CSA2 - Security

Transport Security / Web Application Vulnerabilities

Topics – today

- **Certificates**
- **Transport Security**
  - DNSSEC
  - HTTPS / TLS / SSL
  - Forward Secrecy
- **Top 10 Web Security Risks (OWASP)**
  - SQL-injection
  - CSRF
  - XSS

Message Authentication Code (MAC)

- **Short piece of information which ensures the message has not been altered.**
- No Encryption
- Needs a shared key
- Only provides basic authenticity

Hash-based MAC (HMAC)

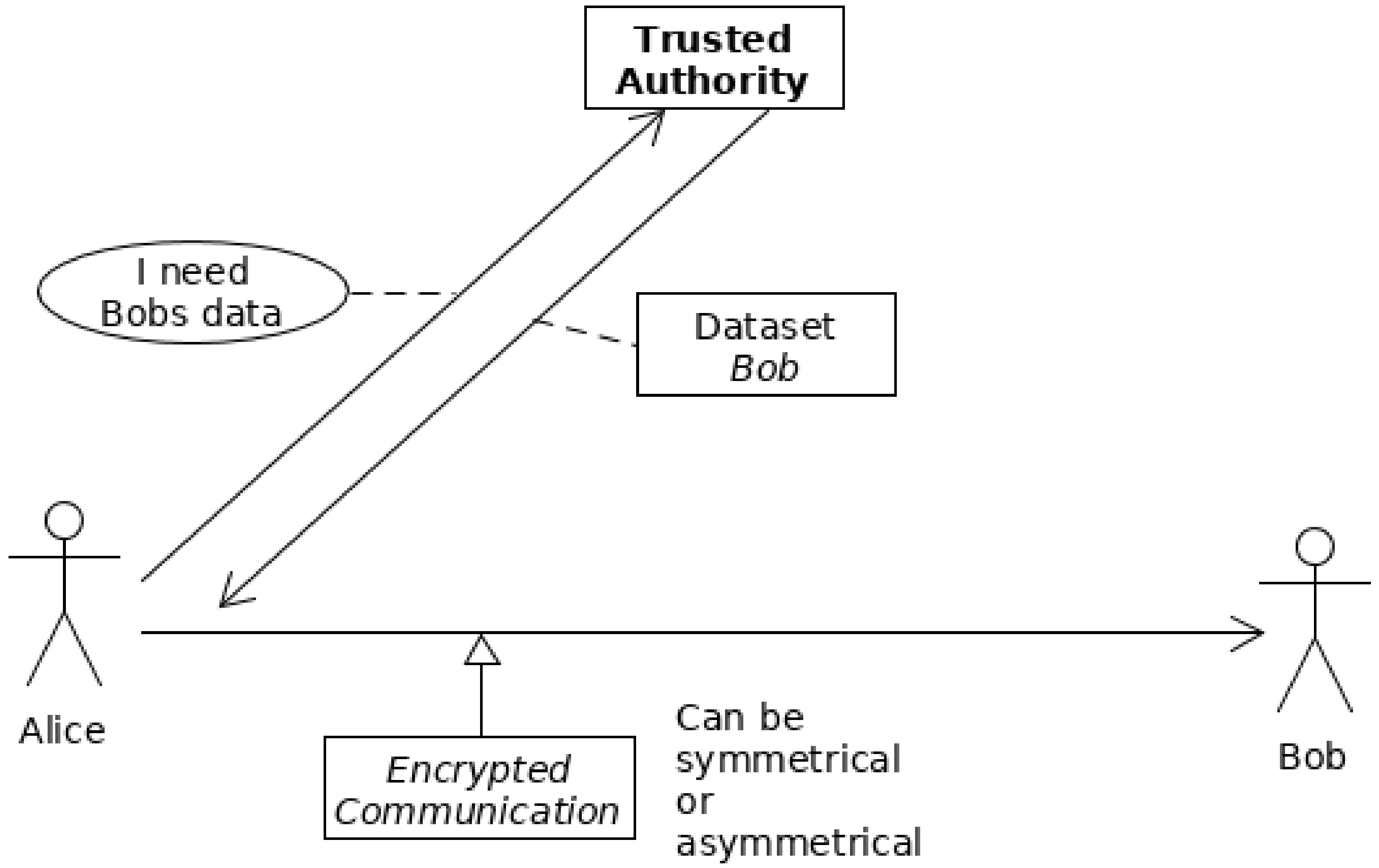
- **Hash concatenation of shared key and message**
  - Uses special concatenation method to prevent attacks
- **Send hash along with the message**

