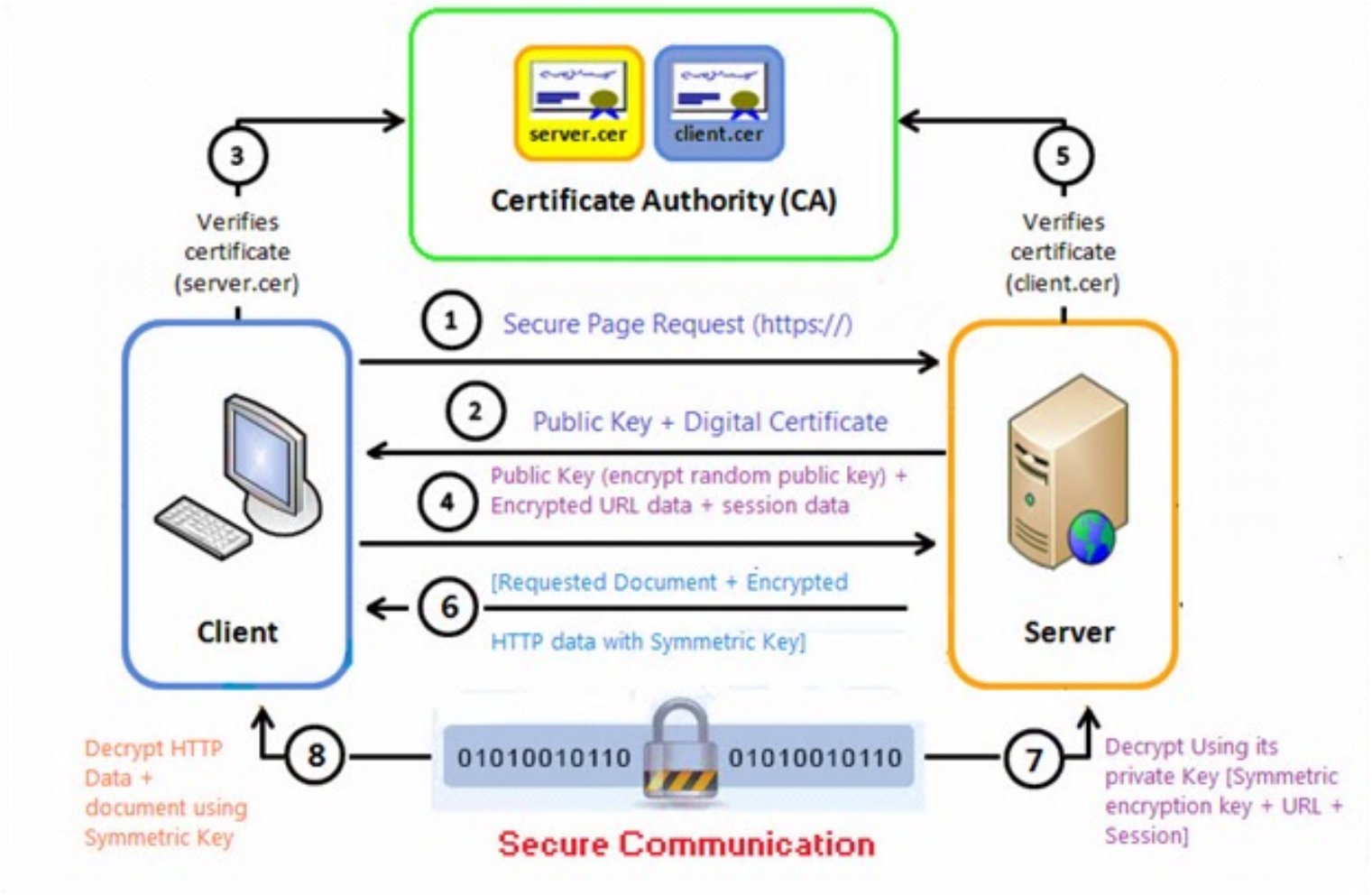
- Due to the fact that the private key should <u>never ever</u> be accessible by someone else but the owner, certificates are typically generated manually by the owner, or as part of an installation by the owner (like ZOWE does):
  - E.g.:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem
out cert.pem OR certsigreq.csr -days 365
```

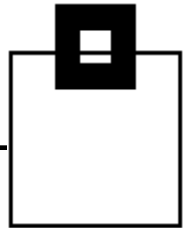


How is it used/realized?

TLS overview:

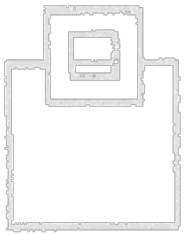


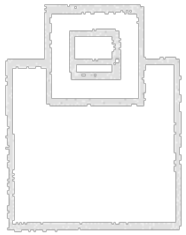
How is it used/realized?

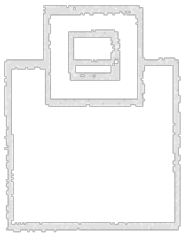


z/OSMF, ZOWE and Db2 work exactly this way:

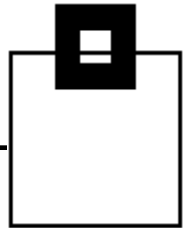
1. Connection request against z/OSMF, ZOWE, Db2 (secure port!)
2. Reply by z/OSMF, ZOWE, Db2 with its certificate (incl. certificate chain with a certificate authority if applicable)
3. Trustworthiness verification of the certificate, resp. of the root/intermediate certificate authority
4. Generation and return of the pre-master-key by the client using the servers public key
5. Generation of the encryption of an individual connection and start of the encrypted communication
   - Manipulation can be detected by an individual message authentication code



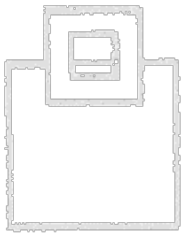




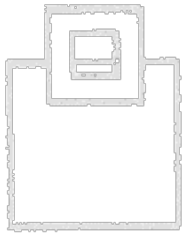
Real examples from the ZOWE ecosystem



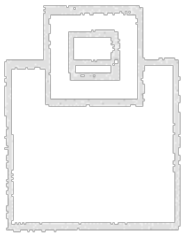
- The standardized certificate based on TLS is used
- Certificates are managed either in a KEYSTORE/TRUSTSTORE, or…
  - https://docs.zowe.org/stable/user-guide/configure-certificates-keystore
- by RACF KEYRINGs
  - https://docs.zowe.org/stable/user-guide/configure-certificates-keyring



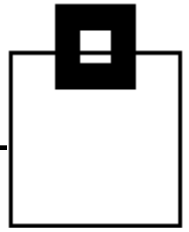
- More detailed information about certificate generation/management for application development extending ZOWE is available at
  - https://docs.zowe.org/stable/extend/extend-apiml/onboard-plain-java-enabler/#api-security



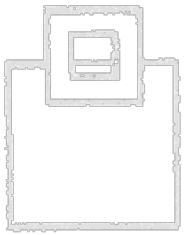
Reminder: It‘s all about trustworthiness!

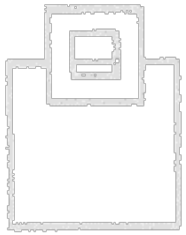


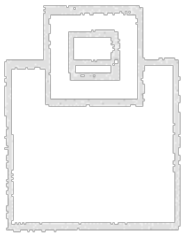
Real examples from the ZOWE ecosystem



- The certificate store is specified in the ZOWE configuration (zowe.yaml, formerly instance.env), as a java keystore/truststore, or…

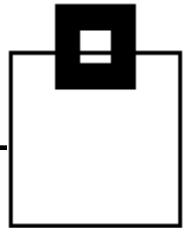






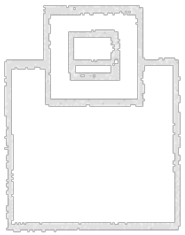
```
certificate:
  keystore:
    type: PKCS12
     file: /zowe/keystore/localhost/localhost.keystore.p12
     password: password
     alias: localhost
  truststore:
     type: PKCS12
     file: /zowe/keystore/localhost/localhost.truststore.p12
     password: password
pem:
     key: /zowe/keystore/localhost/localhost.key
     certificate: /zowe/keystore/localhost/localhost.cer
     certificateAuthorities: /zowe/keystore/local_ca/local_ca.cer
verifyCertificates: STRICT
```

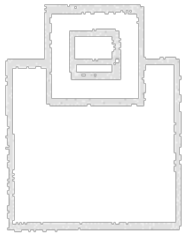
Real examples from the ZOWE ecosystem

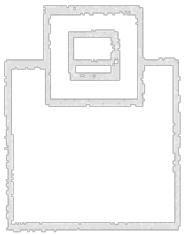


- ... as a RACF keyring

```
certificate:
  keystore:
    type: "JCERACFKS"
    file: "safkeyring:////ZWESVUSR/ZOWEKEYS"
    password: "password"
    alias: "ZWESRV"
  truststore:
    type: "JCERACFKS"
    file: "safkeyring:////ZWESVUSR/ZOWEKEYS"
    password: "password"
  pem:
    key: ""
    certificate: ""
    certificateAuthorities:
"safkeyring:////ZWESVUSR/ZOWEKEYS&SEGROOTCA"
verifyCertificates: "STRICT"
```

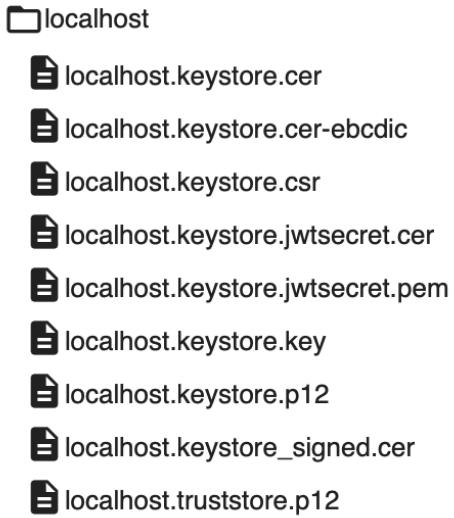






Real examples from the ZOWE ecosystem

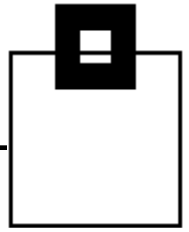
- KEYSTORE:
  - Stores its own certificate
- TRUSTSTORE
  - Stores trusted certificates



- RACF KEYRING
  - Stores both

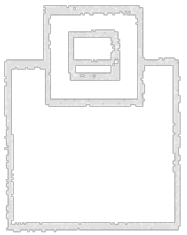
```
Ring:
ZOWEKEYS

  Certificate Label Name            Cert Owner    USAGE      DEFAULT
  ------------------------------    ----------    --------   --------
  SEGROOTCA                         CERTAUTH      CERTAUTH   NO
  ZWESRV                            ID(ZWESVUSR)  PERSONAL   YES
```

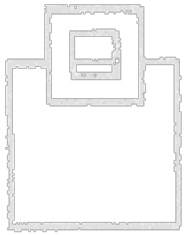
Real examples from UMS and z/OSMF



- IBM Unified Management Server uses ZOWE's keystore/truststore/keyring by default, unless you specify something else in UMS's parmlib member

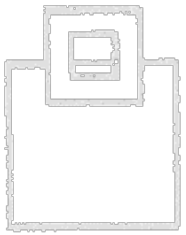
```
certificate:
        allowSelfSigned: true
        truststore:
          location: "safkeyring:////ZWESVUSR/IZPRING"
          type: "JCERACFKS"
        keystore:
          location: "safkeyring:////ZWESVUSR/IZPRING"
          type: "JCERACFKS"
          alias: "UMSSRV"
```



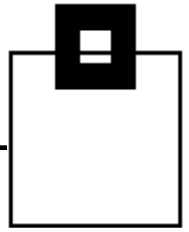


- For z/OSMF you can specify the RACF keyring in the IZU PARMLIB member

```
(…)
KEYRING_NAME('ZOSMFKEYS')
(…)
```

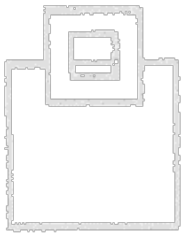


Real examples from SQLDI and Db2



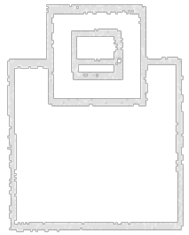
- For SQL Data Insights you are prompted to specify the RACF keyring when running the installation script sqldi.sh

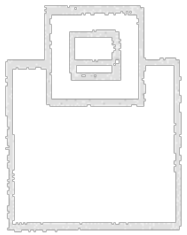
```
Enter your keystore information > SQLDIID.SQLDIKEYRING
```



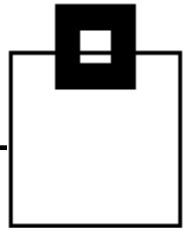
- For Db2 you have to configure the TLS setup via PAGENT

```
TTLSRule DD10SecureServer
{ LocalPortRange 15151
  JobName DD10DIST
  Direction Inbound
  TTLSGroupActionRef DD10SecureGrpAct
  TTLSEnvironmentActionRef DD10SecureEnvAct
  TTLSConnectionActionRef  DD10SvrAuthConn
}
TTLSGroupAction DD10SecureGrpAct
{ TTLSEnabled On
  Trace 15
}
TTLSEnvironmentAction DD10SecureEnvAct
{ TTLSKeyRingParms
  {    Keyring SEGDB2KEYRING
  }
  (...)
