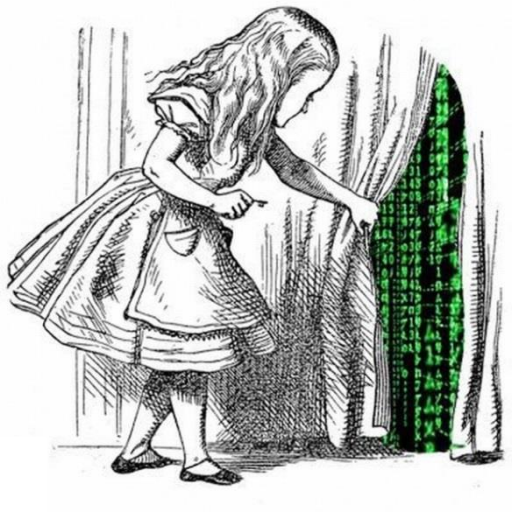
SSL/TLS for Developers

Jayson Salazar
jsalazar@ernw.de

Dominik Schneider
dschneider@ernw.de

Road Map



- Short Introduction to SSL/TLS
  - TLS as a Protocol
  - TLS as a Toolbox
  - TLS as a Process
- The Developer's Role
- Conclusions

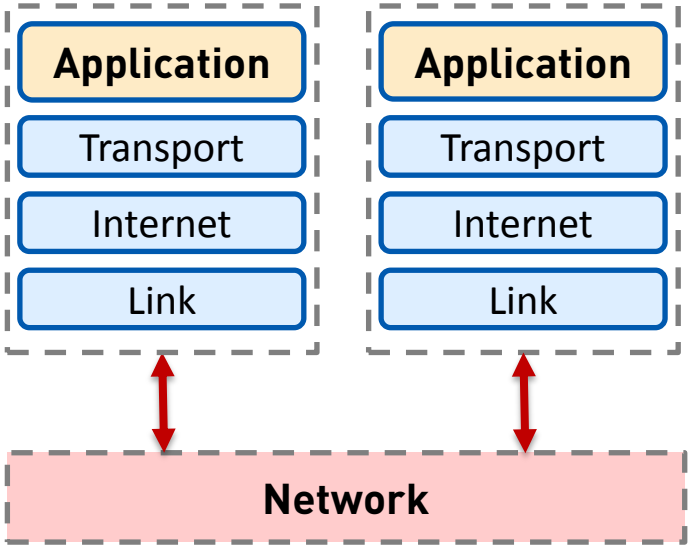
Introduction to SSL/TLS

On how and why everything started

In the Beginning ...

- Information travelling through a **collection of untrusted networks** is what we call the Internet today.
- The Internet has been supporting **increasingly critical applications** since the 80's, from E-mail to E-Commerce it wasn't an experiment anymore.
- **Eavesdropping** was one of the **major concerns**
  - Telnet, FTP, SMTP, POP and HTTP.

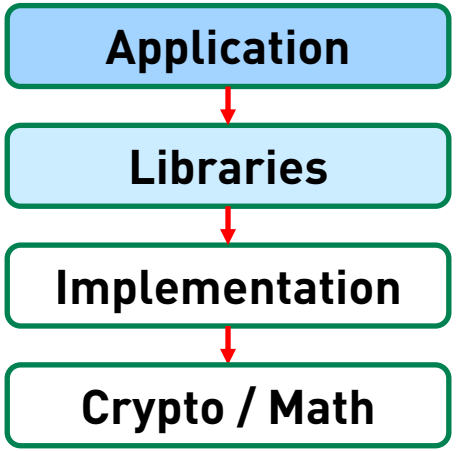
Something had to Change ...



- A **mechanism** was needed to provide **transport** level **security** between two end-points.

- **According to RFC6101, "[...] to provide privacy and reliability between two communicating applications [...]"**

Our Focus will be ...

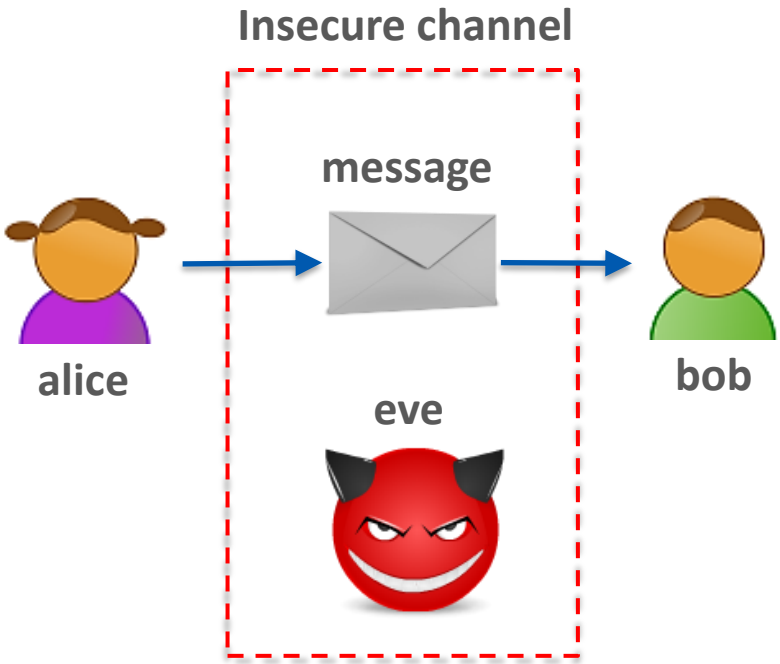


- People **usually** focus on the **math** and **implementation** details behind TLS.
- We, however, will look at **TLS from a different angle**.
- We will explore how **developers** can leverage **libraries** and **tools** in order to **protect** their **infrastructure and applications**.

Cryptography Basics

Concepts behind SSL/TLS

The Initial Situation

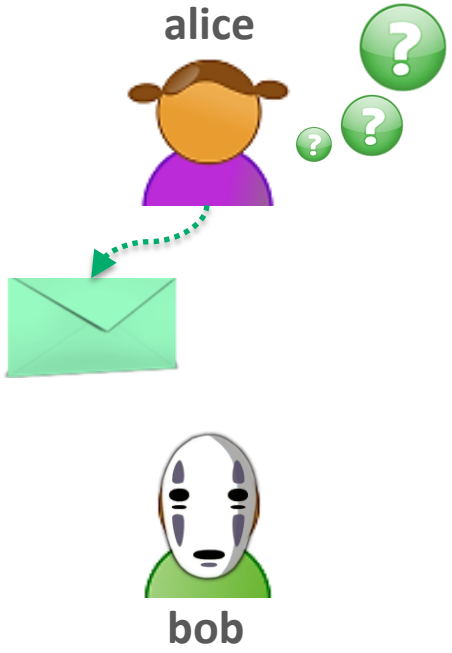


- Alice wants to **securely share** a message with Bob.
- The message is **accessible** to everyone with Alice's or Bob's **capabilities**.
- **Eve** is assumed to have said **capabilities**.
- **Eve has access** to the message.
- Eve can also **modify** the message **in transit**.

Security in our Scenario Means that ...

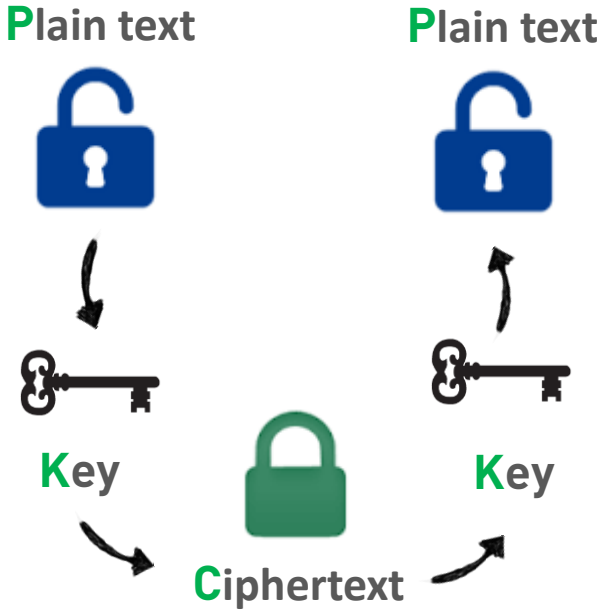
- **Confidentiality**
  - The message's content can only be accessed by the desired participants.
- **Integrity**
  - The message remains unmodified once it leaves its origin or it is trivial to say that it has been manipulated.

Security in our Scenario Means that ...



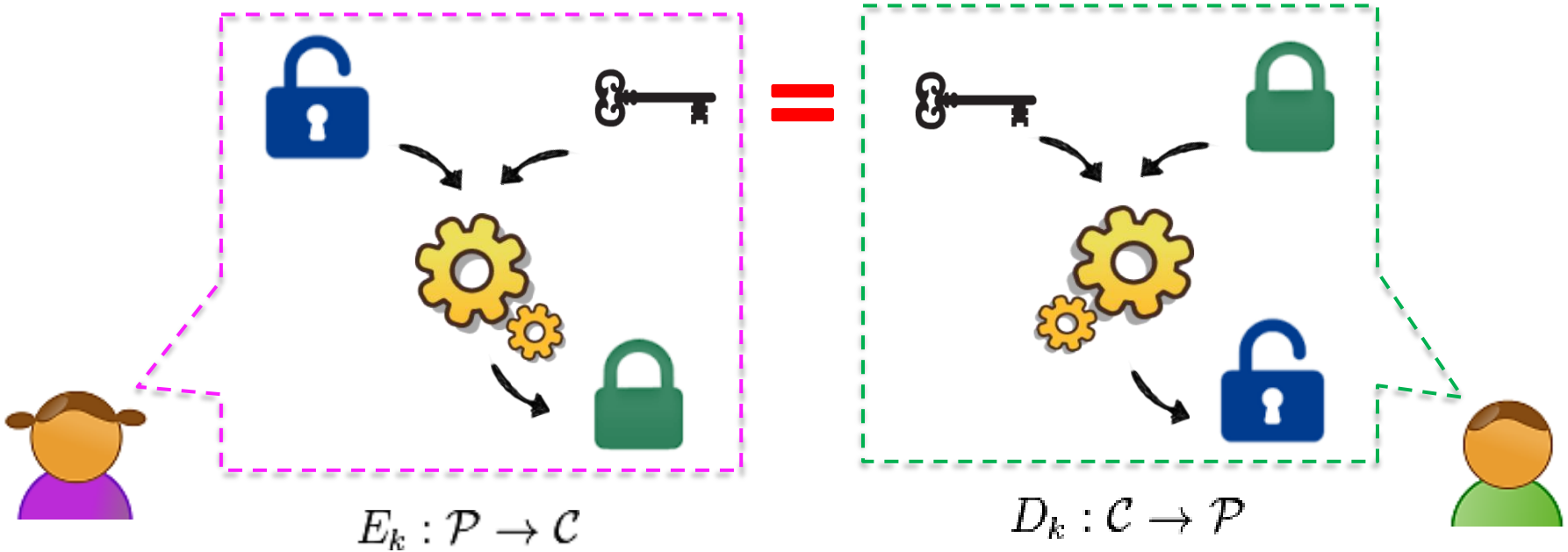
- **Authenticity**
  - The origin of the message can be determined.
- **Non-Repudiation**
  - A party cannot deny having sent the message.