Can this be done with public / private keys?

Problems with Keys

- **How to interchange keys on unsafe channel**
  - How do you know the key belongs to the person you want to communicate with?
  - How do you know the key was not changed?
- **How to communicate with someone I never talked to before?**
  - No shared key exchange
  - No public key exchanged

Trusted Authorities



Certificates

- Before key negotiation you need to know you are talking to the right person / computer.
- **A digital certificate is an electronic document used to prove the ownership of a public key.** [Wikipedia]
- **Data structure to add information to a shared / public key about owner.**
- Needs a trusted authority → Certificate Authority (**CA**)

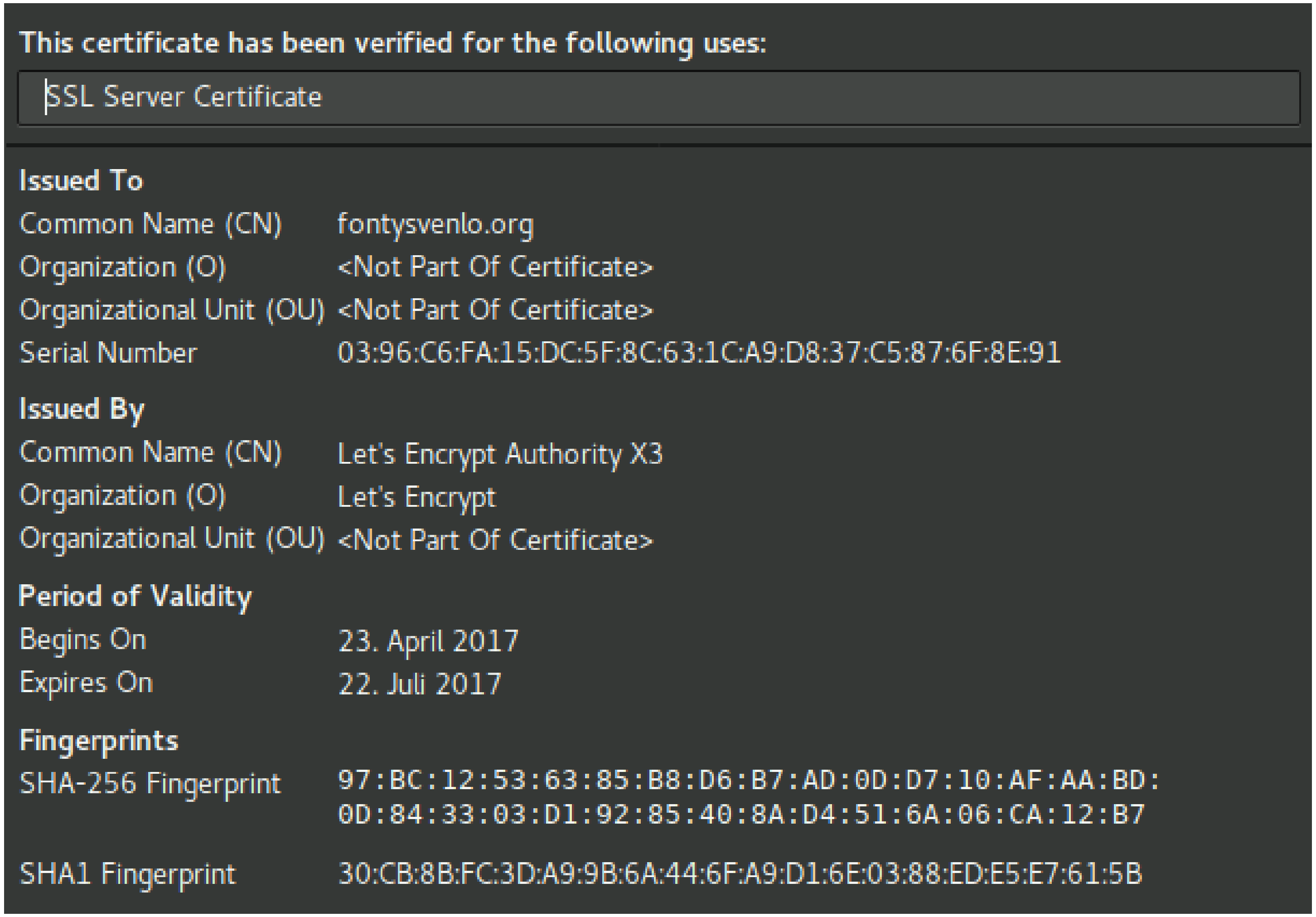
Root-Certificates

- Root certificate is a **self-signed** public key of a CA
- Can sign multiple certificates which can be used to sign other certificates. (**intermediate certificates**)