```





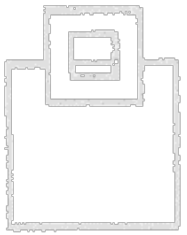
Managing certificates in USS and z/OS

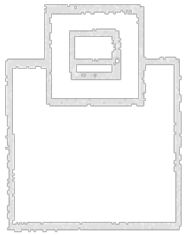


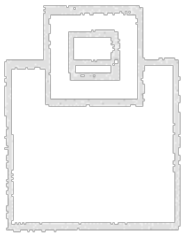
How to manage keystores, truststores, keyrings?

- A keystore/truststore can be managed using the keytool

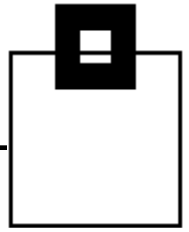
```
>keytool
Key and Certificate Management Tool
Commands:
 -certreq            Generates a certificate request
 -changealias        Changes an entry's alias
 -delete             Deletes an entry
 -exportcert         Exports certificate
 -exportseckey       Export a batch of secret keys
 -genkeypair         Generates a key pair
 -genseckey          Generates a secret key
 -gencert            Generates certificate from a certificate request
 -importcert         Imports a certificate or a certificate chain
 -importpass         Imports a password
 -importkeystore     Imports one or all entries from another keystore
 -importseckey       Import a batch of secret keys
 -keypasswd          Changes the key password of an entry
 -list               Lists entries in a keystore
 -printcert          Prints the content of a certificate
 -printcertreq       Prints the content of a certificate request
 -printcrl           Prints the content of a CRL file
 -storepasswd        Changes the store password of a keystore
```







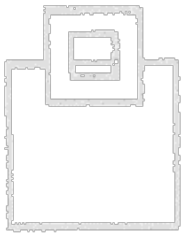
Managing certificates in USS and z/OS

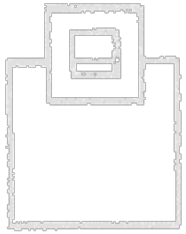


How to manage keystores, truststores, keyrings?

- A keyring can be managed using RACF
  - Services option menu

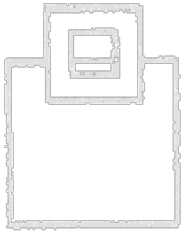
```
                    RACF - SERVICES OPTION MENU
OPTION ===>
SELECT ONE OF THE FOLLOWING:
   1  DATA SET PROFILES
   2  GENERAL RESOURCE PROFILES
   3  GROUP PROFILES AND USER-TO-GROUP CONNECTIONS
   4  USER PROFILES AND YOUR OWN PASSWORD
   5  SYSTEM OPTIONS
   6  REMOTE SHARING FACILITY
   7  DIGITAL CERTIFICATES, KEY RINGS, AND TOKENS
  99  EXIT
```



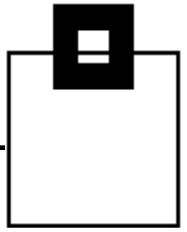


  - RACDCERT (Manage RACF digital certificates)

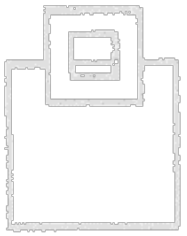
    "Use the RACDCERT command to install and maintain digital certificates, key rings, and digital certificate mappings in RACF."



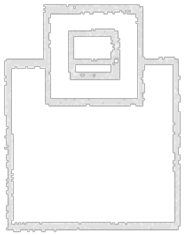
Managing certificates in USS and z/OS

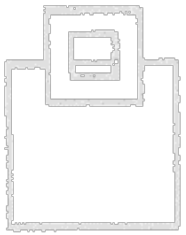


- Using KEYSTORE/TRUSTSTORE with self-signed certificates might be ok for testing,
  - 👍 Easy setup without additional RACF
  - 👍 Unix/USS OPENSSL and KEYTOOL usage as usual
  - 👎 Has to be trusted by the ZOWE user
  - 👎 No centralized certificate management

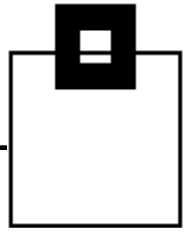


- but at the end, a RACF KEYRING with company CA-signed certificates is a better choice
  - 👍 Centralized z/OS/USS certificate management
  - 👍 Implicitly trusted for all employers
  - 👎 Requires RACDCERT knowledge and authorization
  - 👎 Some (Db2) require additional PAGENT definition



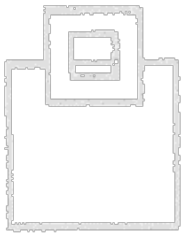


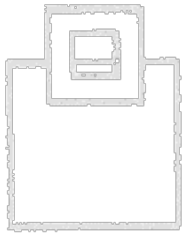
Managing certificates in USS and z/OS

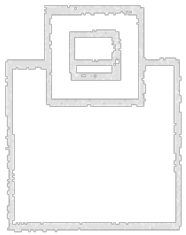


RACDCERT example of a certificate + company CA

1. Create a company CA to make any of your certificates trustworthy

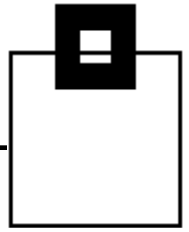






```
//GENCACRT EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DDNAME=RACF
//RACF     DD DATA,DLM=$$,SYMBOLS=JCLONLY
   RACDCERT GENCERT CERTAUTH +
            SUBJECTSDN( +
              CN('SOFTWARE ENGINEERING ROOT CA') +
              OU('DEVELOPMENT') +
              O('SOFTWARE ENGINEERING GMBH') +
              L('DUESSELDORF') +
              SP('NORTH RHINE WESTPHALIA') +
              C('DE')) +
            SIZE(2048) +