Basic Concepts Behind Symmetric Cryptography

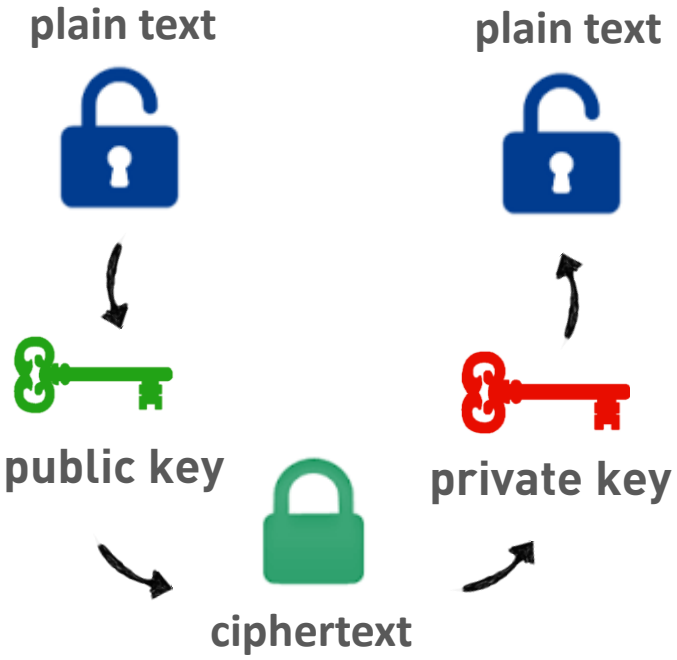


- Plaintext, usually **presented** as **P**, is the original message which is to be transmitted.
- Key, **usually K**, is a parameter which is used to transform **P** in a certain way.
- The resulting **transformation** of **P** by using **K** is what's called **ciphertext**, usually **C**.

A Basic Symmetric Cryptosystem



Asymmetric Encryption and Cryptosystems



- The **Diffie-Hellman** algorithm **marked** the **beginning** of what we know as asymmetric cryptography today.
- However, **other algorithms** can be used to generate keys.
- Keys such as those needed and generated in the **RSA** cryptosystem are the most **commonly used**.

One Concept, Several Mechanisms

- Asymmetric encryption
- Public-key Infrastructure
- Digital signatures
  - Message authentication codes
  - Code signing
  - Application signing

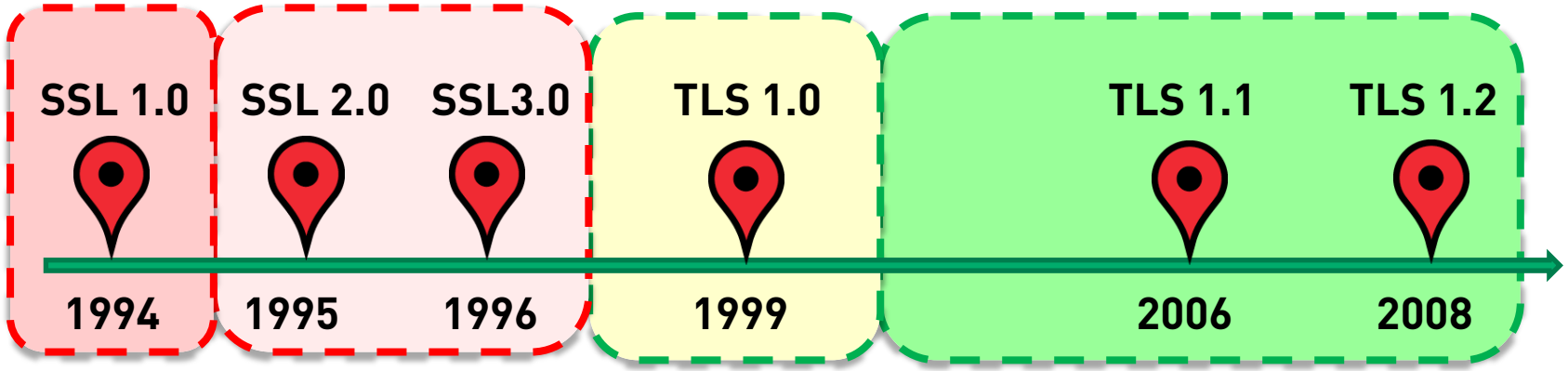
But this is all theory, what about implementations?

Finally, TLS Becomes Part of the Internet Fabric

- Noticing the relative **success** of **SSL 3.0**, the **IETF took over** the development of SSL under the name TLS.
- Contrary to what people think, the **name change** was merely due to **political reasons**.
- **TLS 1.0** has been specified by **RFC2246**.

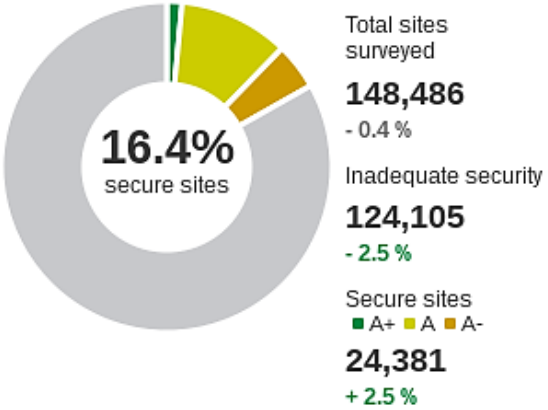
As Usual, there have been Improvements

SSL 1.0 never saw the light and critical security flaws in SSL 2.0 and 3.0 have been discovered in the past decade. The only acceptable standards whose usage is **recommended** are **TLS 1.0** and its newer versions.

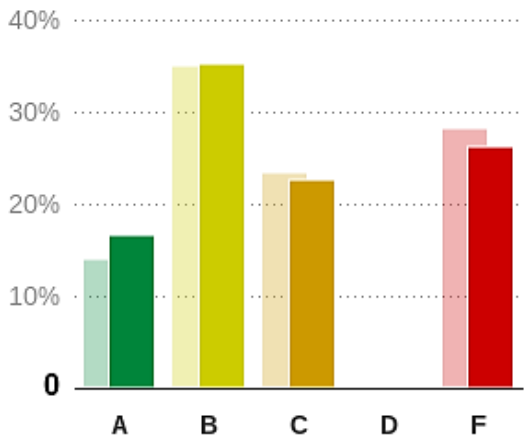


Where are we now a Days?

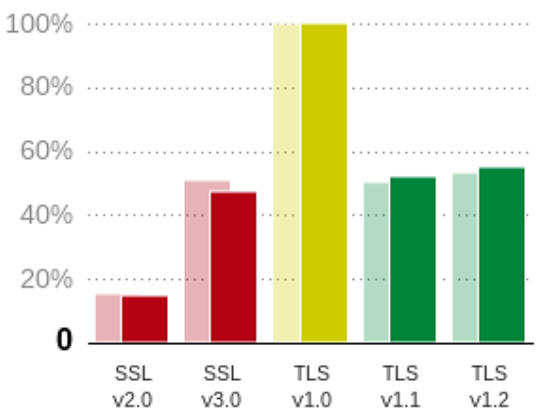
SSL Security Summary



SSL Labs Grade Distribution



Protocol Support



Taken from: https://www.trustworthyinternet.org/ssl-pulse/

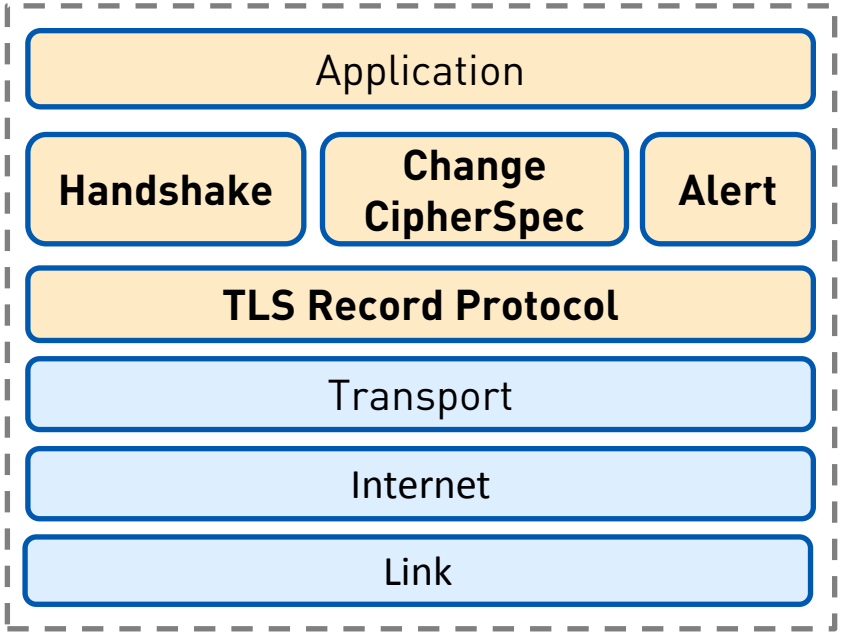
TLS as a Protocol

More than handshake and crypto-buzzwords

In Summary, the Purpose of TLS is ...

- To provide **encapsulation** for application specific protocols such as HTTP, FTP, SMTP, etc.
- To allow client-server applications to **communicate securely** across an untrusted network, to prevent eavesdropping and tampering.

