Cryptography

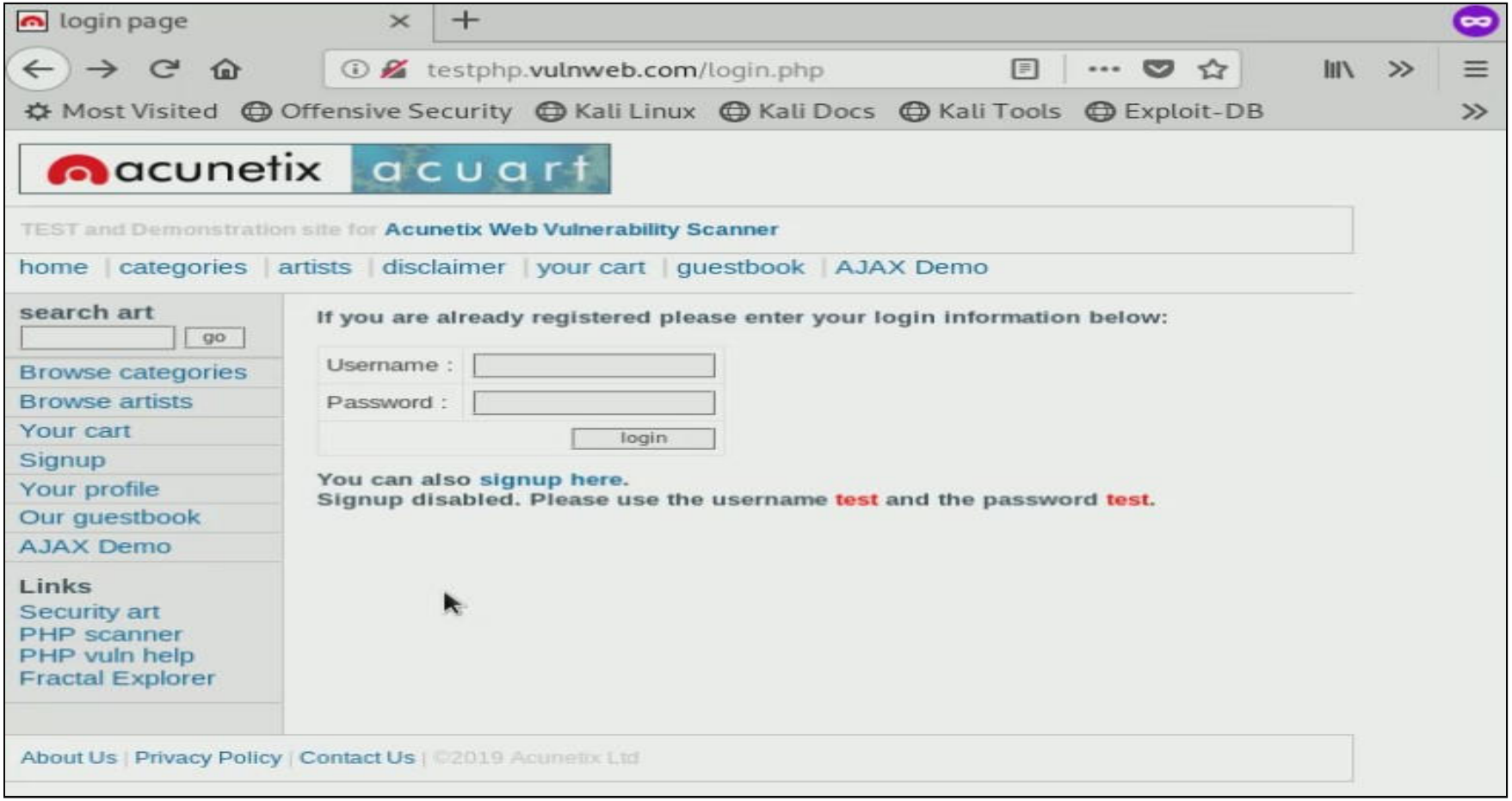
An introduction to OpenSSL, PGP and Let's Encrypt

A Brief intro to cryptography

- Need: To protect applications communication
- Goals: data confidentiality, data integrity, authentication, and non-repudiation
- Network-based attacks: eavesdropping, IP spoofing, connection hijacking, and tampering

Not easy to use cryptographic algorithms in a secure and reliable manner !

Why cryptography is important?