            NOTAFTER(DATE(2033-01-07)) +
            WITHLABEL('SEGROOTCA') +
            KEYUSAGE(CERTSIGN)
$$
```

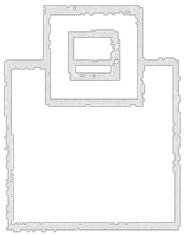
Managing certificates in USS and z/OS

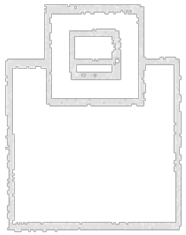


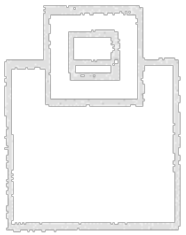
RACDCERT example of a certificate + company CA

2. Create a certificate signed with the CA created before

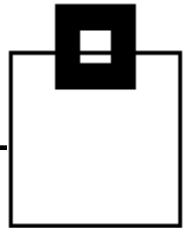
```
//GENSVCRT EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DDNAME=RACF
//RACF     DD DATA,DLM=$$,SYMBOLS=JCLONLY
   RACDCERT GENCERT ID(IZUSVR1) +
            SUBJECTSDN( +
              CN('ZOSMF MANAGEMENT SERVICE') +
              OU('DEVELOPMENT') +
              O('SOFTWARE ENGINEERING GMBH') +
              L('DUESSELDORF') +
              SP('NORTH RHINE WESTPHALIA') +
              C('DE')) +
            SIZE(2048) +
            NOTAFTER(DATE(2025-12-31)) +
            WITHLABEL('IZUSRV') +
            KEYUSAGE(HANDSHAKE) +
            ALTNAME(IP(192.168.9.98) +
                DOMAIN('S0W1.DUS.SEG.DE')) +
            SIGNWITH(CERTAUTH LABEL('SEGROOTCA'))
$$
```







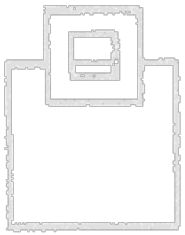
Managing certificates in USS and z/OS

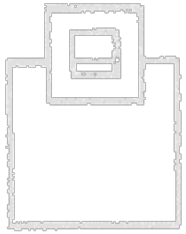


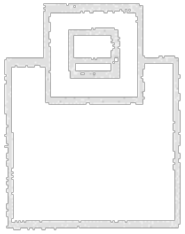
RACDCERT example of a certificate + company CA

3. Create a keyring for the certificates created

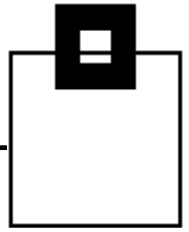
```
//GENSVCRT EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DDNAME=RACF
//RACF     DD DATA,DLM=$$,SYMBOLS=JCLONLY
   RACDCERT ADDRING(ZOSMFKEYS) ID(IZUSVR1)
   SETROPTS RACLIST(DIGTRING) REFRESH
$$
```







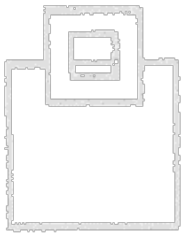
Managing certificates in USS and z/OS

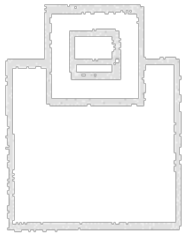


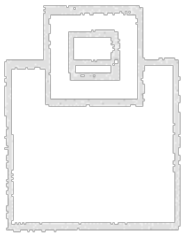
RACDCERT example of a certificate + company CA

4. Add the certificates created to the keyring created

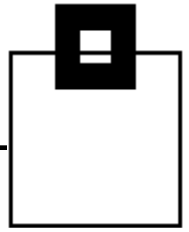
```
//GENSVCRT EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DDNAME=RACF
//RACF     DD DATA,DLM=$$,SYMBOLS=JCLONLY
   RACDCERT CONNECT(CERTAUTH LABEL('SEGROOTCA') +
            RING(ZOSMFKEYS)) +
            ID(IZUSVR1)
   RACDCERT CONNECT(ID(IZUSVR1) +
            LABEL('IZUSRV') +
            RING(ZOSMFKEYS) +
            USAGE(PERSONAL) DEFAULT) +
            ID(IZUSVR1)
$$
```





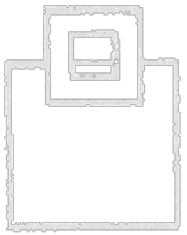


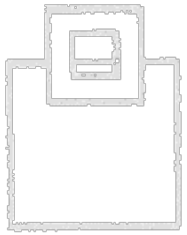
Managing certificates in USS and z/OS

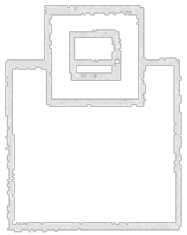


RACDCERT example of a certificate + company CA

5. Permit access to the keyring created



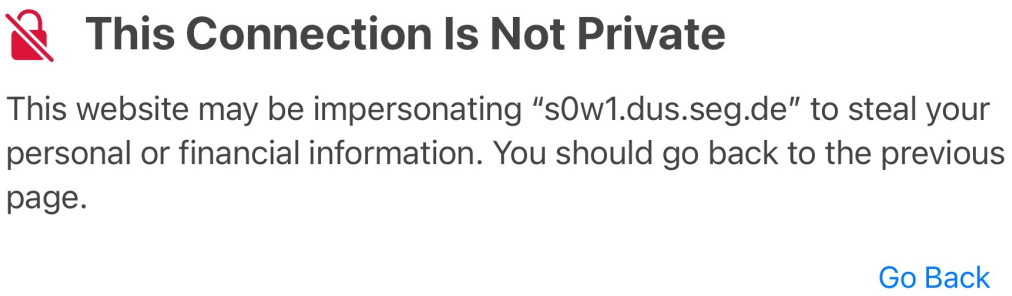




```
//GENSVCRT EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DDNAME=RACF
//RACF     DD DATA,DLM=$$,SYMBOLS=JCLONLY
   RDEFINE RDATALIB IZUSVR1.ZOSMFKEYS.LST UACC(NONE)
   PERMIT IZUSVR1.ZOSMFKEYS.LST CLASS(RDATALIB) ID(IZUSVR1) +
          ACCESS(CONTROL)
/* Uncomment this command to allow other user to access key ring ... */
/* PERMIT IZUSVR1.ZOSMFKEYS.LST CLASS(RDATALIB) ID(<USER>) +         */
/*        ACCESS(READ)                                               */
   SETROPTS RACLIST(RDATALIB) REFRESH

   PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(IZUSVR1) +
          ACCESS(READ)
   PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(IZUSVR1) +
          ACCESS(READ)
   SETROPTS RACLIST(FACILITY) REFRESH
$$
```

Analyzing certificate issues

Trustworthy or not, that's the question!

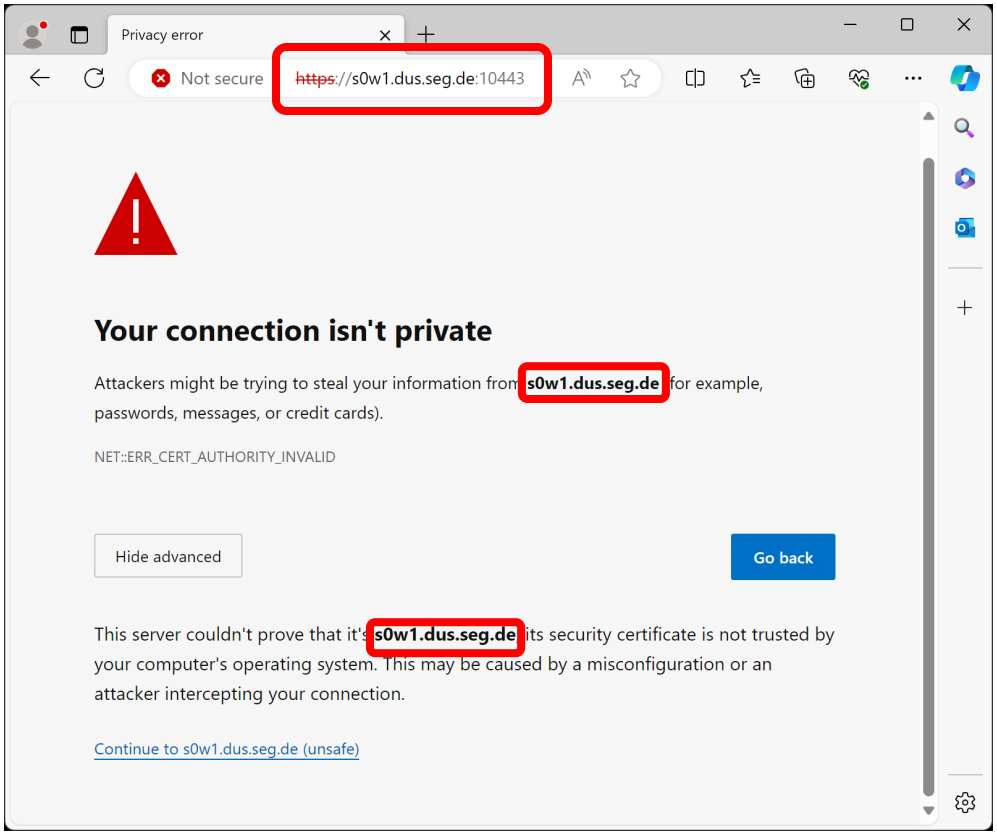


How to fix this???

Analyzing certificate issues

Trustworthy or not, that's the question!

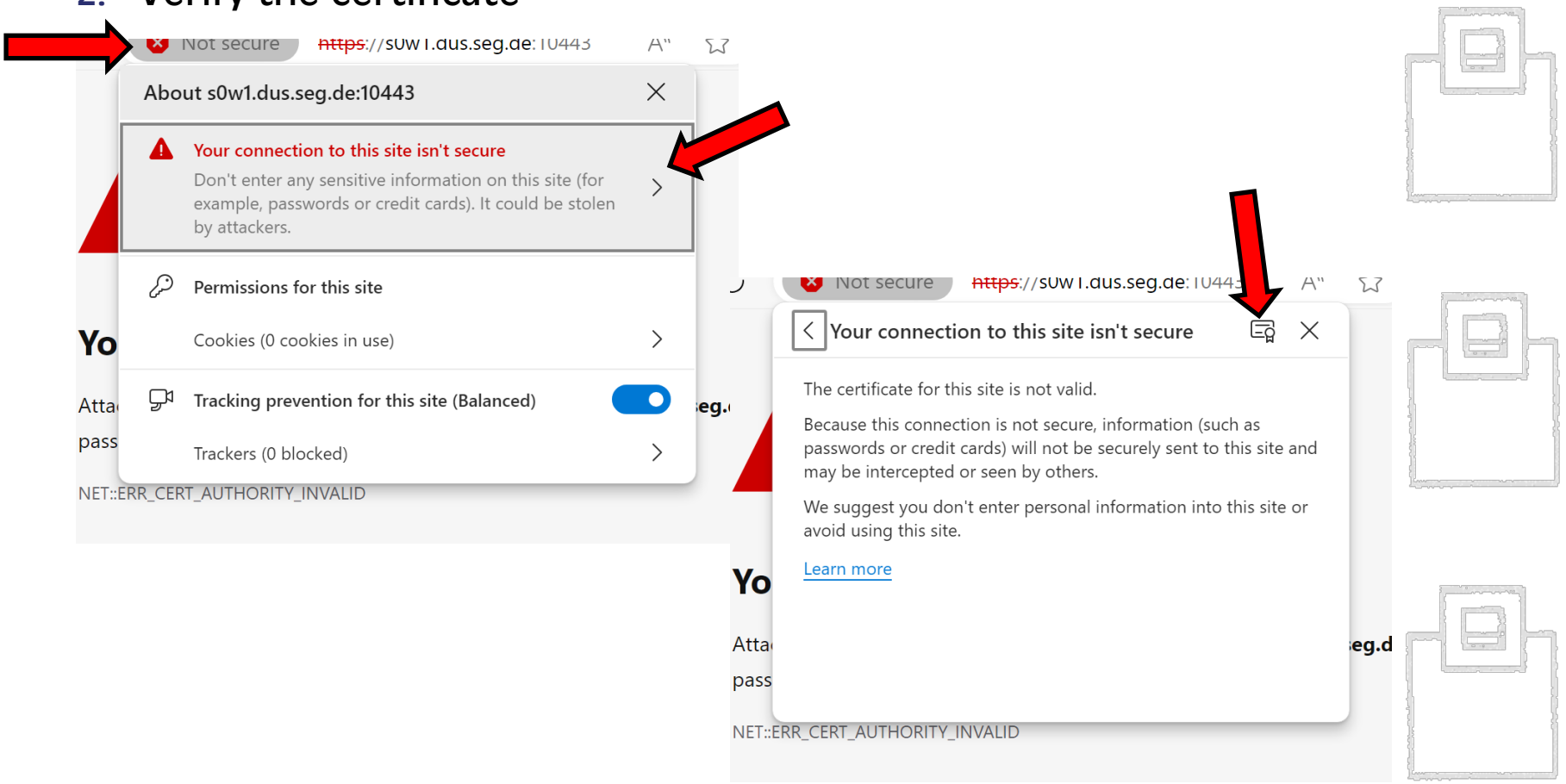
1. Make sure the host, or IP is correct!



Analyzing certificate issues

Trustworthy or not, that's the question!