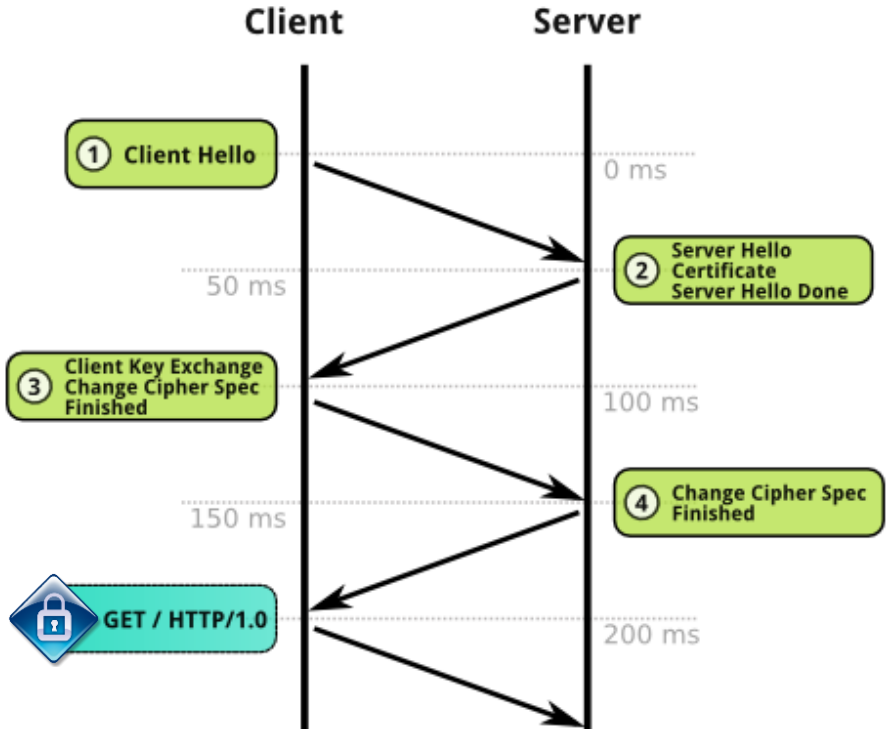
The Actual Components of TLS



TLS subprotocols:

- TLS Record
- TLS Handshake
- TLS Change Cipher Spec
- TLS Alert Protocol
- TLS Application Data

TLS Handshake Protocol



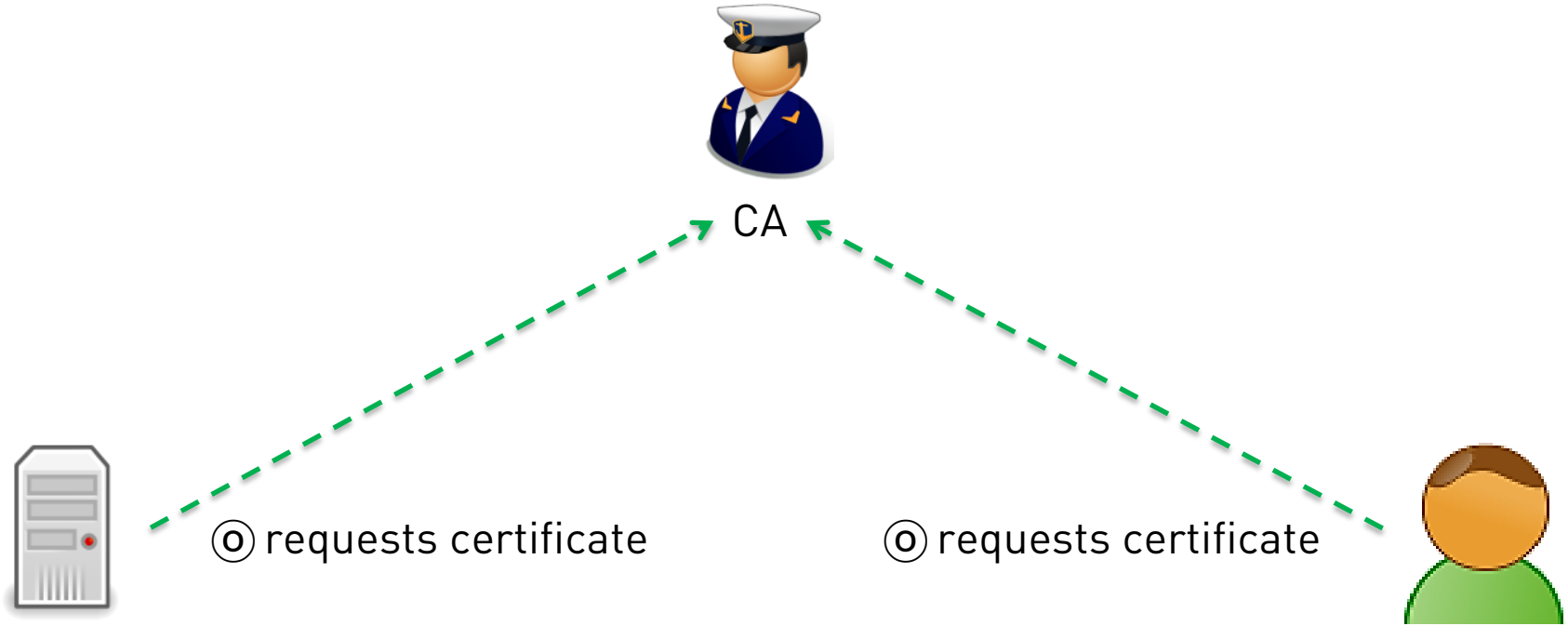
- **Cipher-suite** to be used during the session is **negotiated**.
  - Client and server HELLO messages.
- **Server** and **client** are **authenticated**, their identities verified.
  - By means of Public Key Infrastructure (PKI)
- **Key material** which is to be used during the session is **exchanged**.

TLS Record Protocol

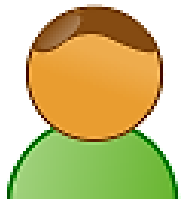
- Outgoing **application data** is **divided** into manageable **blocks** and incoming messages are reassembled.
- Data is **compressed** and **decompressed** (optional).
- **Outgoing** messages are **encrypted** and **incoming** ones are **decrypted**.
- **Verifies** the **integrity** of the data by using a mechanism called Message Authentication Code (MAC).

How does Authentication Work?