- Distributed using **public-key infrastructure** (PKI)
  - Needs a safe channel
  - Included in Operating System
  - Included in Browser
  - Comes with third-party software

X.509 Certificates

- **Standard used in SSL / TLS**
- **Certificate Revocation List (CRL)**
- **Important data fields:**
  - Common Name
  - Public Key & Algorithm
  - Signature of CA & Algorithm
  - <u>Lifetime</u> (start & end)
  - Owner´s personal information

X.509 Certificate - Example



This certificate has been verified for the following uses:

SSL Server Certificate

Issued To

Common Name (CN)	fontysvenlo.org
Organization (O)	<Not Part Of Certificate>
Organizational Unit (OU)	<Not Part Of Certificate>
Serial Number	03:96:C6:FA:15:DC:5F:8C:63:1C:A9:D8:37:C5:87:6F:8E:91

Issued By

Common Name (CN)	Let's Encrypt Authority X3
Organization (O)	Let's Encrypt
Organizational Unit (OU)	<Not Part Of Certificate>

Period of Validity

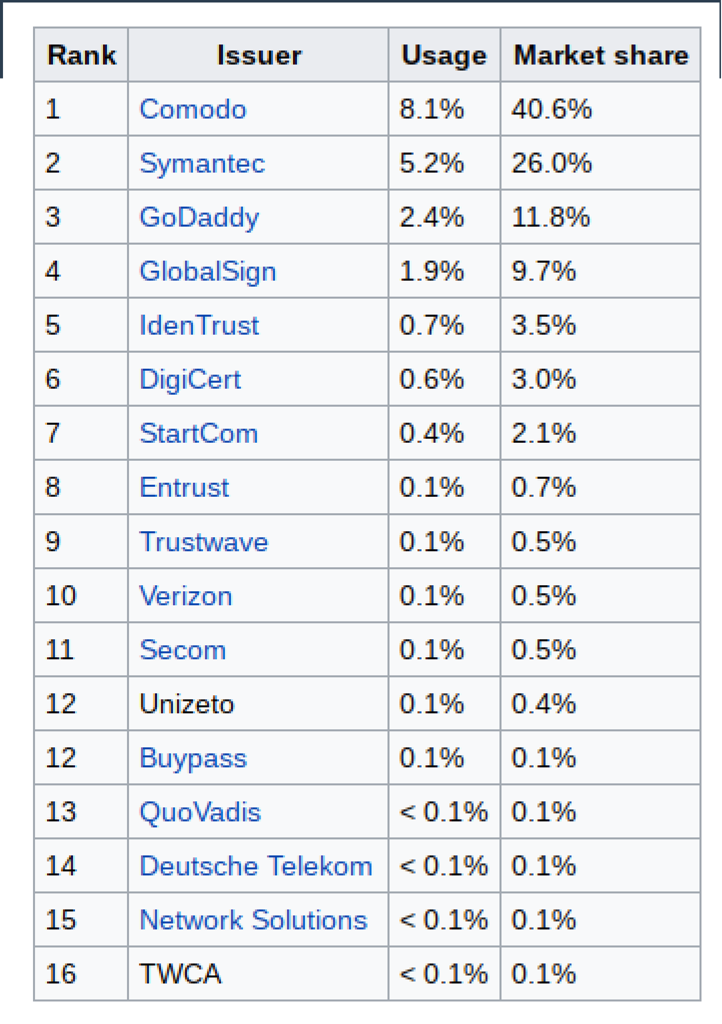
Begins On	23. April 2017
Expires On	22. Juli 2017