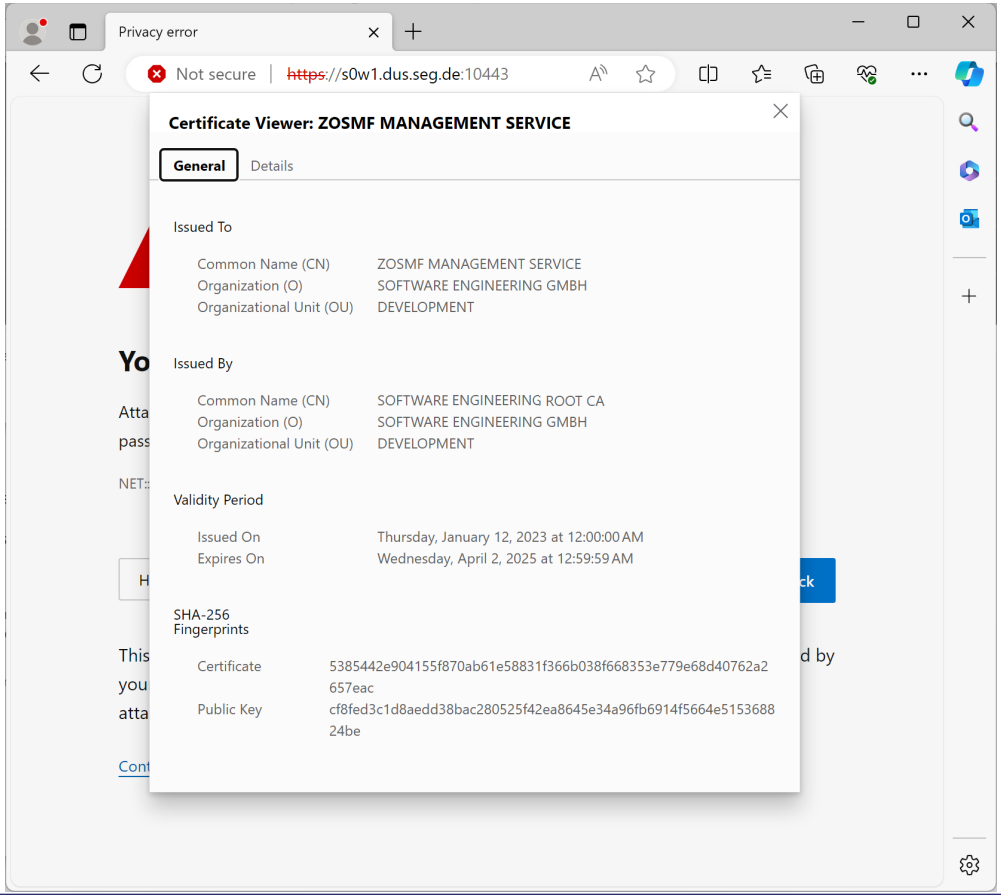
2. Verify the certificate



Analyzing certificate issues

Trustworthy or not, that's the question!

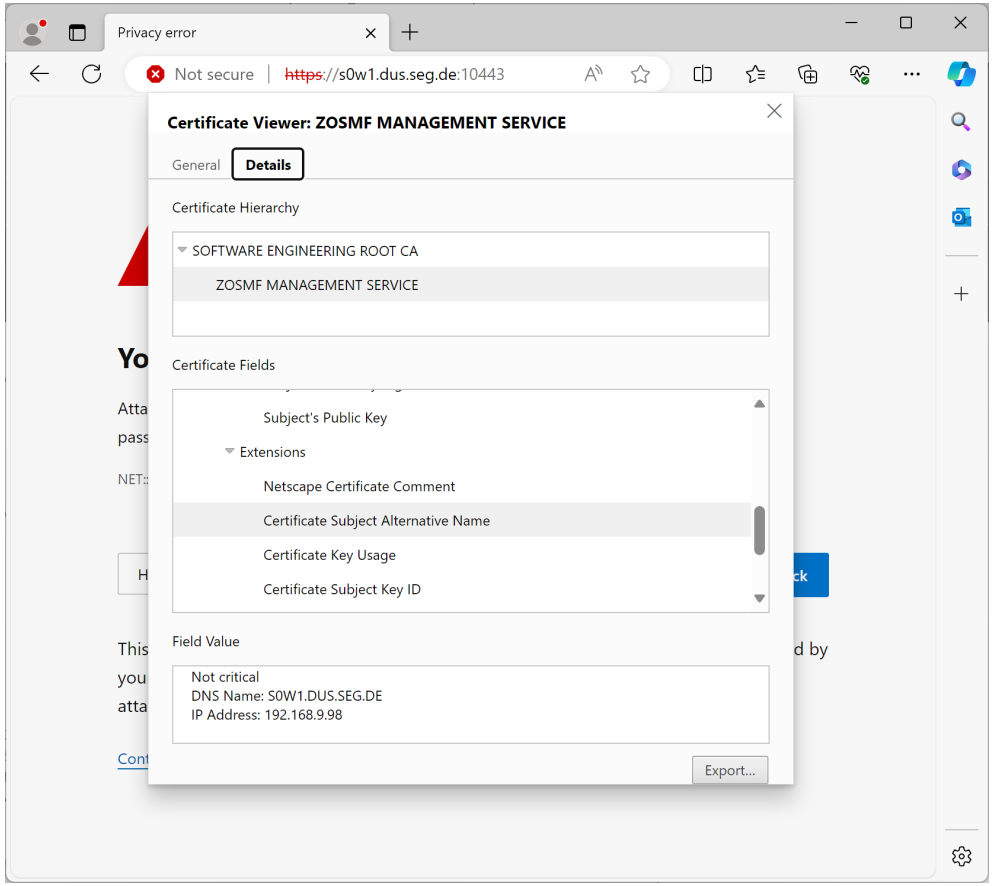
2. Verify the certificate's content



Analyzing certificate issues

Trustworthy or not, that's the question!

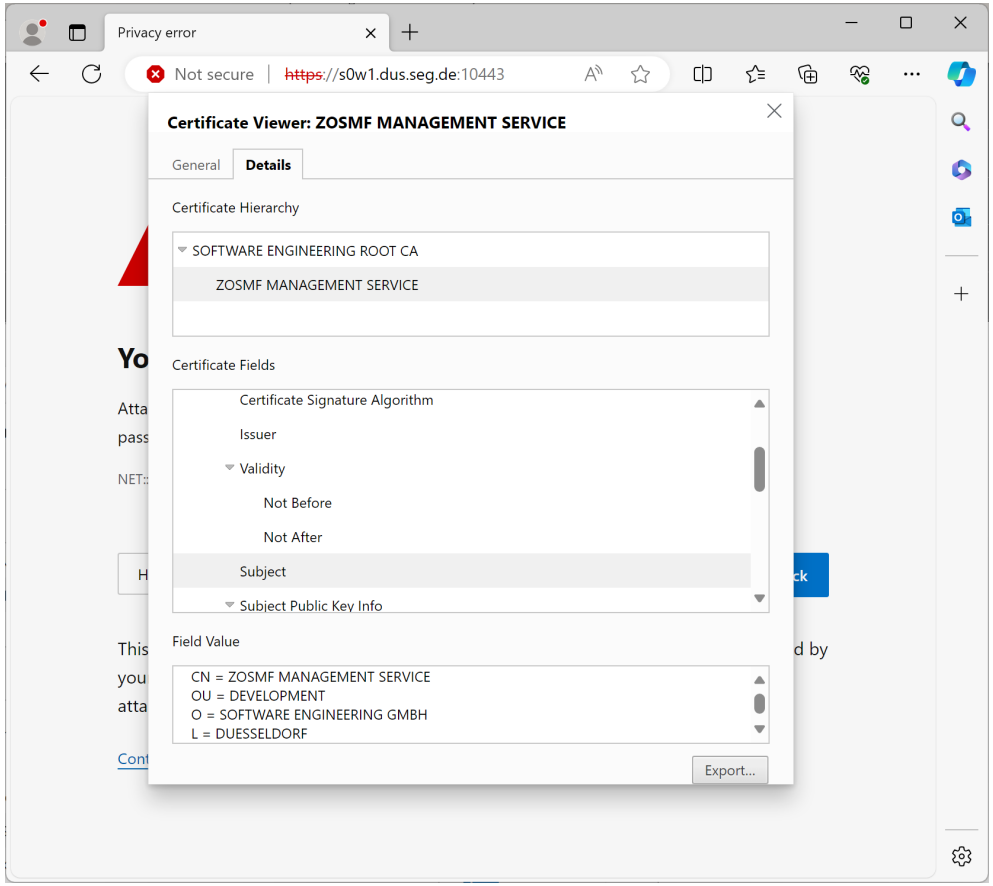
2. Verify the certificate's content



Analyzing certificate issues

Trustworthy or not, that's the question!

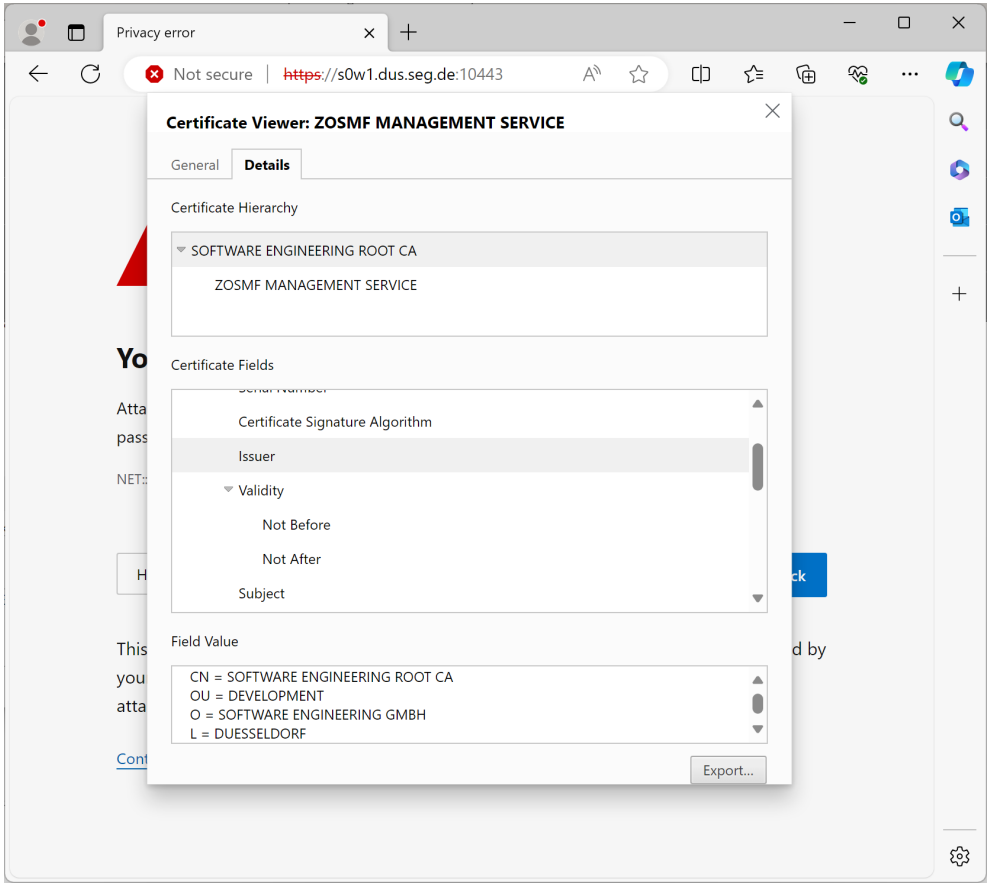
2. Verify the certificate's content



Analyzing certificate issues

Trustworthy or not, that's the question!