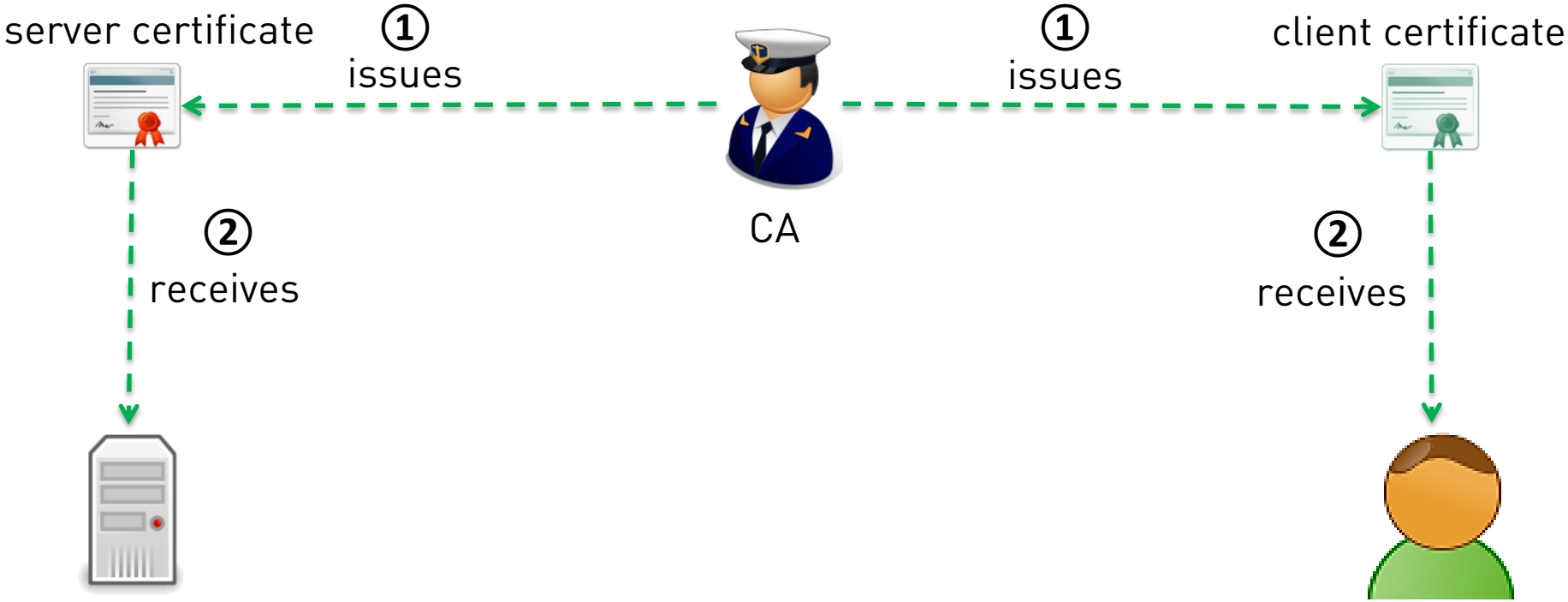
It's all about Trust, Ideal Deployment



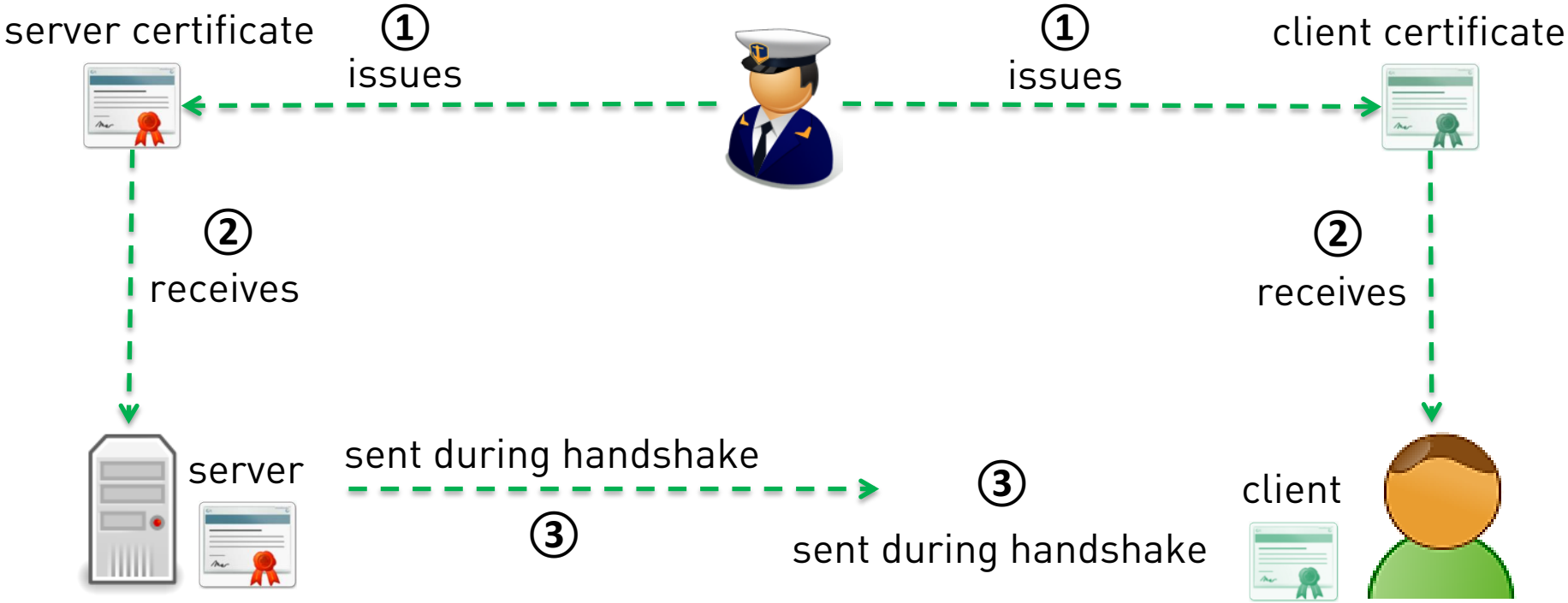
It's all about Trust, Ideal Deployment



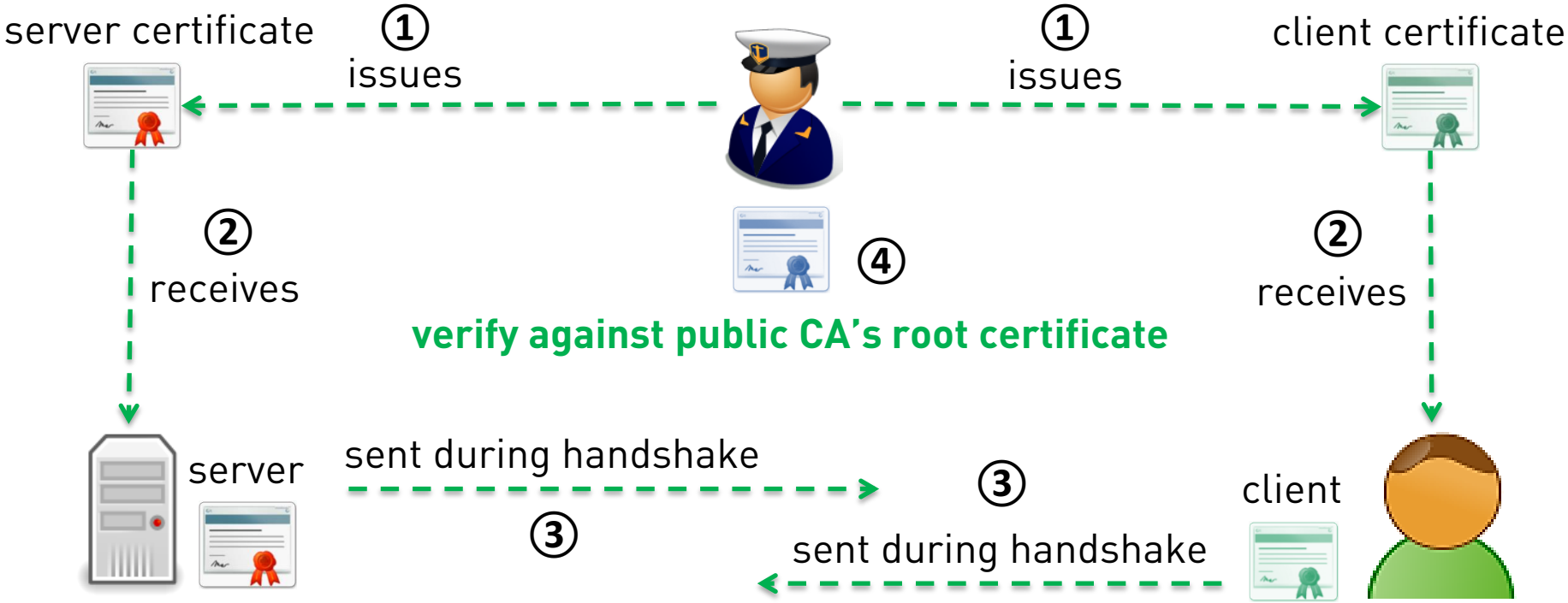
It's all about Trust, Ideal Deployment



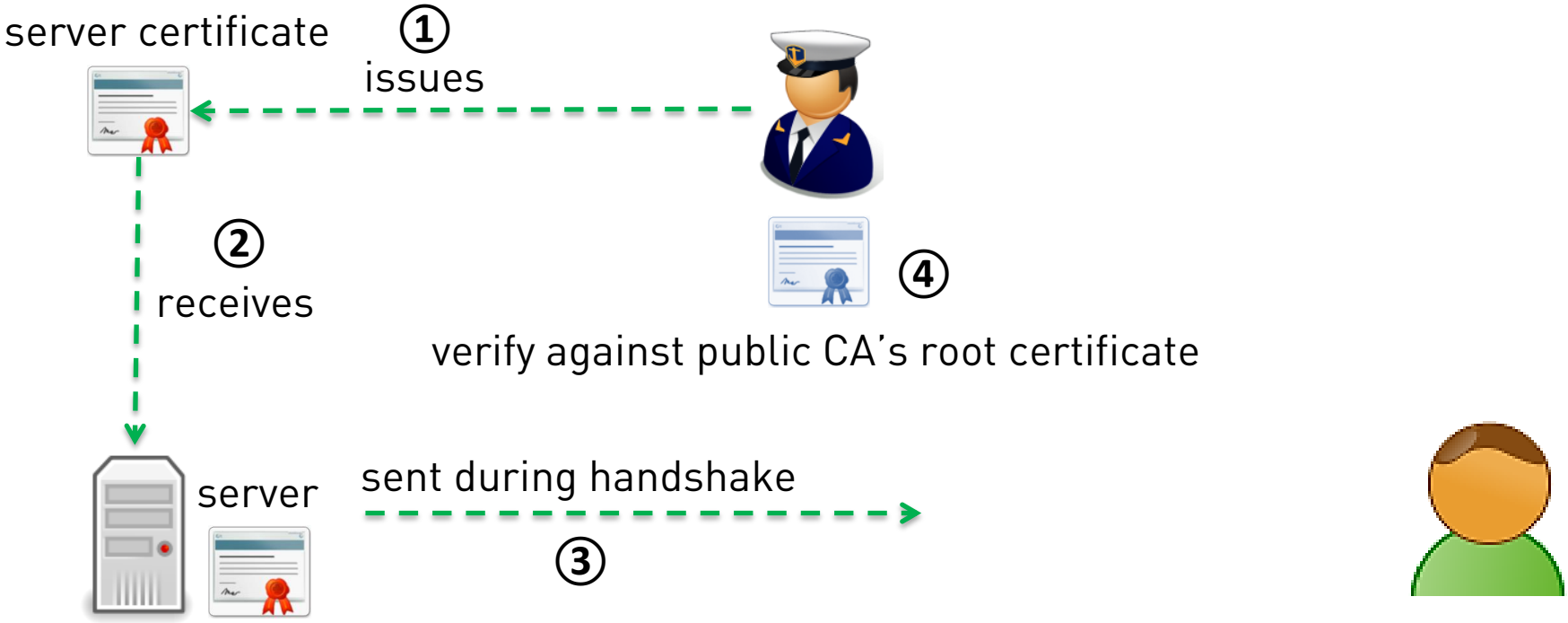
It's all about Trust, Ideal Deployment



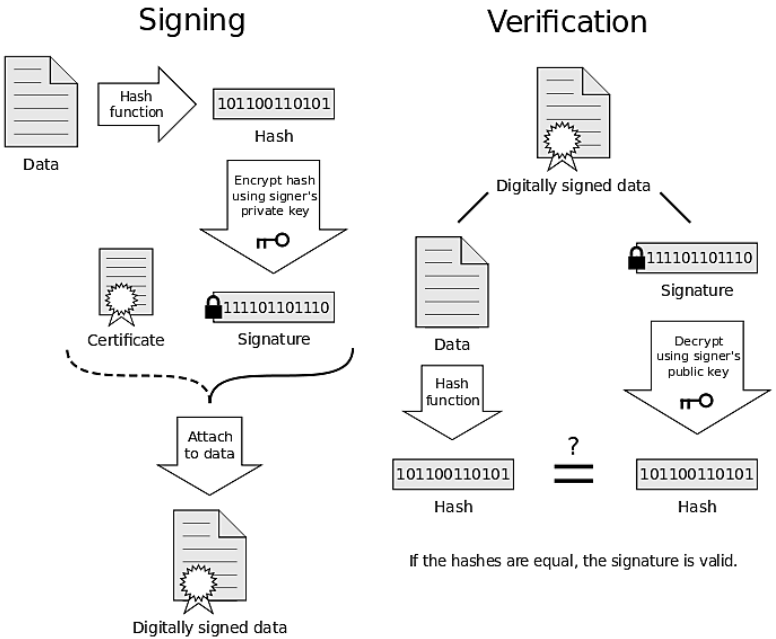
It's all about Trust, Ideal Deployment



However, the Common Deployment is ...



What, exactly, is a Certificate?

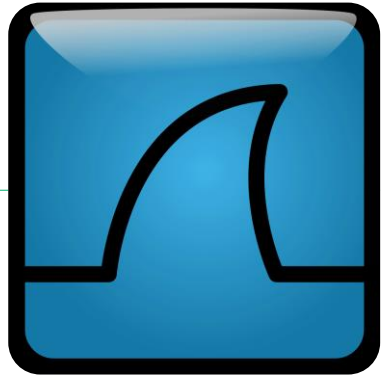


- A certificate **binds** the **identity** of a person or organization **to** a public **key**
- It is **signed by** the **CA** with its private key so that anyone with access to its public key can verify the signature.

Brief Summary

- **Purpose** is to provide **confidentiality**, **integrity** and **authentication** during **data** exchanges
- The **handshake** protocol **negotiates** the parameters for the secure channel and is responsible for the establishment of the TLS session.
- The **record** protocol **encapsulates** application data encrypting it and checking its integrity in the process.
- Public-Key Infrastructure(**X509**) certificates are **fundamental** in TLS.

Let's have a look at TLS on the Wire



TLS as a Toolbox

From conception to certificates

From Theory to Implementations

OpenSSL, most popular SSL/TLS implementation.

- **NSS**, "[...] set of libraries designed to support cross-platform development of security-enabled client and server applications."
- **GnuTLS**, an **OpenSSL** fork due to licensing disagreements.
- **LibreSSL**, post-heartbleed fork of **OpenSSL**.
- Private and proprietary Implementations

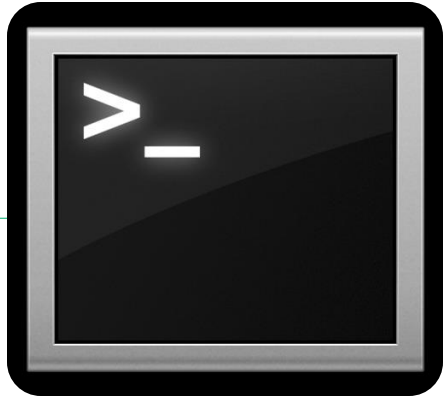
The OpenSSL Command Line Interface

- Is a **wrapper around** the OpenSSL **crypto-library** and allows for:
  - Creation and **management** of private **keys** and public keys
  - Public key **cryptographic operations**
  - Creation of X.509 **certificates**, CSRs and CRLs
  - Calculation of message **digests**
  - SSL/TLS Client and Server **Tests**

Windows-World-Specific Alternatives

- The **CertUtil** command-line tool.
- **Active Directory integration** ☺
- Part of **Active Directory Certification Services**
- Can be used to "[...] **dump** and **display** certification authority (CA) configuration information, **configure** Certificate Services, **back up** and **restore** CA components, and **verify** certificates, key pairs, and certificate chains."