Fingerprints

SHA-256 Fingerprint	97:BC:12:53:63:85:B8:D6:B7:AD:0D:D7:10:AF:AA:BD: 0D:84:33:03:D1:92:85:40:8A:D4:51:6A:06:CA:12:B7
SHA1 Fingerprint	30:CB:8B:FC:3D:A9:9B:6A:44:6F:A9:D1:6E:03:88:ED:E5:E7:61:5B

Certificate authorities

- **Signs certificates of others**
  - Certificate Signing Request (CSR)
- **Needs a root certificate**
- **Usually charges a fee**
- **<u>Required</u> to check the correctness of the CSR by validating the ownership.**



Rank	Issuer	Usage	Market share
1	Comodo	8.1%	40.6%
2	Symantec	5.2%	26.0%
3	GoDaddy	2.4%	11.8%
4	GlobalSign	1.9%	9.7%
5	IdenTrust	0.7%	3.5%
6	DigiCert	0.6%	3.0%
7	StartCom	0.4%	2.1%
8	Entrust	0.1%	0.7%
9	Trustwave	0.1%	0.5%
10	Verizon	0.1%	0.5%
11	Secom	0.1%	0.5%
12	Unizeto	0.1%	0.4%
12	Buypass	0.1%	0.1%
13	QuoVadis	< 0.1%	0.1%
14	Deutsche Telekom	< 0.1%	0.1%
15	Network Solutions	< 0.1%	0.1%
16	TWCA	< 0.1%	0.1%

Domain Name System Security Extensions

- **Domain Name System is vulnerable to manipulation**
  - Protocol does not support authenticity
- **DNS is usually cached in OS which can be attacked.**
- Still no encryption of DNS requests
- Owner signs every DNS entry
- Uses private / public keys

Transport Security

- **Compression**
- **Encryption**
- **Secure Socket Layer (SSL)**
- **Transport Layer Security (TLS)**

SSL = TLS? SSL = TLS!

Is there something you cannot encrypt?

SSL & TLS

- **SSL (deprecated)**
  - 1.0 → 1994
  - 2.0 → 1995
  - 3.0 → 1996
- **TLS**
  - 1.0 → 1999
  - 1.1 → 2006
  - 1.2 → 2008
  - 1.3 → 2016 (Draft)

HTTPS

- **HTTP over TLS**
  - HTTP over SSL
  - HTTP Secure
- **Certificates for server authentication**
- **Asymmetrical Encryption for Key Exchange**
- **Symmetrical Encryption for communication**
- **HMAC for message integrity**

HTTPS Connections

- **Handshake (TLS Handshake Protocol)**
  - What ciphers does the client support (Client Hello)
  - What ciphers does the server support (Server Hello)
  - Server’s certificate
  - Client’s certificate (optional)
  - Key Exchange (can be via Diffie-Hellman-Method)