Why we need SSL/TLS?

- Cryptographic protocols are difficult to implement
- Not easy to use cryptographic algorithms in a secure and reliable manner
- The algorithms are just the building blocks in the protocols
- Cryptographic protocols need to cover and resist all known attacks
- Attackers can perform tampering to data
- Many cryptographic protocols have limited applicability
- SSL makes the security of network connection easier

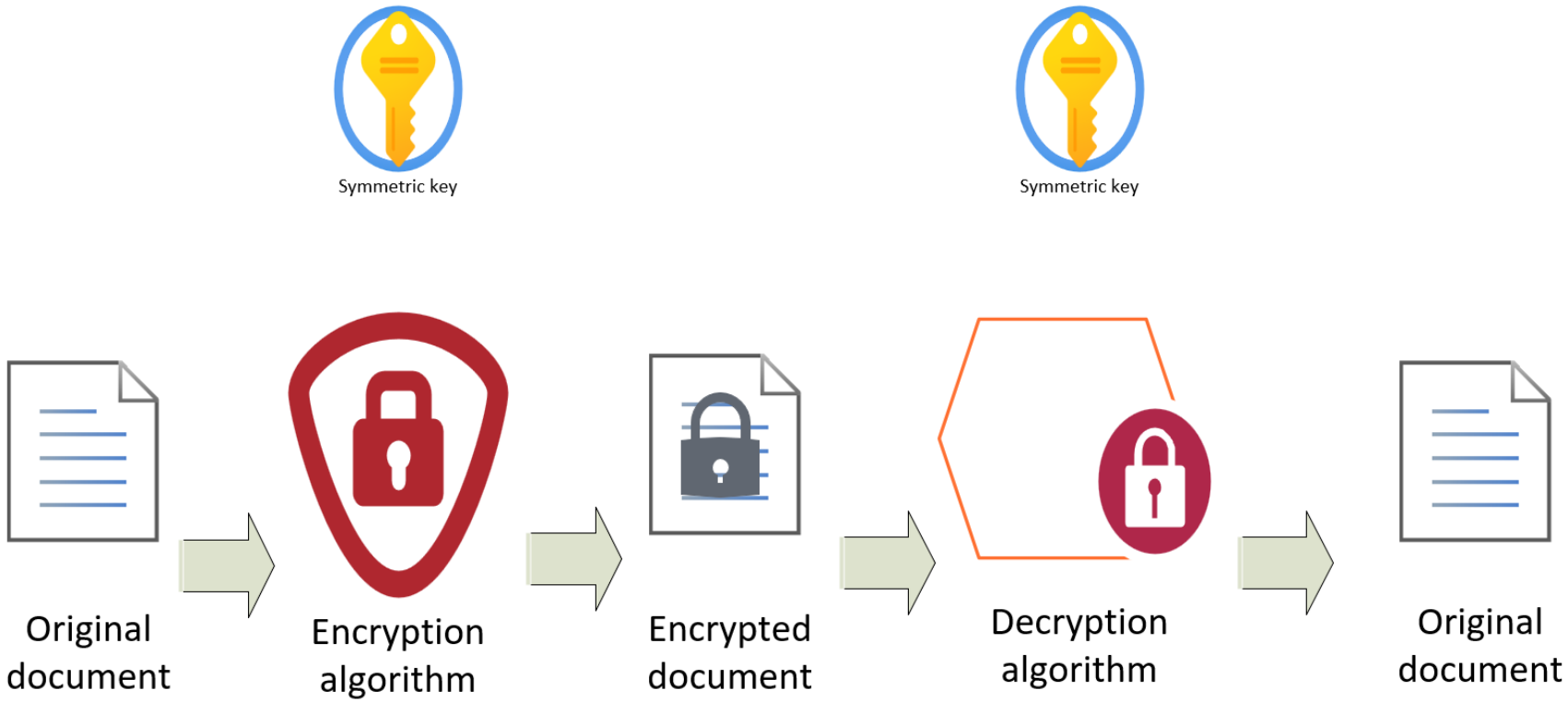
SSL/TLS →

provide nowadays the most common security services for TCP-based connection

Adds transparent confidentiality authentication and integrity to TCP connections

University of BRISTOL

Types of cryptographic algorithms: symmetric key encryption

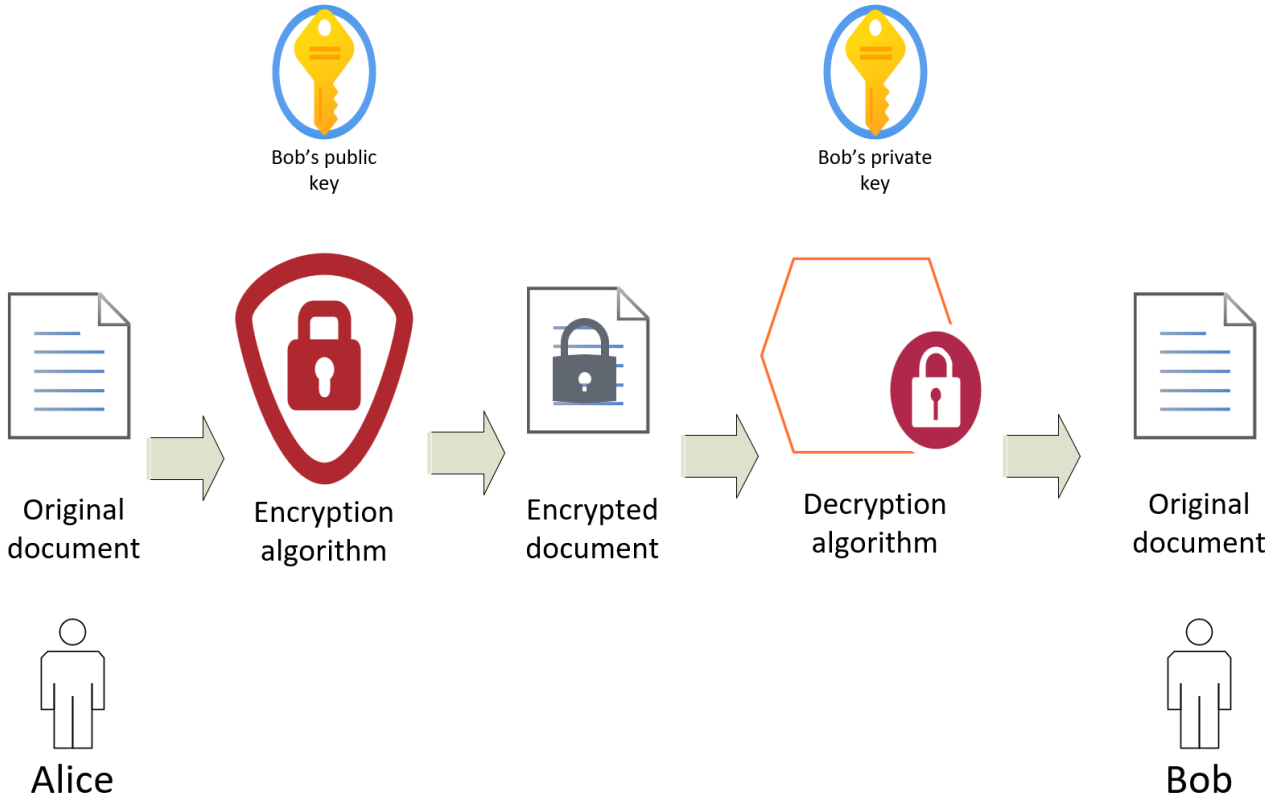


Advantages & Disadvantages of symmetric keys

- **Efficient & faster:** suitable for large amount of data (streaming)
- **Simpler:** it includes less computational steps and effort
- **More suitable for embedded systems and IoT industrial devices:** in some case where we have resource-constrained environments
- **Single Point of Failure:** Encrypts and decrypts only with a single key and must keep safe!
- **Limited Authentication**: Only the one with the key can decrypt the message
- **Key Management:** Revoke, rotation hard in large environments with many users

University of BRISTOL

Types of cryptographic algorithms: asymmetric key encryption (Public key encryption)



Advantages & Disadvantages of Public keys

- **Distribution**: use of public key only (relies on trust in the authenticity of public keys)
- **Non-Repudiation**, with certificates verify the authenticity and integrity of messages
- **Authentication:** third parties can validate certificates sent with public keys

- **Slow**: For large messages typically slower and more computationally intensive
- **Not for large data**: used for key exchange protocols
- **Key size**: Produced keys significantly larger than symmetric keys(increased bandwidth and storage requirements)