Working with the OpenSSL CLI



Getting to Know OpenSSL

```
[~]$ man openssl
[~]$ openssl --help
[~]$ openssl list-standard-commands
[~]$ openssl list-cipher-commands
[~]$ openssl list-message-digest-commands
[~]$ openssl version
[~]$ openssl [command] --help
[~]$
[~]$ # All commands can also be used in interactive mode
[~]$ openssl

OpenSSL>
```

Generating Key-Pairs

```
[~]$ openssl rsa --help
[~]$ openssl genrsa -out priv.key.pem 4096
[~]$
[~]$ # Attention, the private key contains both the public
[~]$ # and private keys!
[~]$
[~]$ # We extract the public key like this:
[~]$ openssl rsa -pubout -in priv.key.pem > pub.key.pem
[~]$
[~]$ # However, it‘s better to keep our keys encrypted
[~]$ openssl rsa –aes256 -in priv.key.pem -out enc.key.pem
[~]$ openssl rsa –in pub.key.pem –text –noout -pubin
```

Trust and the Certificate Workflow

CA

generate key-pair ①

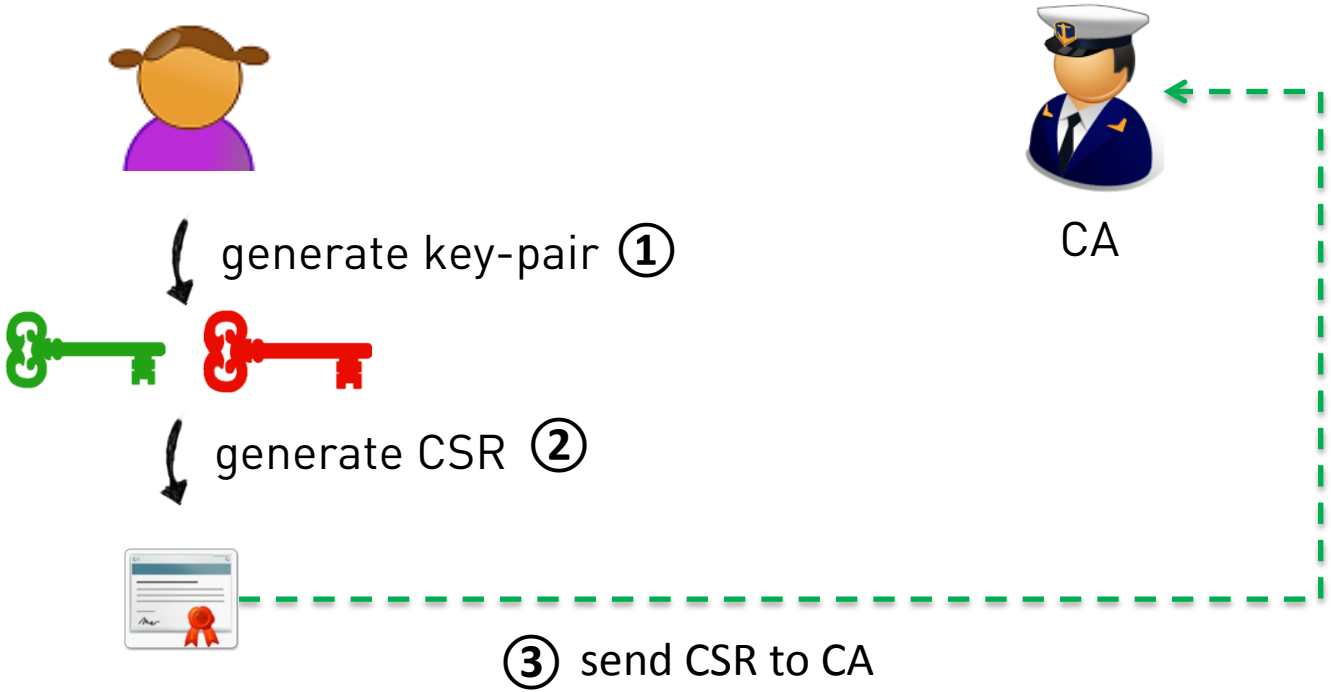
Trust and the Certificate Workflow

CA

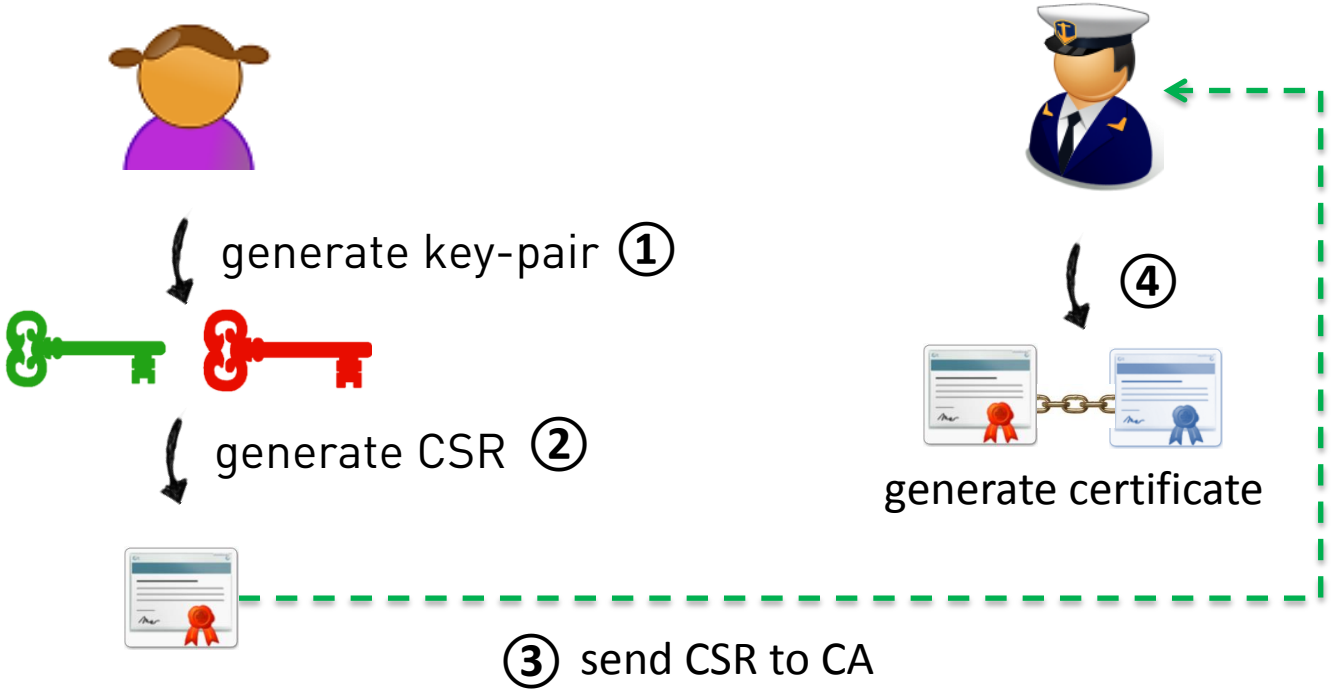
generate key-pair ①

generate CSR ②

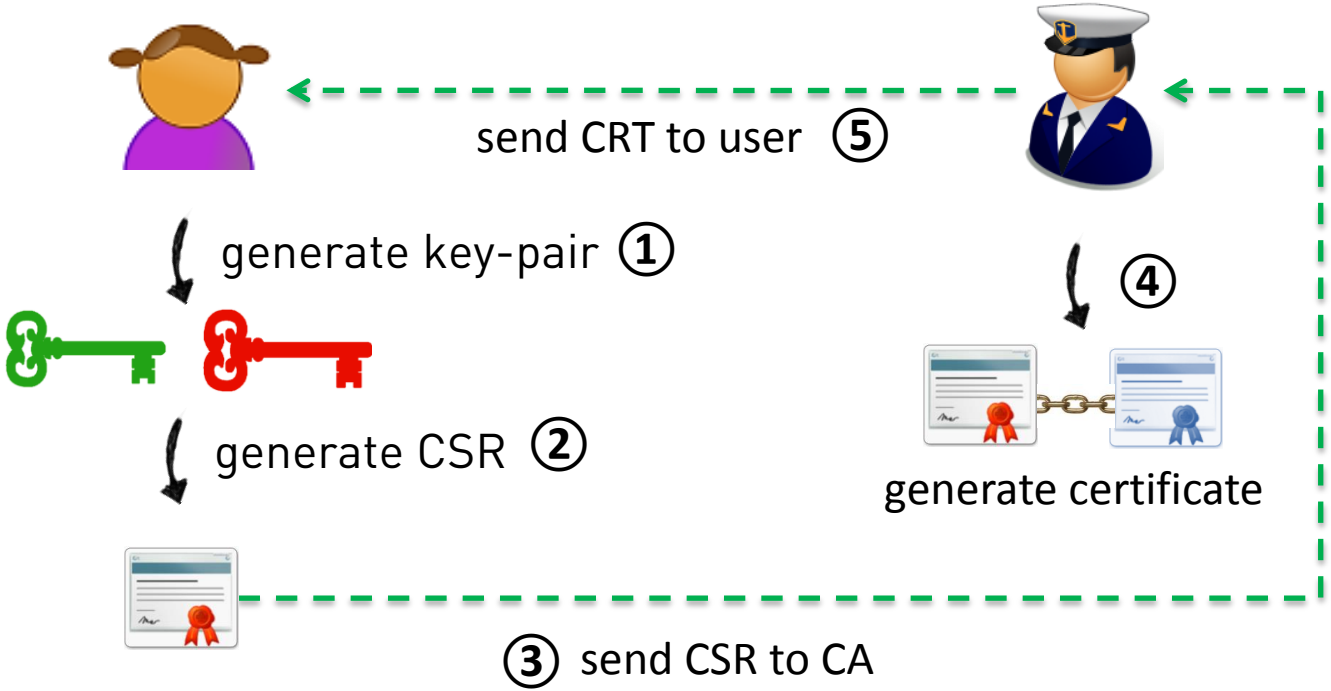
Trust and the Certificate Workflow



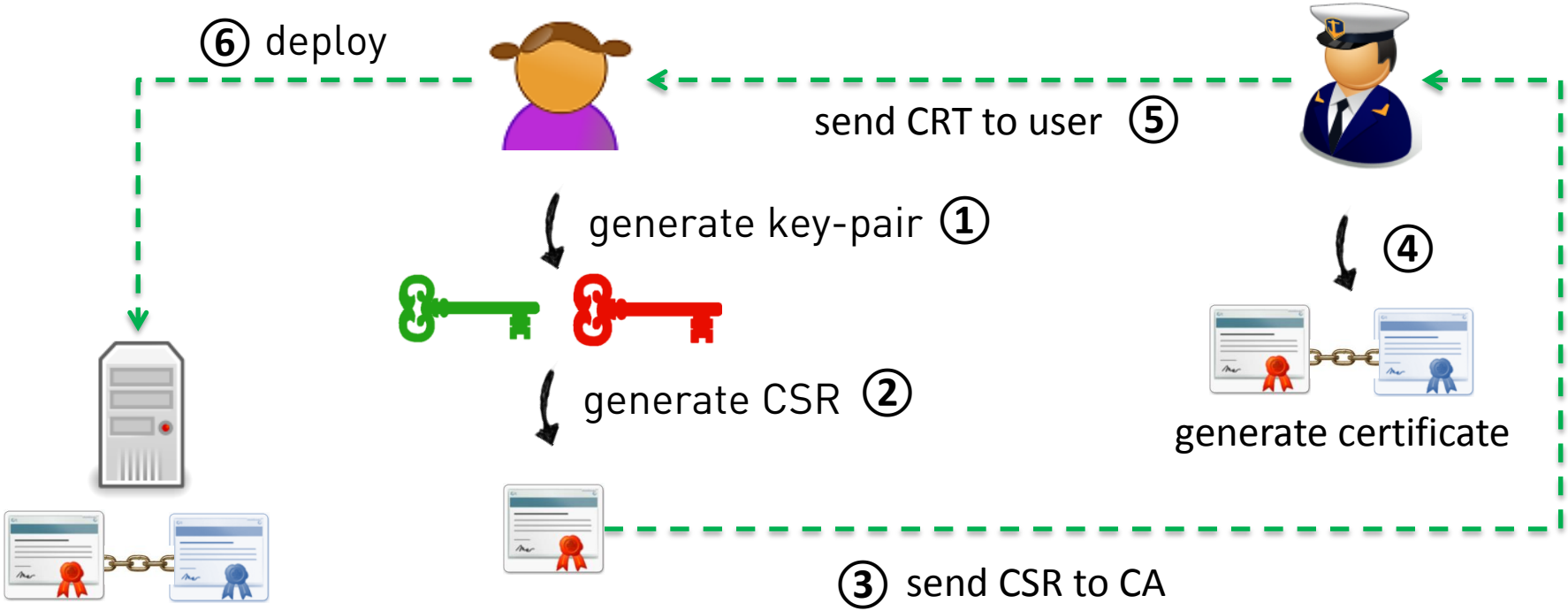
Trust and the Certificate Workflow



Trust and the Certificate Workflow



Trust and the Certificate Workflow



Generating Certificate Signing Requests

```
[~]$ # With an existing key-pair
[~]$ openssl req -new -key priv.key.pem -out leaf.csr.pem -config
csr.cnf
[~]$
[~]$ # Send to CA and wait for reply with an actual certificate
```

Creating your own Root Certificate

```
[~]$ # With an existing key-pair
[~]$ openssl req -x509 -new -key priv.key.pem -days 1024 -out
ca.crt.pem -config ca.cnf
[~]$
[~]$
```

Signing a Certificate Signing Requests

```
[~]$ # Signing the CSR
[~]$ openssl x509 -req -in leaf.csr.pem -CA ca.crt.pem -CAkey
priv.key.pem -CAcreateserial -out leaf.crt.pem -days 365
[~]$
[~]$ # Exporting the certificate to PKCS#12 format
[~]$ openssl pkcs12 -export -in leaf.crt.pem -inkey leaf.key.pem >
leaf.crt.p12
```

Ah! What are all these files? (I)

- **ASN.1** is a **standard** for representing, encoding, transmitting, and decoding data
  - **DER** is a particular binary **encoding**
- **PEM** files are **simply BASE64** encoded data with a simple header and a footer.
- **Anything** whose name begins with **PKCS** is cryptographic **standard** of which the first was defined by **RSA**, the company.

```
MyStructures DEFINITIONS ::= BEGIN

   MyTuple ::= SEQUENCE {
      number    INTEGER,
        word    IA5String