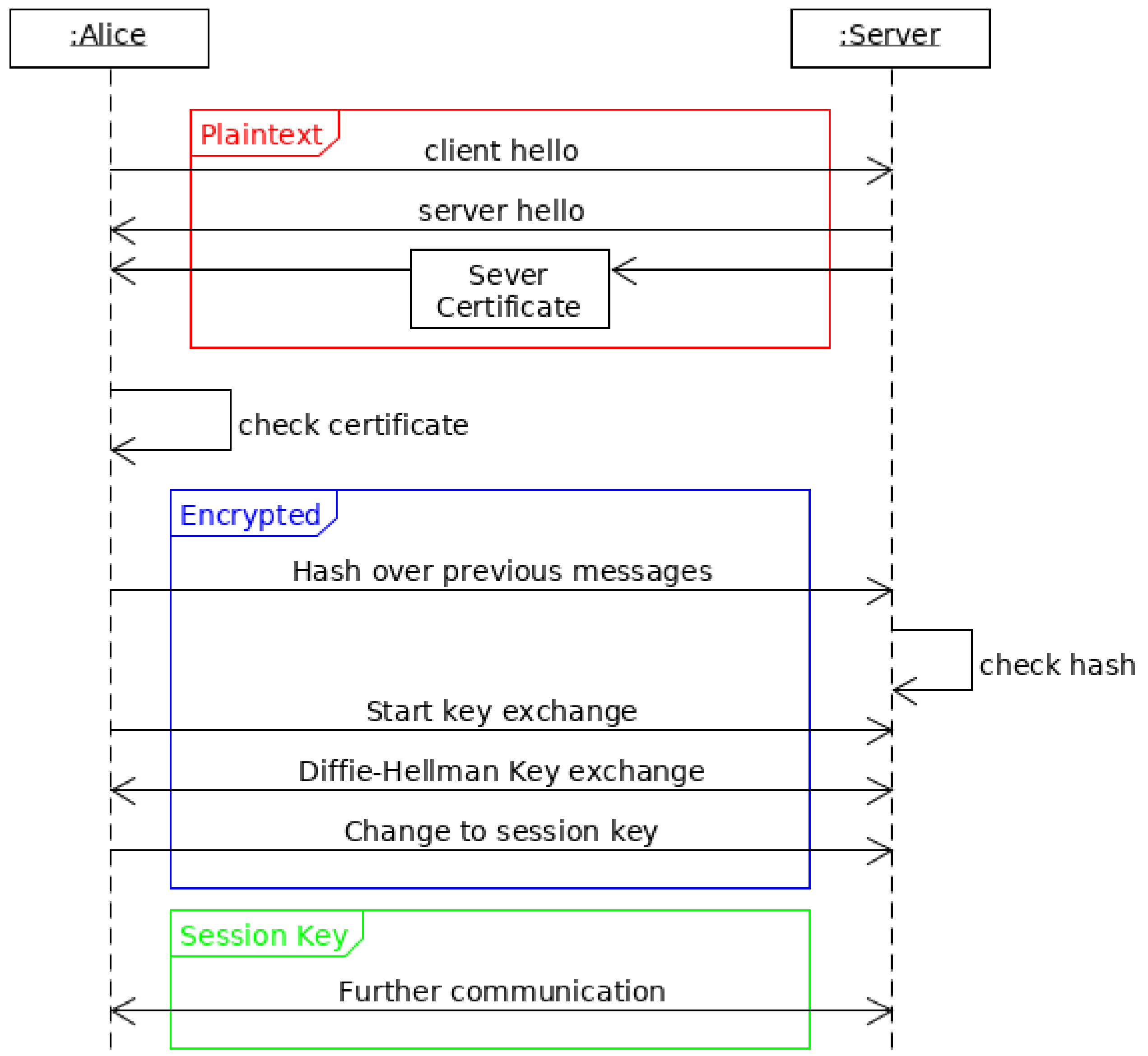
    => *Master Secret*
- **Payload (TLS Record Protocol)**
  - Use Master Secret for symmetrical encryption