University of BRISTOL

Types of cryptographic algorithms: Cryptographic hash functions

- These are checksum algorithms (i.e. MD5 128bits SHA1 160bits → safer)
- Hash functions converts data into a fixed-size checksum (message digest)
- Any change to the data gives different output (tampering)
- The output reveals no info about the data
- Impossible to find two inputs to produce same checksum
- Practically impossible to algorithmically reconstruct the input (one-way)
- Output twice as large as the symmetric key algorithm

University of BRISTOL

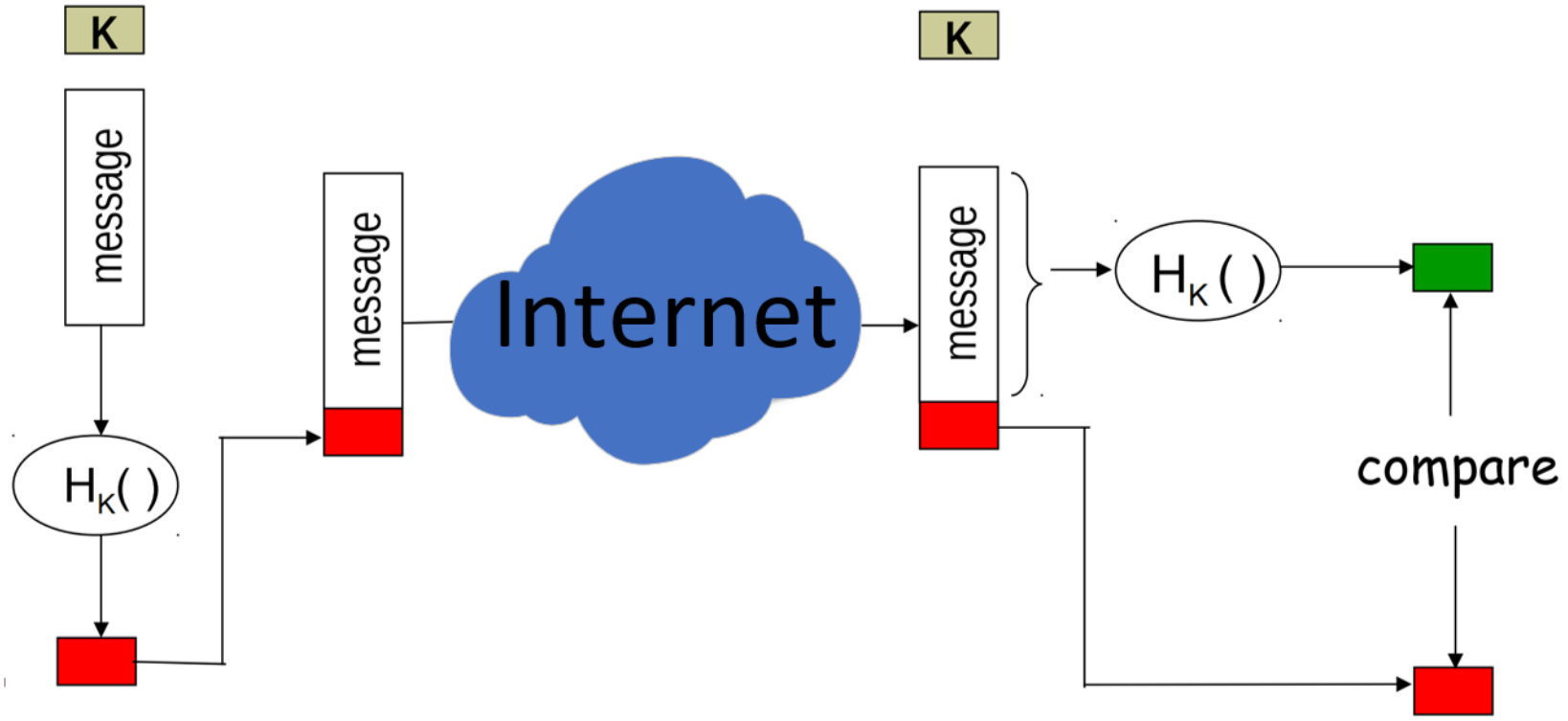
Hash functions demo

- Password storage solution
- Protect software release

```
import hashlib

message = input("Enter the message to hash with md5: ")
md5 = hashlib.md5(message.encode())
print ("Hash message with SHA1 128 bits: "+ str(md5))

message = input("Enter the message to hash with sha1: ").encode('utf-8')
sha = hashlib.sha1(message)
sha1 = sha.hexdigest()
print ("Hash message with SHA1 160 bits: "+ sha1)