   }
END
```

```
mydata MyTuple::= {
        mynumber    5,
        myword      “information”
}
```

```
30 13 02 01 05 16 0e 69 6e 66 6f 72 6d 61 74
                69 6f 6e 0a
```

```
----- BEGIN MESSAGE -----
MzATAgEFFg5pbmZvcm1hdGlvbgoK
----- END MESSAGE -----
```

Ah! What are all these files? (II)

- **PKCS#1 (.key)** is used for keys, usually unencrypted. If encrypted, it will have a header.
- **PKCS#7 (.p7b .p7c)** can only contain certificates and chain certificates without keys
- **PKCS#8 (.p8)** is used for keys regardless of encryption, no human-readable header
- **PKCS#10 (.csr)** used exclusively for Certificate Singing Requests
- **PKCS#12 (.p12)** for exporting and backing up certificates and their certification paths with keys
- **CERT**, **CER** or **CRT** a certificate in DER or PEM

- All of the them come in DER and PEM flavors

TLS as a Process

Deployment, usage and maintenance

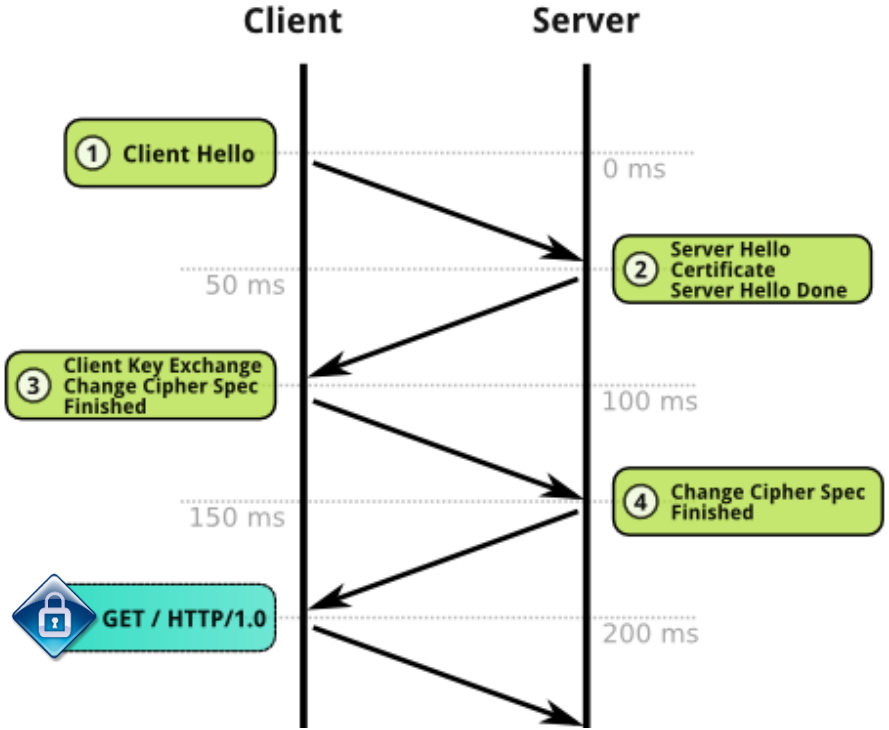
Crucial Parameters, so-called Cipher-Suites

- **Key Exchange Algorithm** (RSA or DH)
  symmetric or asymmetric.
- **Authentication Algorithm** (RSA or DSA)
  X.509 certificates in the case of SSL.
- **Encryption Algorithm** (DES, 3DES, AES, RC4, ...)
  – Used to encrypt the message payload
- **Message Authentication Code** (MD5, SHA-1, ...)
  – Used for message integrity

NSA-Secure Cipher-Suites Combos

- **Key Exchange Algorithm** (**DHE or ECDHE**) – asymmetric or symmetric
- **Authentication Algorithm** (**RSA 4096 or ECDSA**) – X.509 certificates in the case of SSL.
- **Encryption Algorithm** (**AES256**) – Used to encrypt the message payload
- **Message Authentication Code** (**>=SHA256**) – Used for message integrity

Perfect Forward Secrecy (I)

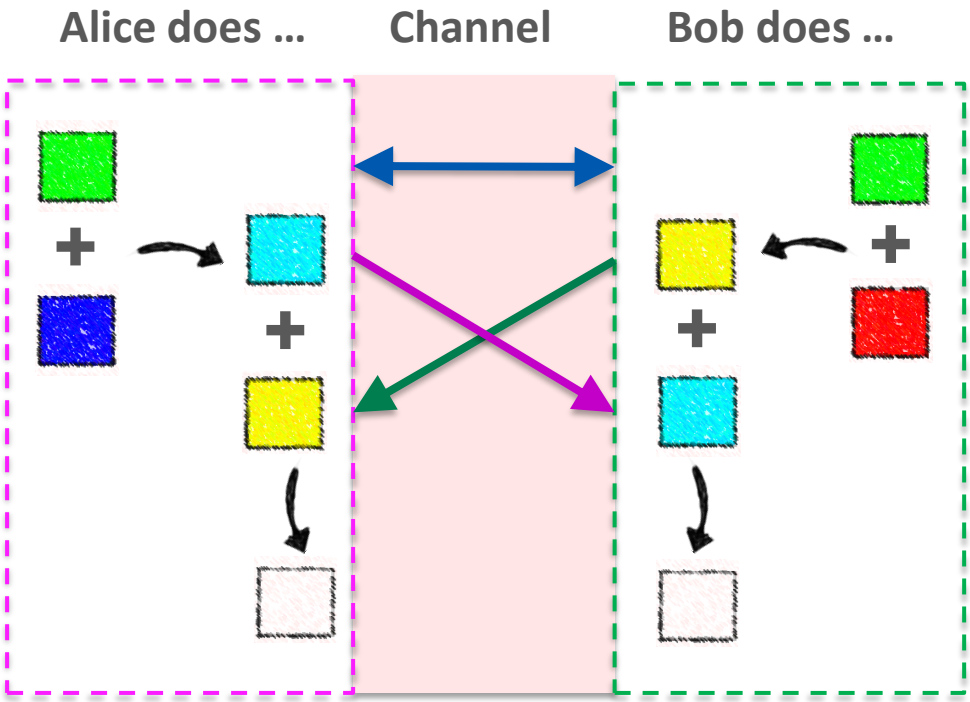


- In a **traditional** TLS **handshake** the authentication key is also the encryption key.
- If the **key** is **compromised** suddenly all **past traffic** can be retroactively **decrypted**.
- This **isn't desirable**
- How can this be **prevented?**

Perfect Forward Secrecy (I)

- The **ability** of a cryptosystem to **remain** retroactively **secure** in face of a key **compromise** is called Perfect Forward Secrecy
- There are two **algorithms for** generating **ephemeral** session keys
  - DHE, **secure** but **slow**
  - ECDHE, **fast** but still **not** widely **supported**

Perfect Forward Secrecy (II)



Diffie-Hellman is a method for the secure **exchange** of cryptographic **keys** over a **public** untrusted channel.