Forward Secrecy (FS)

- Decode past communication if private key is compromised or encryption get broken.
- NSA and other agencies store encrypted communication in order to decrypt them in the future.
- Achieved with **random session keys**

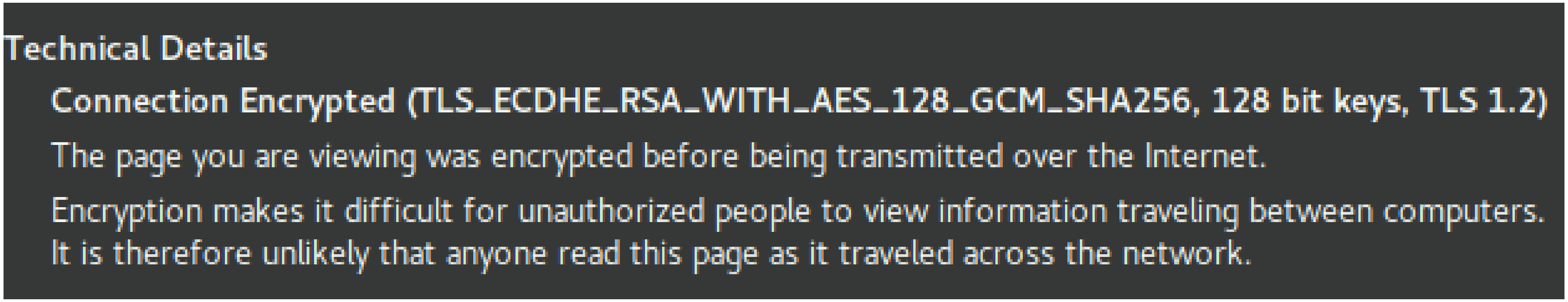
A public-key system has the property of forward secrecy if it generates one random secret key per session to complete a key agreement, without using a deterministic algorithm. [Wikipedia]

Forward Secrecy & TLS



Cipher Suites

- **Collection of algorithms used to secure the current connection.**
  - Key exchange algorithm
  - Encryption algorithm
  - Hash algorithm for HMAC



Technical Details

Connection Encrypted (TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, 128 bit keys, TLS 1.2)

The page you are viewing was encrypted before being transmitted over the Internet.

Encryption makes it difficult for unauthorized people to view information traveling between computers. It is therefore unlikely that anyone read this page as it traveled across the network.

Disadvantages

- **Handshake takes a lot of time**
- **Handshake has to take place every new session**
- **Encryption needs CPU resources**
- **Bit Errors affect whole document**
- **Certificates costs money**
  - Expensive for a personal website
- **Security software causes severe problems**

Should I use HTTPS anytime?