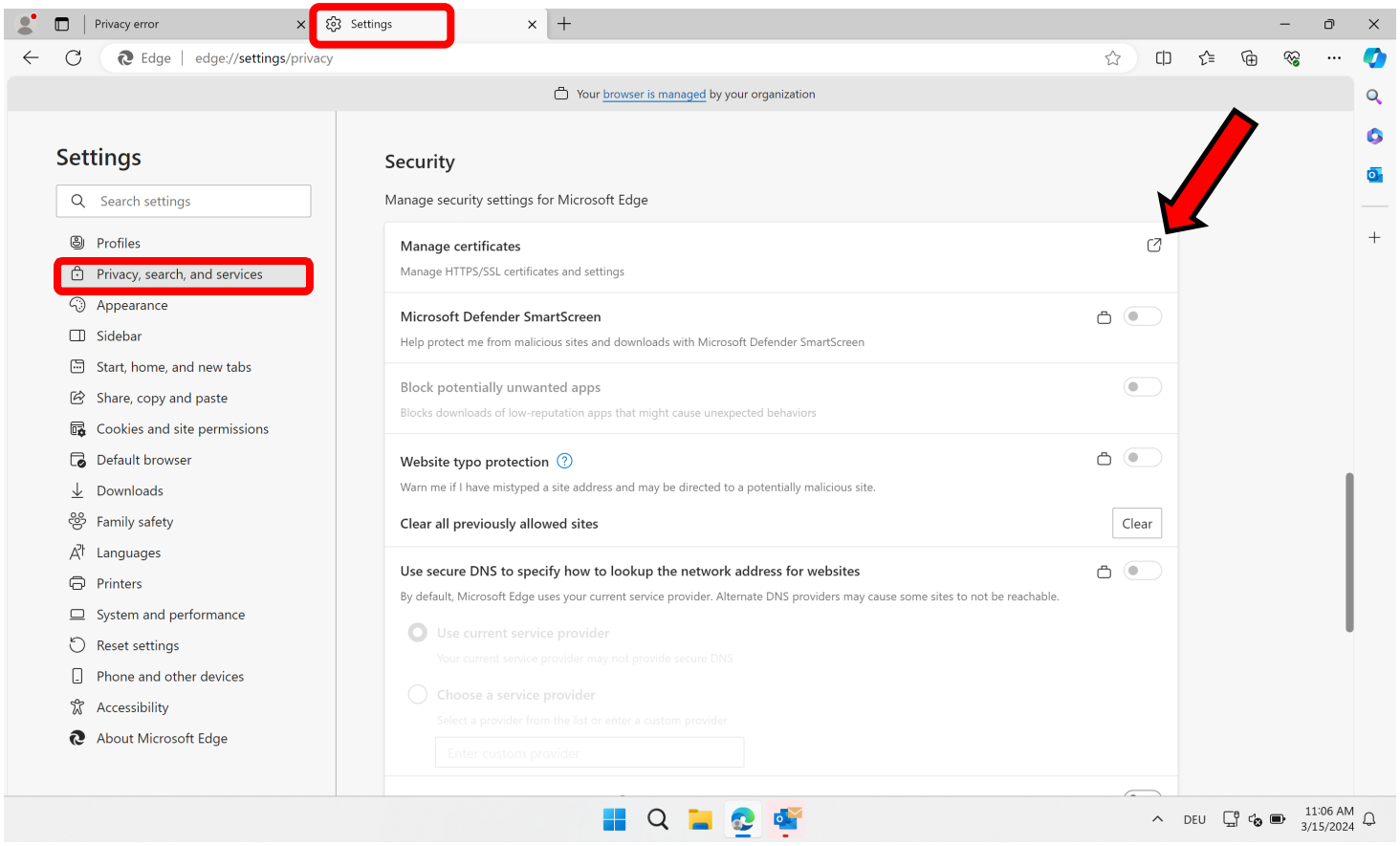
2. Verify the certificate's content



Analyzing certificate issues

Trustworthy or not, that's the question!

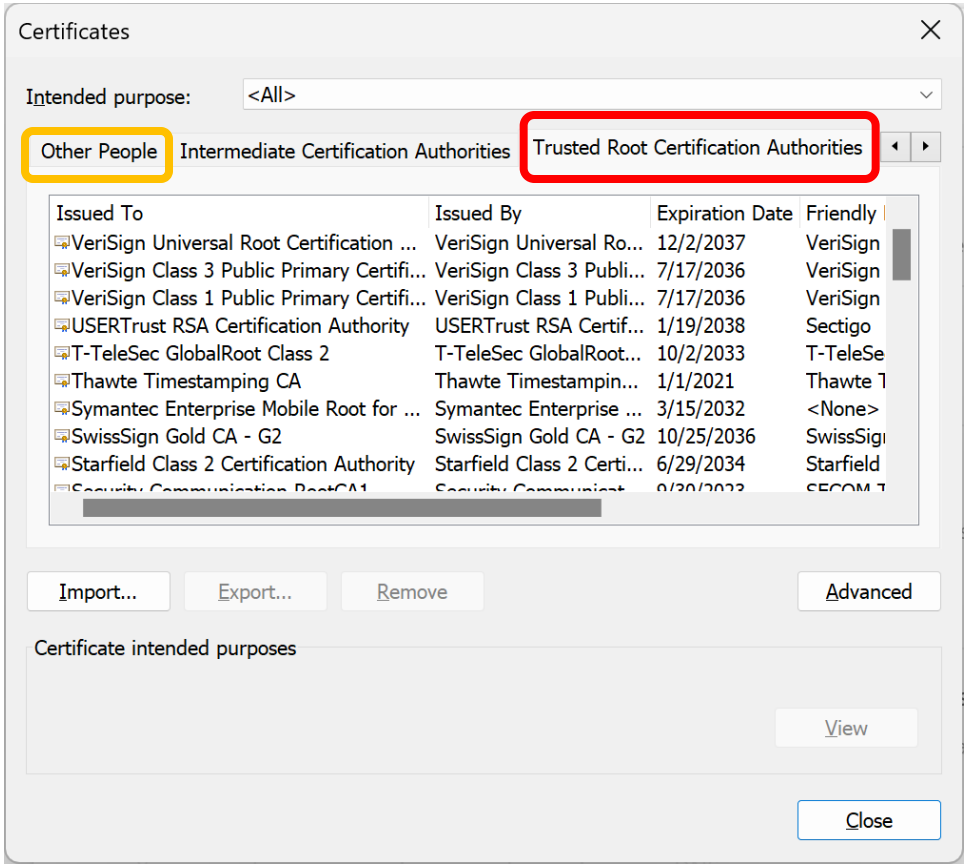
3. Verify that the CA (or the certificate) is trusted



Analyzing certificate issues

Trustworthy or not, that's the question!

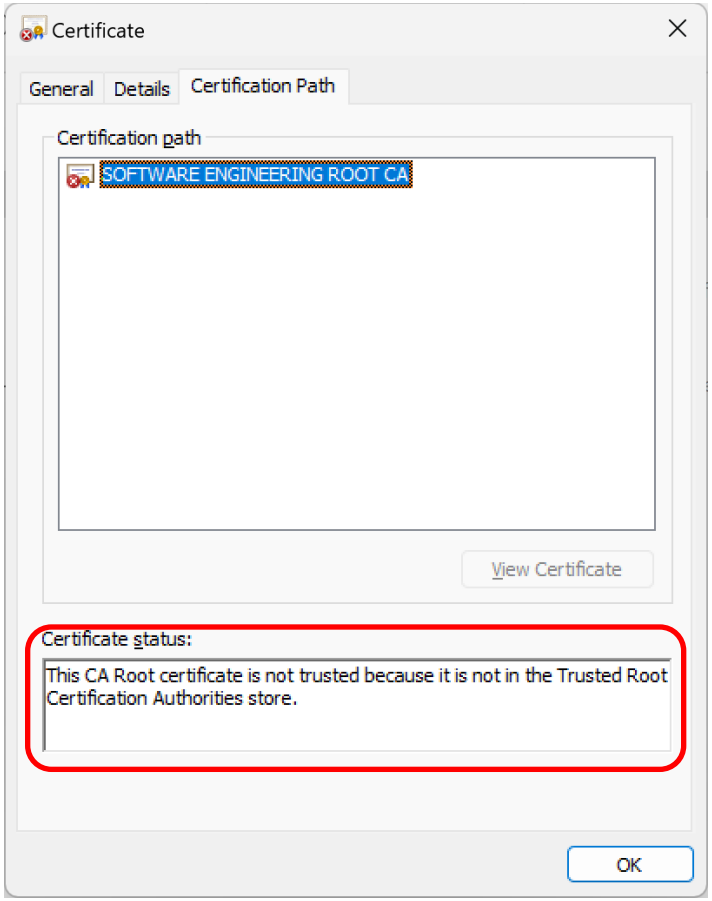
3. Verify that the CA (or the certificate) is trusted



Analyzing certificate issues

Trustworthy or not, that's the question!

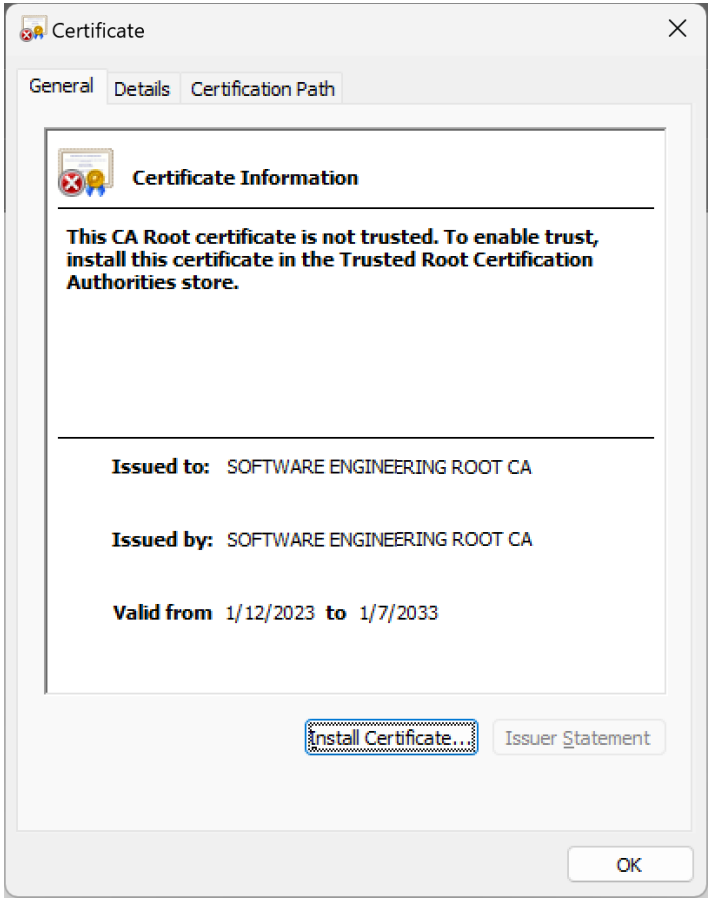
3. Verify that the CA (or the certificate) is trusted – add it, if missing



Analyzing certificate issues

Trustworthy or not, that's the question!