- Agreed common parameter
- Bob's private parameter
- Alice's private parameter
- Shared cryptographic key

Good Deployment Practices (I)

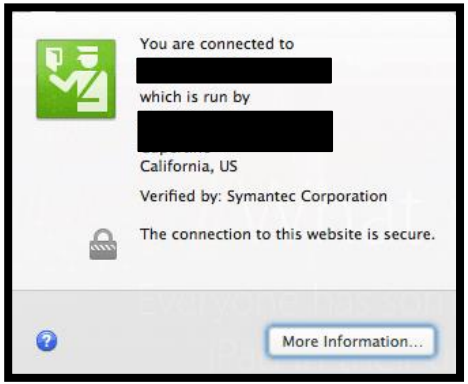
- **Symmetric at least 128** bits long.
- **Asymmetric at least 2048** bit long.
  3000 bit according to the BSI if the keys are intended to be used after 2016.
- Always **plan** your certificate **renovations** in advance and think about **revocations**.
- Think about **your needs**, **don't follow** the **hype**.
  E.g. self-signed certificates are neither good nor bad, it depends.

Good Deployment Practices (II)

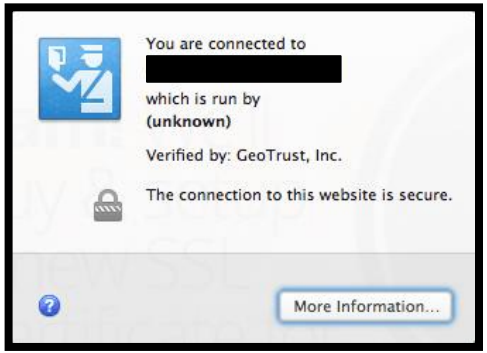
- **Manage** your private **keys diligently**, don't leave them in forgotten network shares
- If **buying certificates** from a CA, **remember** that **cheap** certificates often **become** really **expensive** mistakes.
  - Look into Extended Validation Certificates!
- Remember, **TLS 1.0** and up is **the way to go**.
  - Don't forget to disable TLS compression!

* more at http://datatracker.ietf.org/doc/draft-ietf-uta-tls-bcp/?include_text=1

Good Deployment Practices (III)



- **Educate** your users, no seriously, please do!
- Not all indicators and errors are created equal.



* more at https://docs.google.com/presentation/d/1TNFx6eaQVfe83PV80-FZ39QY1dSLGCWW8f2i5-NeJ48/edit#slide=id.p

SSL Labs, for all your SSL needs

Let's see how https://basta.net does

*SSL labs best deployment practices

The Developer's Role

It's more than just crypto-libraries

You Should Care Because ...

- How you or your **provider deployed** TLS **matters**.
- How secure the **libraries** you use are **matters**.
- The overall **architecture** of your **application matters** and if done wrong can render TLS useless.

In Your Web Applications ...

- Be sure to set the **secure flag** on your cookies
- Only include **external ressources** via **HTTPS** or through protocol-relative URLs
- **Only serve** the application over **HTTPS**
- **Redirect HTTP** directly to the HTTPS site
- Never keep track of a user's identity by means of the URL.
- Are you **serving** or **consuming web-services** over **HTTPS**?

Resource Inclusion, What could go Wrong? (I)

https://www.example.com/banking.aspx?sessionid=jnk34nkjn3k5j2kjn17k

...

<img name="advertisement" height="200" width="300"
src="**http**://externalinclude.net/public/picture.jpg"></img>

...

Resource Inclusion, What could go Wrong? (I)

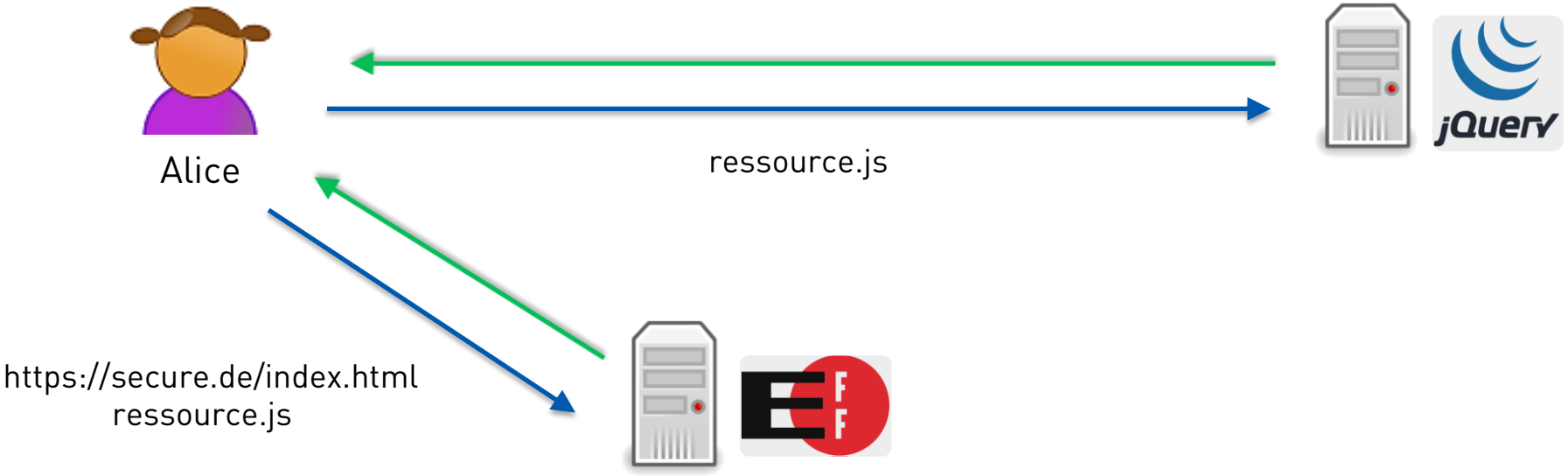
https://www.example.com/banking.aspx?sessionid=jnk34nkjn3k5j2kjn17k

```
...
<img name="advertisement" height="200" width="300"
src="http://externalinclude.net/public/picture.jpg"></img>
...
```

GET /public/picture.jpg
Host: externalinclude.net
Referer: **https**://www.example.com/banking.aspx?sessionid=jnk34nkjn3k5j2kjn17k

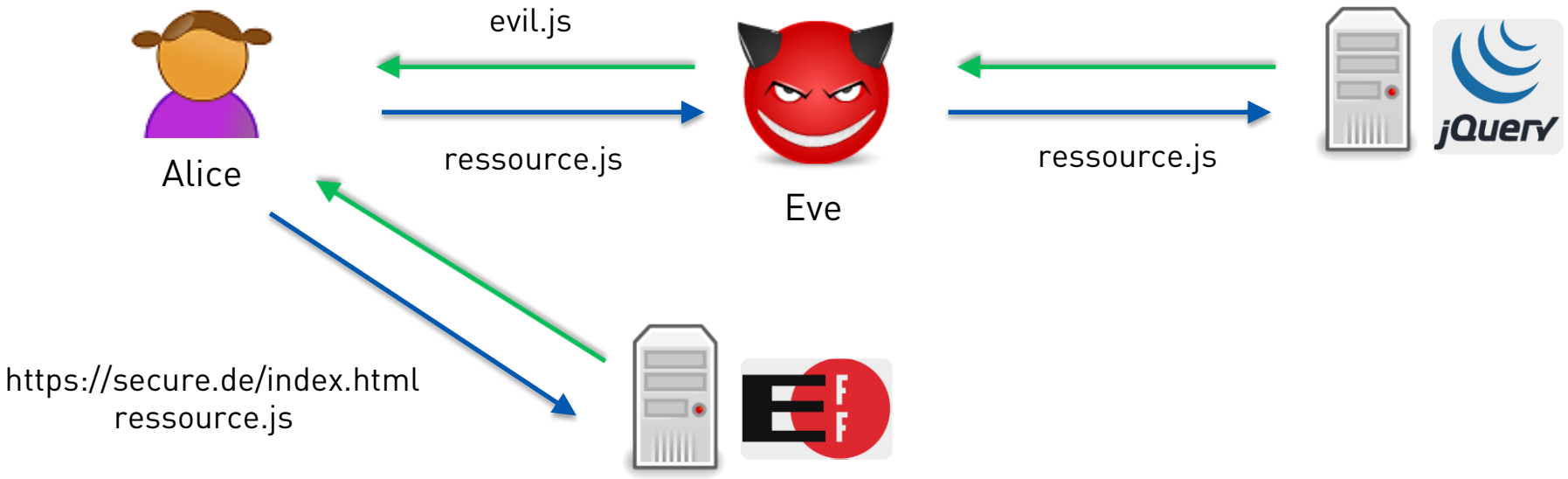
Resource Inclusion, What could go Wrong? (II)

- Content tampering through Man-in-the-Middle Attacks



Resource Inclusion, What could go Wrong? (II)

- Content tampering through Man-in-the-Middle Attacks



In Your Standalone Applications ...

- **Try to always rely on libraries for validation**
- Look into **certificate** and public **key pinning**
- If you still want to implement certificate **validation**
  - Is the **Common Name** valid for the domain?
  - Has the certificate **expired**?
  - Certificate valid for the intended **Usage**?
  - Is the **signature valid**?
  - Certificate **Chain** without **errors**?

The Future of the Secure Web

- By using **HTTP Strict Transport Security** (HSTS) headers a **website** can **request** the browser to **only allow secure** connections.
- Content Security Policy (CSP) enables the **server** to define which kind of resources and from where are to be used.
- **Certificate transparency** initiatives will enable people to held **CAs accountable** for what they sign and for who.

HTTP Strict Transport Security

HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Wed, 25 Feb 2015 07:54:20 GMT
Content-Type: text/html
Content-Length: 178
Connection: keep-alive
Location: https://www.eff.org/
Strict-Transport-Security: max-age=31536000; includeSubdomains

- Instructs the browser to send all **further requests** via **HTTPS**
- **TLS errors** can't be **ignored by users** anymore.
- Popular?

Image taken from: https://www.trustworthyinternet.org/ssl-pulse/

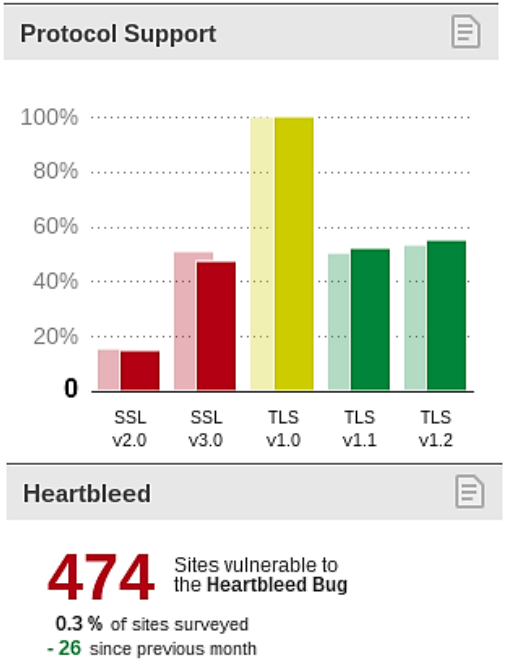
In Light of Recent and not-so Recent Events

The good, the bad and the ugly

The Good News, ☺

- IE **finally** supports HSTS!
- Again, IE supports HSTS!
- **RC4** is officially **deprecated**, RFC7465

The Bad News, ☹



Even if deployed correctly, there's always something that can go wrong.