message = input("Enter the message to hash with sha256: ").encode('utf-8')
sha = hashlib.sha256(message)
sha256 = sha.hexdigest()
print ("Hash message with SHA1 256 bits: "+ sha256)

message = input("Enter the message to hash with sha512: ").encode('utf-8')
sha = hashlib.sha512(message)
sha512 = sha.hexdigest()
print ("Hash message with SHA1 512 bits: "+ sha512)
```

University of BRISTOL

Types of cryptographic algorithms: Message Authentication Codes (MACs)

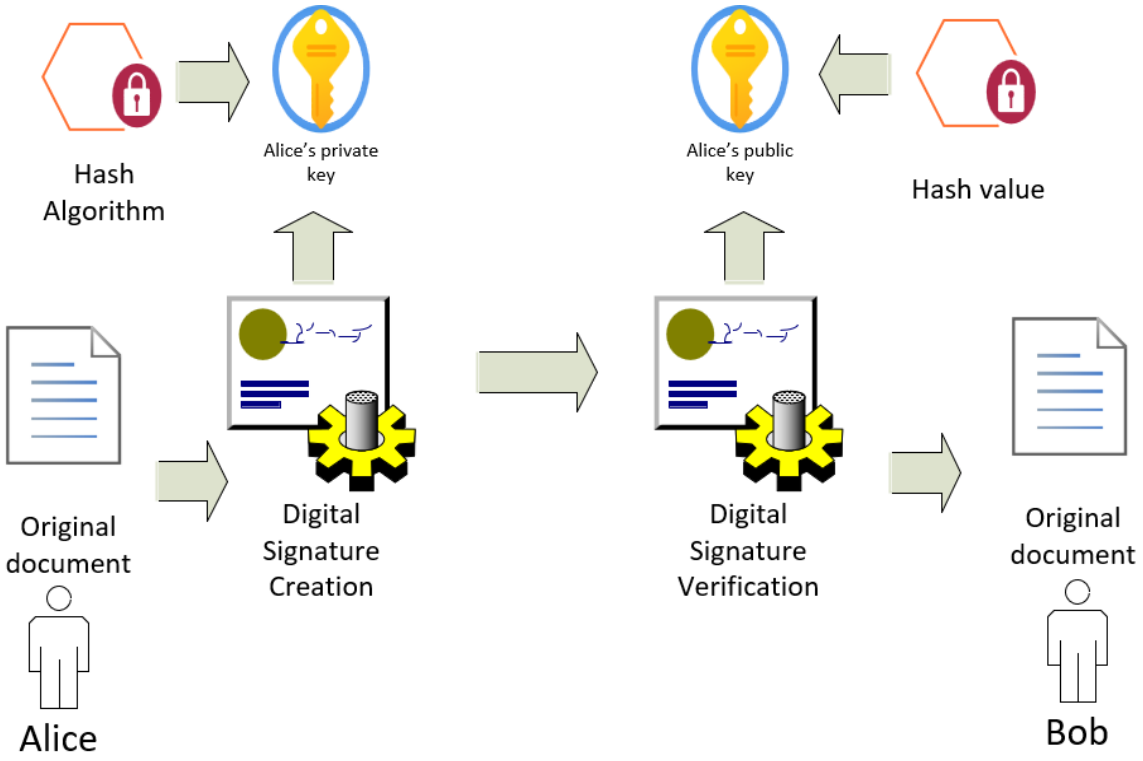


K = secret key, MAC is supported in SSL and in OpenSSL as only HMAC
Ensure integrity for the "***message digest***"

University of BRISTOL

Types of cryptographic algorithms: Digital signatures

- A form of public key cryptography
- Used to provide digital identity authentication and encryption
- Public key and private key are interchangeable
- Digital signatures are very slow
- Use 1,024 bits or higher to ensure security
- Sender signs a message with the secret private key
- Receiver use the sender's public key to verify that the sender signed the message