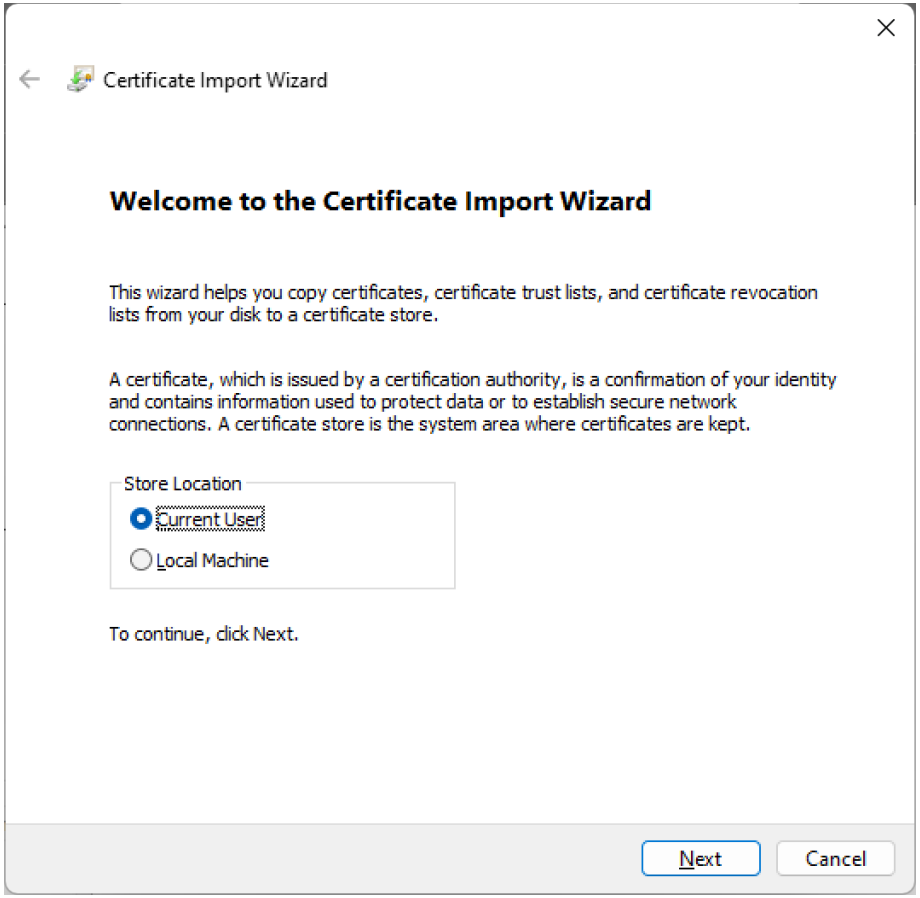
3. Verify that the CA (or the certificate) is trusted – add it, if missing



Analyzing certificate issues

Trustworthy or not, that's the question!

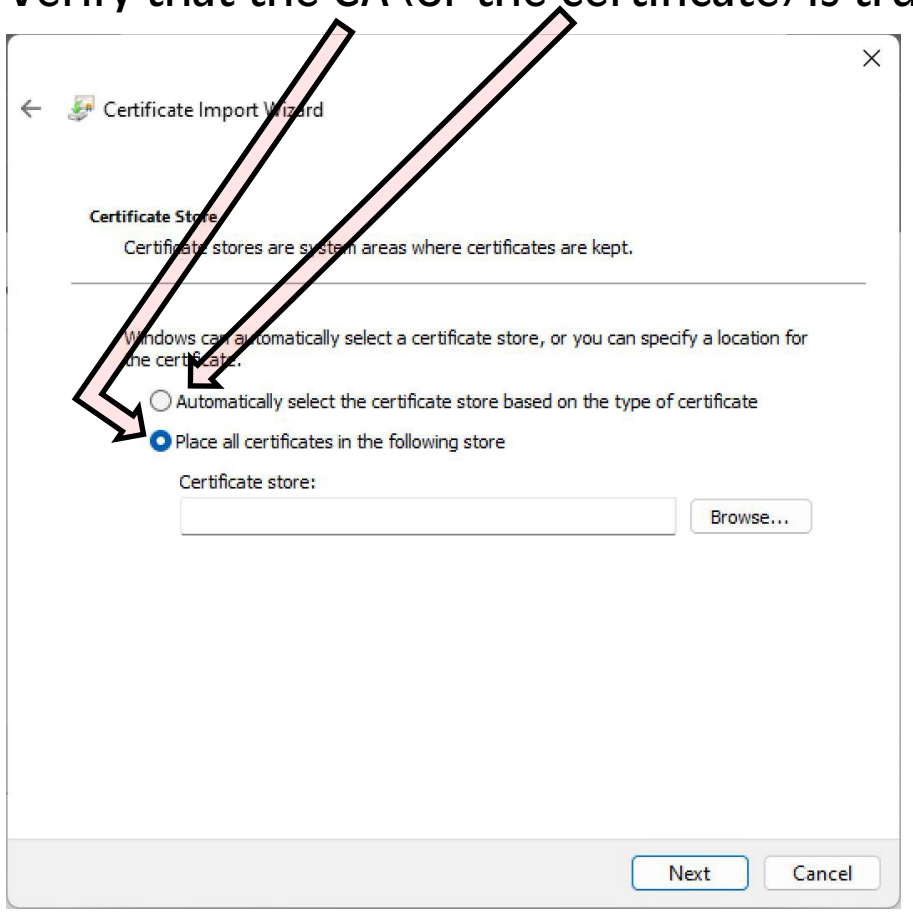
3. Verify that the CA (or the certificate) is trusted – add it, if missing



Analyzing certificate issues

Trustworthy or not, that's the question!

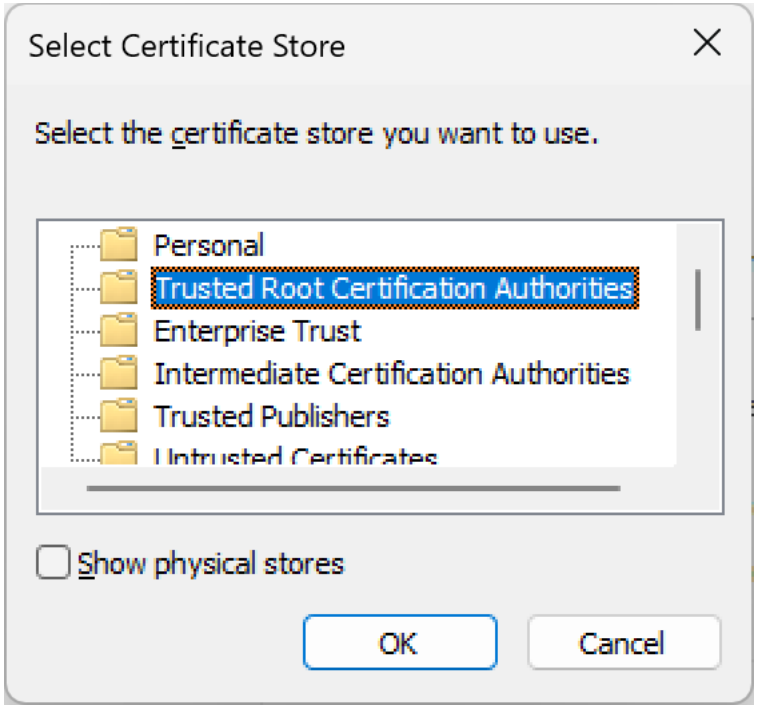
3. Verify that the CA (or the certificate) is trusted – add it, if missing



Analyzing certificate issues

Trustworthy or not, that's the question!

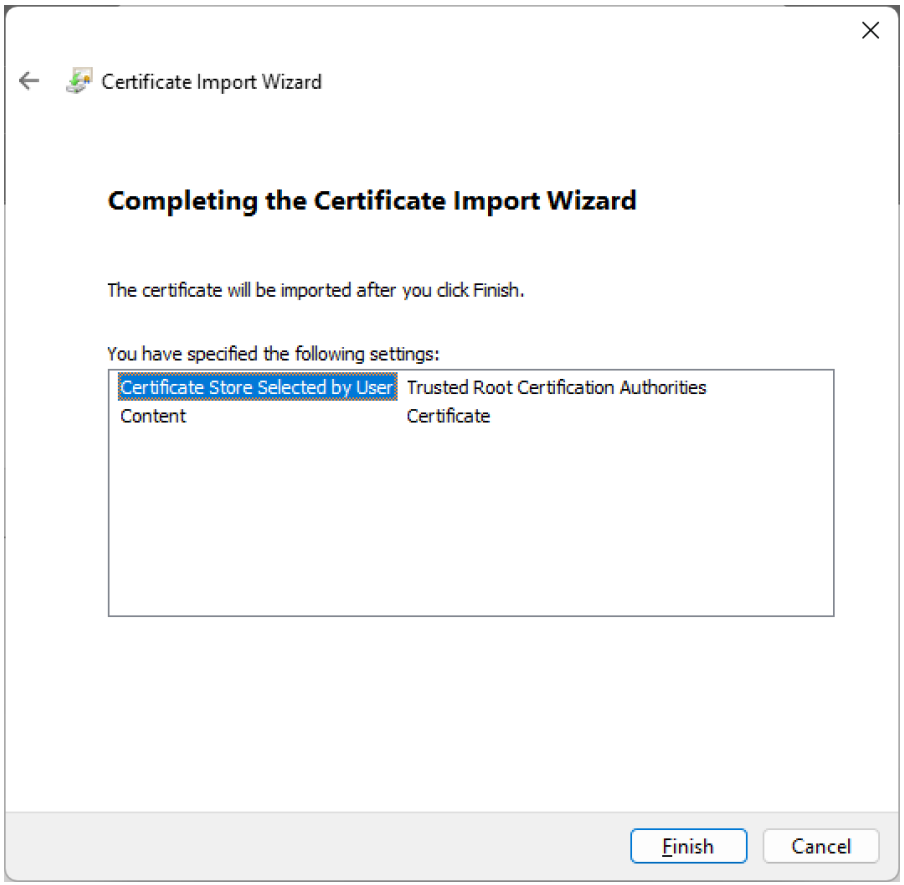
3. Verify that the CA (or the certificate) is trusted – add it, if missing



Analyzing certificate issues

Trustworthy or not, that's the question!