- **POODLE**, killed SSL 3.0
- **Heartbleed**, well, we all know …

Taken from: https://www.trustworthyinternet.org/ssl-pulse/

The Really Bad News, (ノ°□°)ノ ⌒ ┻━┻

- **Lenovo** and their **traffic interception** and injection tool, „Superfish Visual Discovery“
- **Installed** by Lenovo **on numerous** laptop **models**
- Superfish **generates certificates** for visited sites **on-the-fly** and injects content.
- Is a **highly insecure** software, specially regarding crypto.

Super Ugly News, (ﾉ°□°)ﾉ ⌒┻┻

The Superfish **certificate and** its private **key** are already **floating** on the Internet.

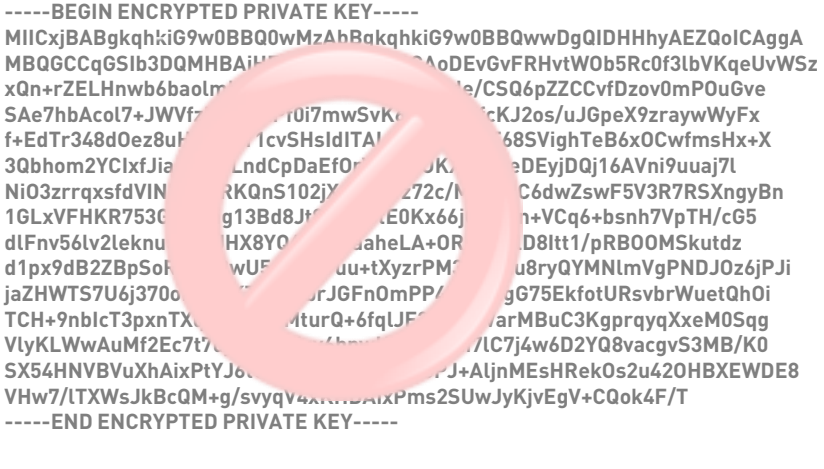
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIICxjBABgkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQIDHHhyAEZQoICAggA
MBQGCCqGSIb3DQMHBAiHEg+MCYQ30ASCAoDEvGvFRHvtWOb5Rc0f3lbVKqeUvWSz
xQn+rZELHnwb6baolmbFcsi6XkacVzL/EF7Ll4de/CSQ6pZZCCvfDzov0mPOuGve
SAe7hbAcol7+JWVfzbnVTblPf0i7mwSvK61cKq7YfcKJ2os/uJGpeX9zraywWyFx
f+EdTr348dOez8uHkURyY1cvSHsIdITALkChOonAYT68SVighTeB6xOCwfmsHx+X
3Qbhom2YCIxfJiaAoz2/LndCpDaEf0rVrxXFOKXrIbmeDEyjDQj16AVni9uuaj7l
NiO3zrrqxsfdVINPaAYRKQnS102jXqkH01z72c/MpMMC6dwZswF5V3R7RSXngyBn
1GLxVFHKR753Gt0IDag13Bd8Jt890/v0tE0Kx66jCkRGn+VCq6+bsnh7VpTH/cG5
dlFnv56lv2leknu5ghdJHX8YQ6HjnioaaheLA+ORAxqAlD8Itt1/pRBOOMSkutdz
d1px9dB2ZBpSoRAOcBwU5aFaw9uu+tXyzrPM3tZomu8ryQYMNlmVgPNDJOz6jPJi
jaZHWTS7U6j370oH/B0KTUG/ybrJGFnOmPP4h2u/ugG75EkfotURsvbrWuetQhOi
TCH+9nbIcT3pxnTXqI2IRHZXMturQ+6fqlJF3bb9bWarMBuC3KgprqyqXxeM0Sqg
VlyKLWwAuMf2Ec7t7ujqaNmVgv6bpwHEbR6njIi7lC7j4w6D2YQ8vacgvS3MB/K0
SX54HNVBVuXhAixPtYJ6tOBGm7QFAKaXju0PJ+AljnMEsHRekOs2u42OHBXEWDE8
VHw7/lTXWsJkBcQM+g/svyqV4xKHDAixPms2SUwJyKjvEgV+CQok4F/T
-----END ENCRYPTED PRIVATE KEY-----

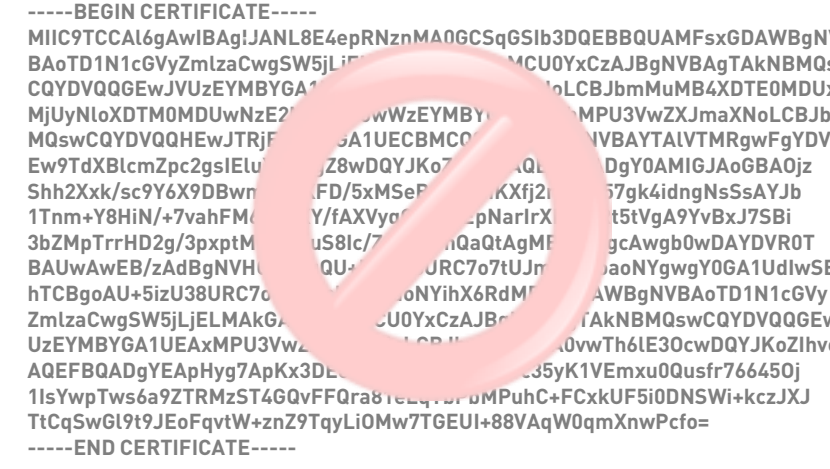
Cert Passphrase: Komodia

-----BEGIN CERTIFICATE-----
MIIC9TCCAl6gAwIBAgIJANL8E4epRNznMA0GCSqGSIb3DQEBBQUAMFsxGDAWBgNV
BAoTD1N1cGVyZmlzaCwgSW5jLjELMAkGA1UEBxMCU0YxCzAJBgNVBAgTAkNBMQsw
CQYDVQQGEwJVUzEYMBYGA1UEAxMPU3VwZXJmaXNoLCBJbmMuMB4XDTE0MDUxMjE2
MjUyNloXDTM0MDUwNzE2MjUyNlowWzEYMBYGA1UEChMPU3VwZXJmaXNoLCBJbmMu
MQswCQYDVQQHEwJTRjELMAkGA1UECBMCQ0ExCzAJBgNVBAYTAlVTMRgwFgYDVQQD
Ew9TdXBlcmZpc2gsIEluYy4wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAOjz
Shh2Xxk/sc9Y6X9DBwmVgDXFD/5xMSeBmRImIKXfj2r8QlU57gk4idngNsSsAYJb
1Tnm+Y8HiN/+7vahFM6pdEXY/fAXVyqC4XouEpNarIrXFWPRt5tVgA9YvBxJ7SBi
3bZMpTrrHD2g/3pxptMQeDOuS8lc/ZJKocPnQaQtAgMBAAGjgcAwgb0wDAYDVR0T
BAUwAwEB/zAdBgNVHQ4EFgQU+5izU38URC7o7tUJml4OVoaoNYgwgY0GA1UdIwSB
hTCBgoAU+5izU38URC7o7tUJml4OVoaoNYihX6RdMFsxGDAWBgNVBAoTD1N1cGVy
ZmlzaCwgSW5jLjELMAkGA1UEBxMCU0YxCzAJBgNVBAgTAkNBMQswCQYDVQQGEwJV
UzEYMBYGA1UEAxMPU3VwZXJmaXNoLCBJbmMuggkA0vwTh6lE3OcwDQYJKoZIhvcN
AQEFBQADgYEApHyg7ApKx3DEcWjzOyLi3JyN0JL+c35yK1VEmxu0Qusfr76645Oj
1IsYwpTws6a9ZTRMzST4GQvFFQra81eLqYbPbMPuhC+FCxkUF5i0DNSWi+kczJXJ
TtCqSwGl9t9JEoFqvtW+znZ9TqyLiOMw7TGEUI+88VAqW0qmXnwPcfo=
-----END CERTIFICATE-----

Super Ugly News, (ﾉ°□°)ﾉ ⌒┻━┻

An **attacker** does **not** even **need** to **sign** anything!
They **failed**, as expected, to implement **certificate validation** right!





Conclusions

- The **way** TLS is **deployed** matters.
- The way **crypto-libraries** are **used** matters.
- **Informed users** and how they react matters.
- The '**crypto-landscape**' of the internet is **complex** with a lot of moving parts.
- Saying "**we do crypto**" is **not enough**, TLS deployments must be assessed.
- Don't stop here, **look at** our **suggested readings**.

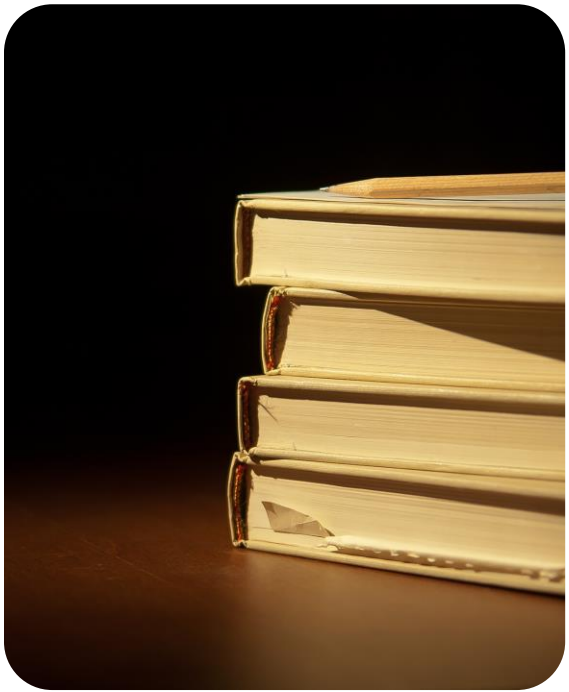
Thank You for Your Time!

Enjoy BASTA!

Jayson Salazar
jsalazar@ernw.de

Dominik Schneider
dschneider@ernw.de

Some Resources for those Interested in More



- The most complete and comprehensible book regarding what we've discussed today: Bulletproof TLS by Ivan Ristic

- https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices.pdf
- https://owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
- www.asp.net/web-api/overview/security/working-with-ssl-in-web-api
- https://msdn.microsoft.com/en-us/library/ff649205.aspx
- https://datatracker.ietf.org/doc/draft-ietf-uta-tls-bcp/?include_text=1
- https://docs.google.com/presentation/d/1TNFx6eaQVfe83PV80-FZ39QY1dSLGCWW8f2i5-NeJ48/edit#slide=id.p
- https://tools.ietf.org/html/draft-mattsson-uta-tls-overhead-01.html
- https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
- http://www.tldp.org/HOWTO/SSL-Certificates-HOWTO/x64.html