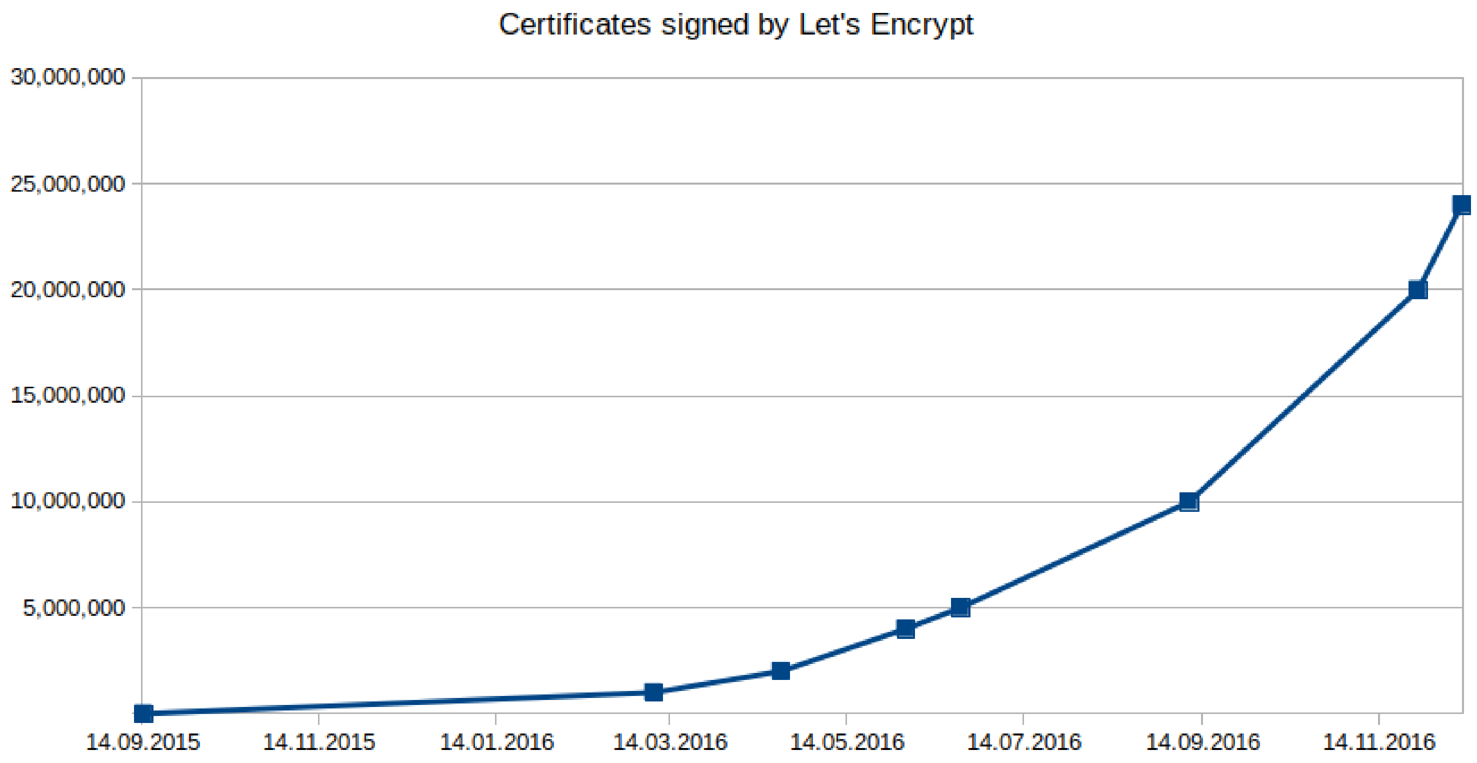
YES!

Why to use HTTPS?

- **Encrypted communication**
- **<u>Authenticated communication</u>**
  - Server Authentication
  - Message Authentication

Let’s Encrypt Project

- Founded 2014
- Public Beta 12/2015
- Start service 04/2016
- Certificates only valid for 3 month



HTTP Strict Transport Security

- **HSTS**
- Forces browser to use HTTPS in future
  - Limited for a specific time span (max-age)
- Browser fails connection of HTTPS is not available or certificate is invalid.
- **Needs to be enabled on server-side**
  - HTTP Header: Strict-Transport-Security

Check your configuration

- Check your server configuration for known vulnerabilities and activated prevention measures.

https://observatory.mozilla.org/

https://www.ssllabs.com/ssltest/

Web Application Security

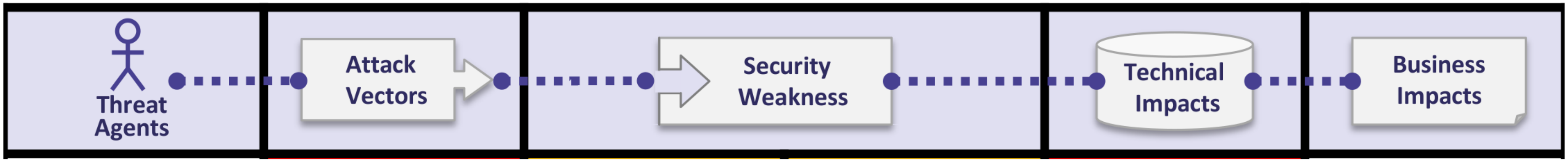
- **Most critical web application security risks**
- Examples are given using PHP and Javascript
- **Not dependent to language or framework**
  - Risks can be equally applied to other languages:
    - Ruby on Rails
    - Python & Django
    - Java EE / Spring / Vaadin