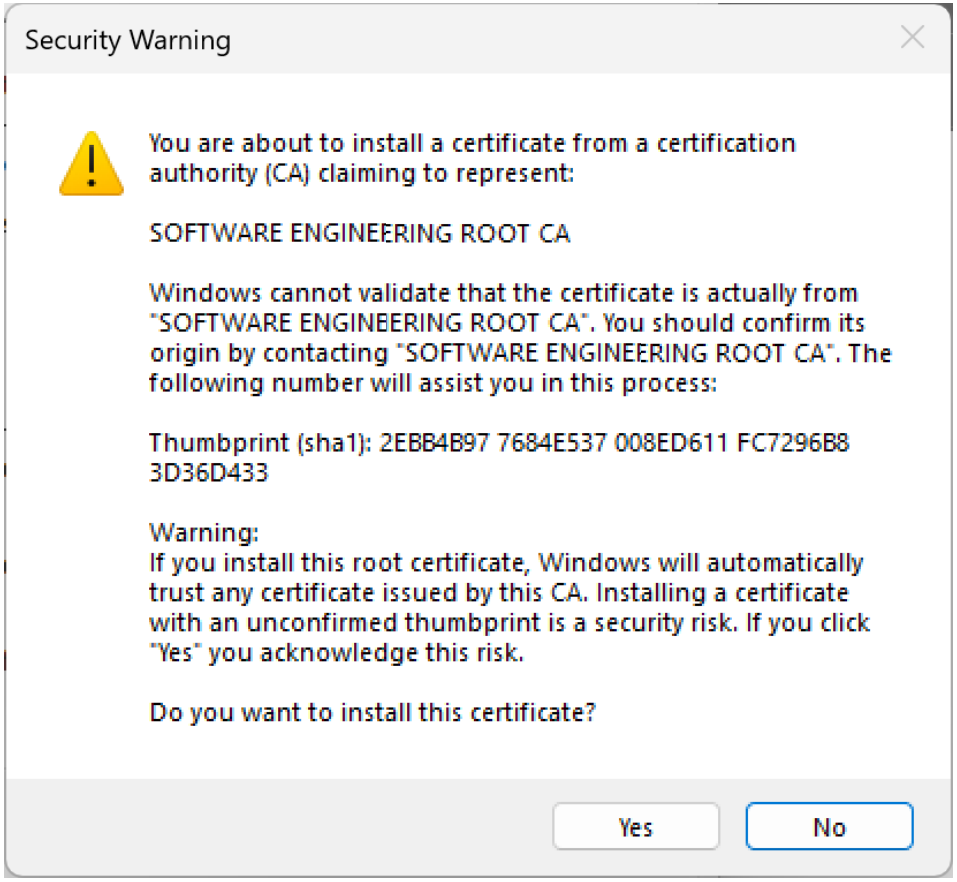
3. Verify that the CA (or the certificate) is trusted – add it, if missing



Analyzing certificate issues

Trustworthy or not, that's the question!

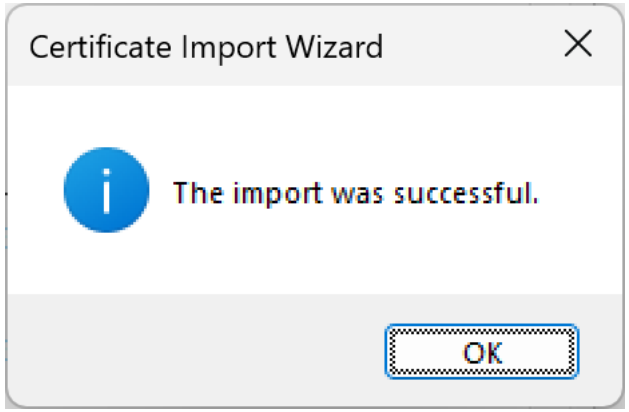
3. Verify that the CA (or the certificate) is trusted – add it, if missing

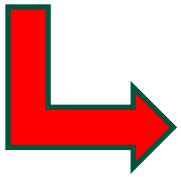


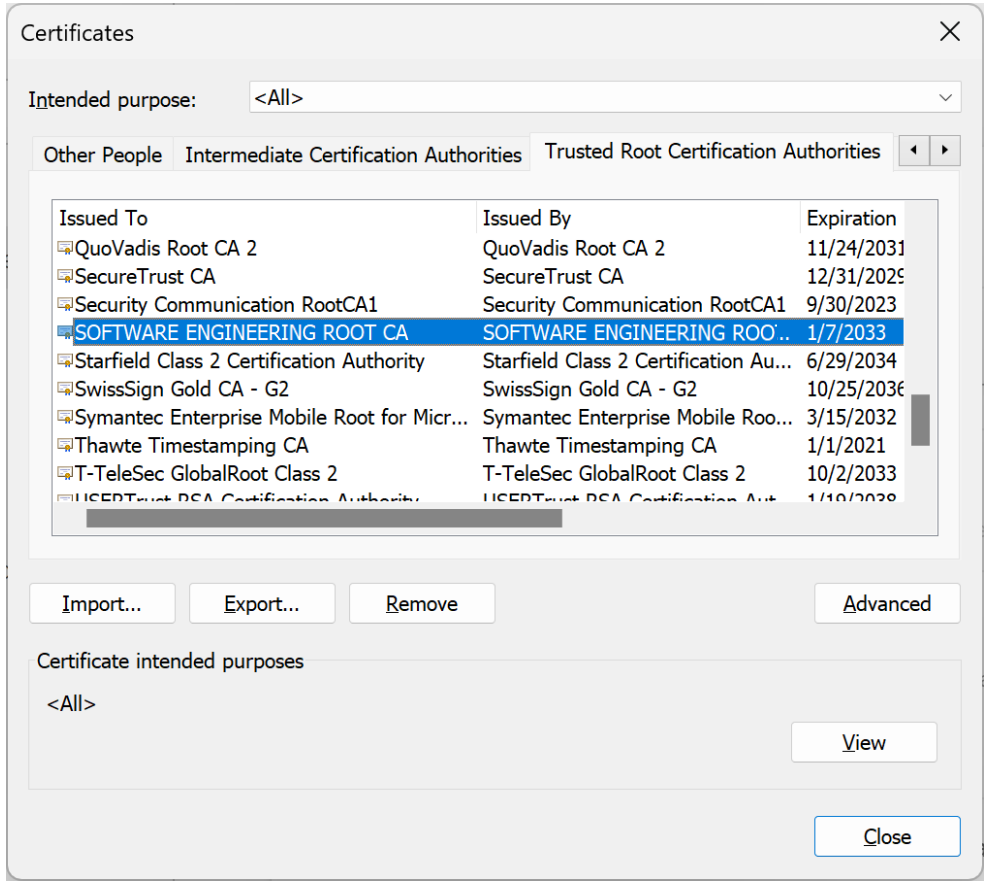
Analyzing certificate issues

Trustworthy or not, that's the question!

3. Verify that the CA (or the certificate) is trusted – add it, if missing



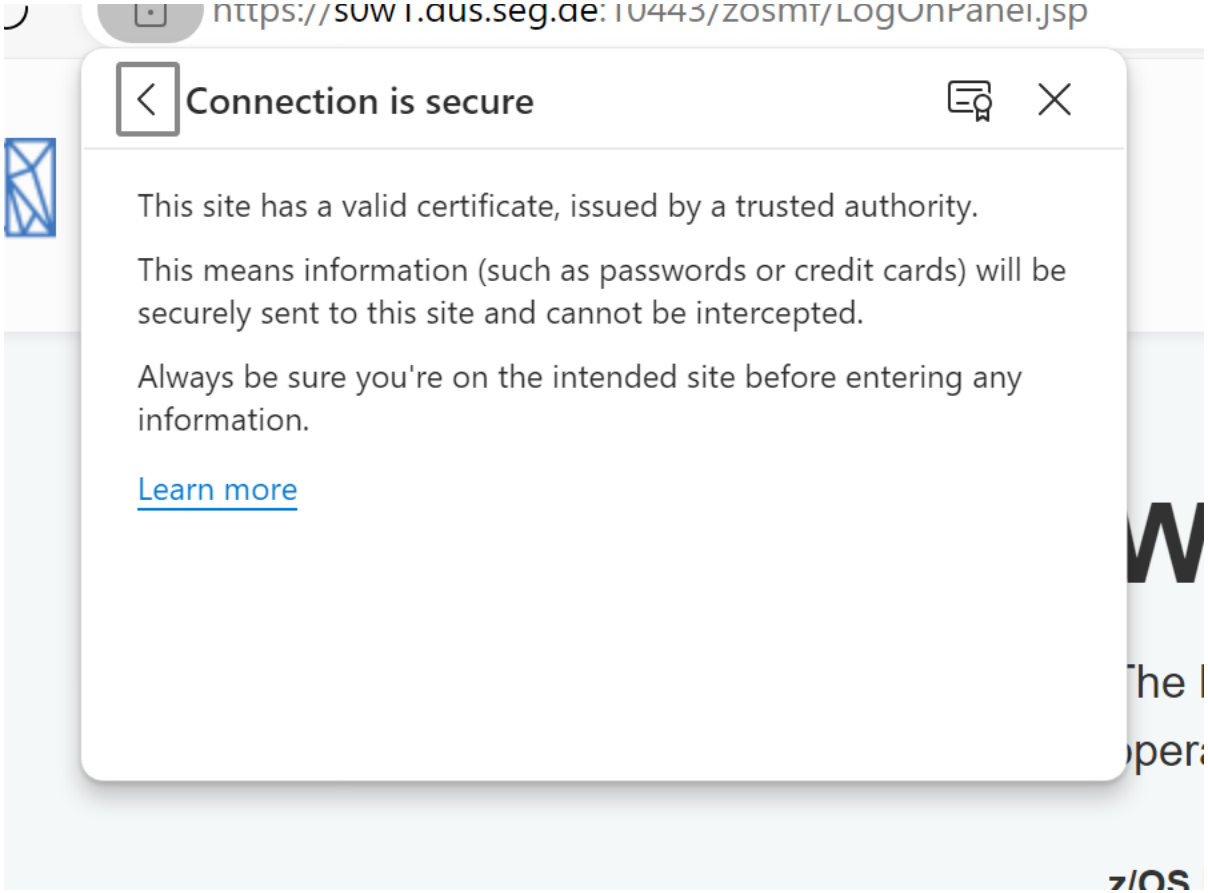




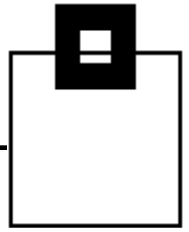
Analyzing certificate issues

Trustworthy or not, that's the question!

3. Verify that the CA (or the certificate) is trusted – add it, if missing



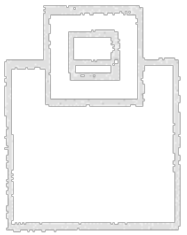
Analyzing certificate issues

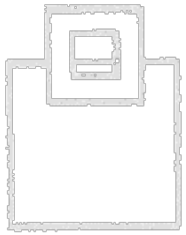


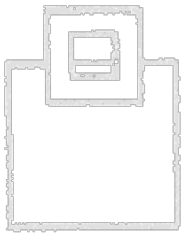
Trustworthy, or not, that's the question!

But what can you do if it's not a browser client, but an API, like a RESTful service?

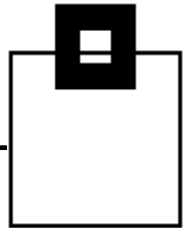
→ OPENSSLs tls debugging is your friend!