University of BRISTOL

Encryption Algorithms advantages & disadvantages

- Data Encryption Standard (DES)
  - It was one of the first widely used but not longer considered secure because it holds small key size
- Triple DES (3DES)
  - Effective as it applies the DES algorithm three times (168-bit key), but it consumes much more time than other
- Advanced Encryption Standard (AES)
  - Strong security, efficient in terms of computational resources and memory usage
  - Flexible as it supports many key lengths but vulnerable to side-channel attacks
- RSA (Rivest-Shamir-Adleman)
  - Widely used to secure key exchange, digital signatures, and public key encryption
  - It has built-in mechanisms for non-repudiation through digital signatures
  - To resist attacks large RSA keys, consumes more time for encryption
- Elliptic Curve Cryptography (ECC)
  - A strong security algorithm with small key sizes.
  - Efficient in terms of bandwidth and computational resources (IoT devices)
  - It has Implementation complexity, and need careful implementation

University of BRISTOL

How to select the key lengths

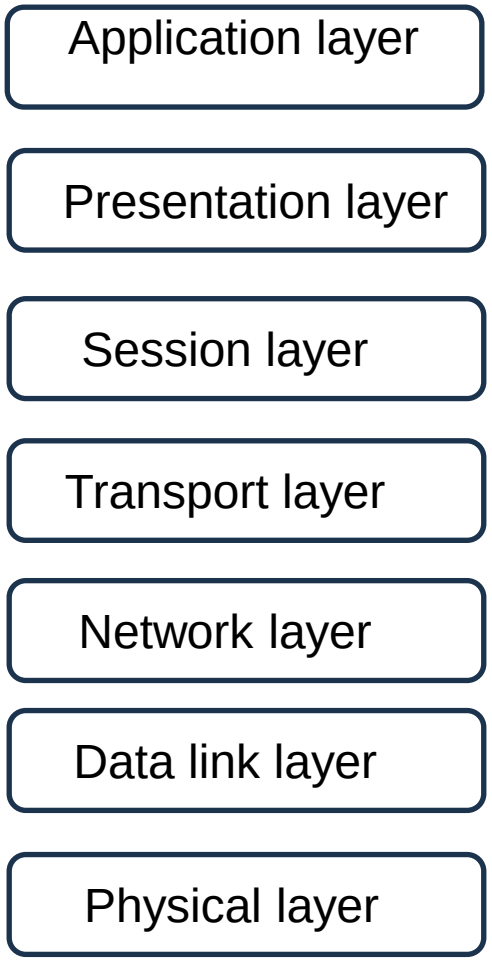
- Consider the Encryption Algorithm:
  - length of keys in public key are large numbers comparable to symmetric algorithms
  - 512-bit keys too weak, 2,048 bits may be too slow
- Security Requirements:
  - Sensitivity of the data being encrypted and potential threats
- Lifespan of the Data:
  - Consider for how long you need your data to remain secure (longer key lengths?)
- Regulation:
  - Ensure to comply with regulatory requirements with minimum key lengths (ISO 27001)
- Maintain balance:
  - Longer keys provide more security but may increase computational overhead and give slower performance

https://www.sciencedirect.com/science/article/abs/pii/S1363412710000312

University of BRISTOL

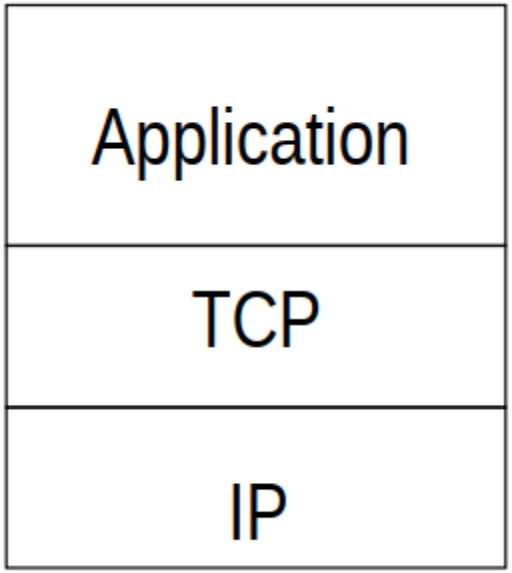
Overview of SSL/TLS

- SSL is a widely deployed security protocol (HTTPS)
- Secures any protocols over TCP
- Client sends a handshake to the server and the server in the response sends the certificate