Open Web Application Security Project (OWASP)

- Project lines out the 10 most important web application security risks.
- Last Report published 2013
- New Report available as release candidate

Excerpt:
A1 – (SQL)-Injection
A2 – Broken Authentication
A3 – Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)
A9 – Components with Known Vulnerabilities

Security Risk classification



- **Threat Agent**
  - Who is able to perform an attack (insider, outsider, ...)
- **Attack Vector**
  - How is the attack performed / What is done
- **Security Weakness**
- **Technical impact**
  - Impact on technical systems / *Availability*
- **Business impact**
  - Impact on business value / *Confidentiality & Integrity*

SQL - Injection

- **Manipulated input will be sent to database as SQL statements.**
- **Happens when input data is directly integrated into a SQL query**
- Can be prevented by properly escaping input data.
- Can be prevented by using prepared statements.
- Impact can be reduced by using unprivileged user.

SQL – Injection - Example

```
SELECT * FROM user WHERE
    username = ‘$user’ AND password = ‘$password’
```

```
$user = “tobias’ -- ”

SELECT * FROM user WHERE
    username = ‘tobias’ -- ’ AND password = ‘$password’
```

```
$user = “’; DROP TABLE user; -- ”

SELECT * FROM user WHERE
    username = ‘’; DROP TABLE user;
        -- ’ AND password = ‘$password’
```

Other Injections

- **Code Injection**
  - Script interprets external input as code

Can happen when using unsafe methods:

eval()

Method interprets any string as code

Broken Authentication

- **Authentication is implemented incorrectly**
- **Attackers can exploit session, passwords or authentication tokens**
- Insecure storing of passwords
- Session IDs are exposed in the URL
- Credentials are sent unencrypted

Cross-Site Scripting (XSS)

- **Execute external code on browser**
- **Happens when input data is directly printed into HTML**
- **Attacker can get authentication cookies**
- **Attacker can load external scripts**
- Javascript is somehow integrated into the request
- Can be prevented by properly escaping input data.

Cross-Site Request Forgery (CSRF)

- **An attacker tricks the browser so send a HTTP request to a target site.**
- **Can be done via images**
- **Browser sends authentication automatically**
- **Problem: HTTP is stateless**

Components with Known Vulnerabilities

- **Use of old components with known vulnerabilities**
  - Libraries
  - Frameworks
  - Runtimes
- **High risk when using open-source software**
  - Wordpress / Joomla / Typo3 / etc...

Other Security Risks

- A4 – **Broken Access Control**
  - Authentication restricts are not properly enforced
- A5 – **Security Misconfiguration**
  - Attack prevention measure, safe passwords, unprivileged users
- A6 – **Sensitive Data Exposure**
  - Weakly protected sensitive data
- A7 – **Insufficient Attack Protection**
  - Lack of capabilities to detect and prevent attacks
- A10 – **Underprotected APIs**
  - Application uses API without access control

What should I do?

Never trust user´s input!

- **Check every external input**
  - Data type
  - Content
- **Escape all external input properly**
- **Avoid naturally dangerous methods**
- **Enable / implement prevention measures**

Questions?

Examples

- Examples available on Github: https://github.com/lukeelten/csa2-examples

- OWASP Top 10 RC 1 (10-4-2017):

  https://github.com/OWASP/Top10/raw/master/2017/OWASP%20Top%2010%20-%202017%20RC1-English.pdf