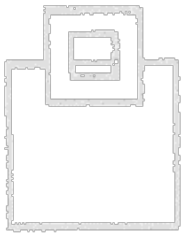


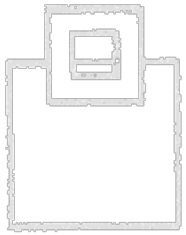


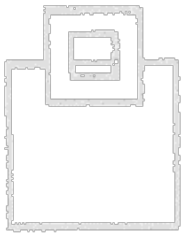
Analyzing certificate issues



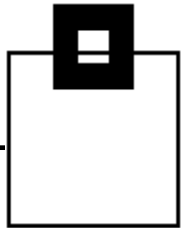
```
openssl s_client -connect s0w1.dus.seg.de:15151 -tlsextdebug
CONNECTED(00000005)
TLS client extension "renegotiation info" (id=65281), len=1
0001 - <SPACES/NULS>
depth=1 C = DE, ST = NORTH RHINE WESTPHALIA, L = DUESSELDORF, O =
SOFTWARE ENGINEERING GMBH, OU = DEVELOPMENT, CN = SOFTWARE
ENGINEERING ROOT CA
verify error:num=19:self signed certificate in certificate chain
verify return:0
write W BLOCK
---
Certificate chain
 0 s:/C=DE/ST=NORTH RHINE WESTPHALIA/L=DUESSELDORF/O=SOFTWARE
ENGINEERING GMBH/OU=DEVELOPMENT/CN=DB2 SECURE DISTRIBUTION SERVICE
   i:/C=DE/ST=NORTH RHINE WESTPHALIA/L=DUESSELDORF/O=SOFTWARE
ENGINEERING GMBH/OU=DEVELOPMENT/CN=SOFTWARE ENGINEERING ROOT CA
 1 s:/C=DE/ST=NORTH RHINE WESTPHALIA/L=DUESSELDORF/O=SOFTWARE
ENGINEERING GMBH/OU=DEVELOPMENT/CN=SOFTWARE ENGINEERING ROOT CA
   i:/C=DE/ST=NORTH RHINE WESTPHALIA/L=DUESSELDORF/O=SOFTWARE
ENGINEERING GMBH/OU=DEVELOPMENT/CN=SOFTWARE ENGINEERING ROOT CA
---
```





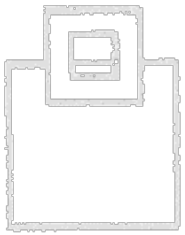


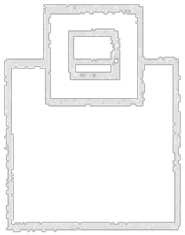
Analyzing certificate issues

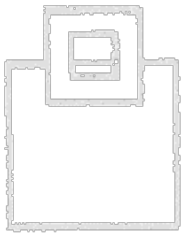


Server certificate

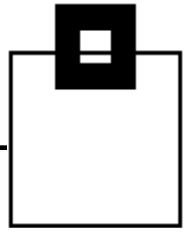

```
-----BEGIN CERTIFICATE-----
MIIEgDCCA9igAwIBAgIBBDANBgkqhkiG9w0BAQsFADCBpDELMAkGA1UEBhMCREUx
HzAdBgNVBAgTFk5PUlRIIFJISU5FIFdFU1RQSEFMSlHjjO85BgNVBAcTC0RVRVNT
RUxET1JGMSIwIAYDVQQKExlTT0ZUV0FSRSBFTkdJTkVFUklORyBHTUJIMRQwEgYD
VQQLEwtERVZFTE9QTUVOVDEkMCIGA1UEAxMbU09GVFdBUkUgRU5HSU5FUklORyBS
T09UIENBMB4XDTIzMDExNTIzMDAwMFoXDTI1MDQwMTIyNTk1OVowgagxCzAJBgNV
BAYTAkRFMR8wHQYDVQQIExZOT1JUSCBSSElORSBXRVNUUEhBTElBMRQwEgYDVQQH
EwtEVUVTU0VMRE9SRjEiMCAGA1UEChDGC09GVFdBUkUgRU5HSU5FRVJJTkcgR01C
SDEUMBIGA1UECxMLREVWRUxPUE1FTlQxKDAmBgNVBAMTH0RCMiBTRUNVUkUgRElT
VFJJQlVUSU9OIFNFUlZJQ0UwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
AQD70x0TZ5WsqsK6ZTy3b+Ry+xIcMTawO1+OeVG04dOPvrZEtVsvicS74vdllilB
I1OYncHNZ9/3E8RwxTv5qSxG4KW6PKsgd2Qpk7iBP4rMXKkrvp8rEp00OW0LgPur
4sCtQpEytfYps/AFhNwPoT1hK1hZkXjywILn7/sJ3t9zYCesDDUJlEJkywaO8U/V
vgLh0SsEq2aUlaxSYhyc4KAPsdencU0QuzSZhbwMyA+4i0eSK4fgOsGUmSoACVc4
Tg0qvFLF6iTcPEXW9XNJqlVGqg1RaWuNwKG00Z0lETZUbAVZsam4exiYnRUiT6J9
oyPfzQnB8+w59ir2Jx3p8wfbAgMBAAGjgbYwgbMwPwYJYIZIAYb4QgENBDIWMEdl
bmVyYXRlZCBieSB0aGUgU2VjdXJpdHkgU2VydmVyIGZvciB6L09TIChSQUNGKTAg
BgNVHREEGTAXgg9TMFcxLkRVUy5TRUcuREWHBMCoCWIwDgYDVR0PAQH/BAQDAgWg
MB0GA1UdDgQWBBQlyjuoy6SipU3H23fH7cpw+ALB0zAfBgNVHSMEGDAWgBT/MgiN
4im65Gpt4iPBBGhEz1XpXzAhffEdq2iG9w0BAQsFAAOCAQEAkDFU531SDp3lG1jH
IPdA6w9MeJx344sgd/K4LPzfIGuzmmuHZrAHCHZNaA64BBMogeGOV2zoxenwf07A
CIeTQpqE19TuNH2vyrulMd8p4c6VwUjto/N+GXobE3WmNt5nrdGLOIqrxutwmiMD
2HElOlIh7unsVqq24qfDczxHNVLapJlYy4gXiqC/UG8055GhjIwEaMvfEQ82GhcI
v1pekhL7hK0p8xGOAYQVBUM0MrpVBCSiFYdVs2hPaTA86QcyngT9CGNrXf2JeTgk
FIzH7h3nLdCRZd9KXQATQ5b24a9OXGzC6bKgiSD9unxWI8DYxBXOx3G3kufaXn2X
kOE/EQ==
-----END CERTIFICATE-----
```

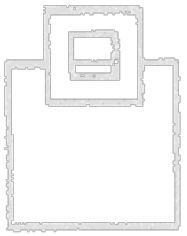


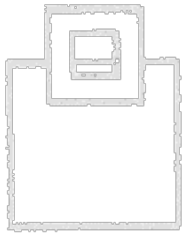


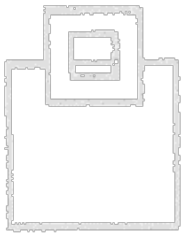
Analyzing certificate issues



Base 64 encoded certificates can be decoded using OPENSSL:







```
Certificate: Data: Version: 3 (0x2) Serial Number: 4 (0x4) Signature Algorithm: sha256WithRSAEncryption
Issuer: C=DE, ST=NORTH RHINE WESTPHALIA, L=DUESSELDORF, O=SOFTWARE ENGINEERING GMBH, OU=DEVELOPMENT,
CN=SOFTWARE ENGINEERING ROOT CA Validity Not Before: Jan 15 23:00:00 2023 GMT Not After : Apr 1 22:59:59
2025 GMT Subject: C=DE, ST=NORTH RHINE WESTPHALIA, L=DUESSELDORF, O=SOFTWARE ENGINEERING GMBH,
OU=DEVELOPMENT, CN=DB2 SECURE DISTRIBUTION SERVICE Subject Public Key Info: Public Key Algorithm:
rsaEncryption Public-Key: (2048 bit) Modulus: 00:fb:d3:1d:13:67:95:ac:aa:c2:ba:65:3c:b7:6f:
e4:72:fb:12:1c:31:36:b0:3b:5f:8e:79:51:b4:e1: d3:8f:be:b6:44:b5:5b:2f:89:c4:bb:e2:f7:65:96:
29:41:23:53:98:9d:c1:cd:67:df:f7:13:c4:70:c5: 3b:f9:a9:2c:46:e0:a5:ba:3c:ab:20:77:64:29:93:
b8:81:3f:8a:cc:5c:a9:2b:be:9f:2b:12:9d:34:39: 6d:0b:80:fb:ab:e2:c0:ad:42:91:32:b5:f6:29:b3:
f0:05:84:dc:0f:a1:3d:61:2b:58:59:91:78:f2:c0: 82:e7:ef:fb:09:de:df:73:60:27:ac:0c:35:09:94:
42:64:cb:06:8e:f1:4f:d5:be:02:e1:d1:2b:04:ab: 66:94:95:ac:52:62:1c:9c:e0:a0:0f:b1:d7:a7:71:
4d:10:bb:34:99:85:bc:0c:c8:0f:b8:8b:47:92:2b: 87:e0:3a:c1:94:99:2a:00:09:57:38:4e:0d:2a:bc:
52:c5:ea:24:dc:3c:45:d6:f5:73:49:aa:55:46:aa: 0d:51:69:6b:8d:c0:a1:b4:d1:9d:25:11:36:54:6c:
05:59:b1:a9:b8:7b:18:98:9d:15:22:4f:a2:7d:a3: 23:df:cd:09:c1:f3:ec:39:f6:2a:f6:27:1d:e9:f3: 07:db
Exponent: 65537 (0x10001) X509v3 extensions: Netscape Comment: Generated by the Security Server for z/OS
(RACF) X509v3 Subject Alternative Name: DNS:S0W1.DUS.SEG.DE, IP Address:192.168.9.98 X509v3 Key Usage:
critical Digital Signature, Key Encipherment X509v3 Subject Key Identifier:
25:CA:3B:A8:CB:A4:A2:A5:4D:C7:DB:77:C7:ED:CA:70:F8:02:C1:D3 X509v3 Authority Key Identifier:
FF:32:08:8D:E2:29:BA:E4:6A:6D:E2:23:C1:04:68:44:CF:55:E9:5F Signature Algorithm: sha256WithRSAEncryption
Signature Value: 90:31:54:e7:7d:52:0e:9d:e5:1b:58:c7:20:f7:40:eb:0f:4c:
78:9c:77:e3:8b:20:77:f2:b8:2c:fc:df:20:6b:b3:9a:6b:87:
66:b0:07:08:76:4d:68:0e:b8:04:13:28:81:e1:8e:57:6c:e8:
c5:e9:f0:7f:4e:c0:08:87:93:42:9a:84:d7:d4:ee:34:7d:af:
ca:bb:a5:31:df:29:e1:ce:95:c1:48:ed:a3:f3:7e:19:7a:1b:
13:75:a6:36:de:67:ad:d1:8b:38:8a:ab:c6:eb:70:9a:23:03:
d8:71:25:3a:52:21:ee:e9:ec:56:aa:b6:e2:a7:c3:73:3c:47:
35:52:da:a4:99:58:cb:88:17:8a:a0:bf:50:6f:34:c3:b8:d0:
33:1c:04:68:cb:df:11:0f:36:1a:17:08:bf:5a:5e:92:12:fb:
84:ad:29:f3:11:8e:01:84:15:05:43:34:32:ba:55:04:24:a2:
15:87:55:b3:68:4f:69:30:3c:e9:07:32:9e:04:fd:08:63:6b:
5d:fd:89:79:38:24:14:8c:c7:ee:1d:e7:2d:d0:91:65:df:4a:
5d:00:13:43:96:f6:e1:af:4e:5c:6c:c2:e9:b2:a0:89:20:fd:
ba:7c:56:23:c0:d8:c4:15:ce:c7:71:b7:92:e7:da:5e:7d:97: 90:e1:3f:11
```