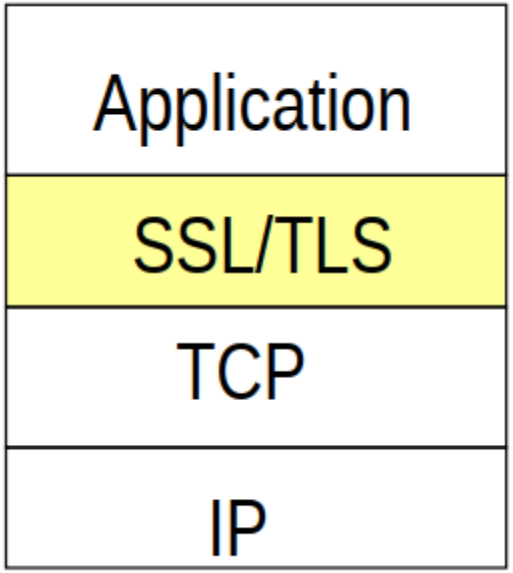


University of BRISTOL

SSL/TLS and TCP/IP



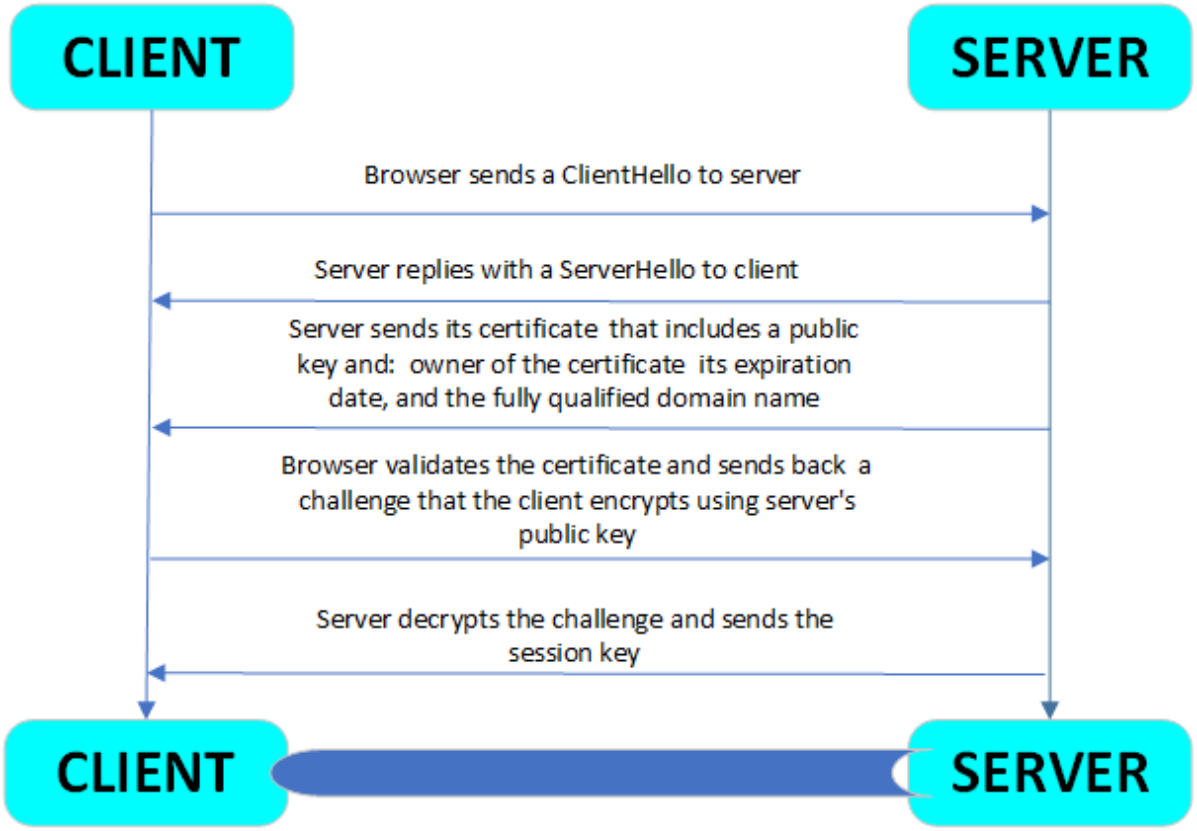
Application



Application with SSL/TLS

University of BRISTOL

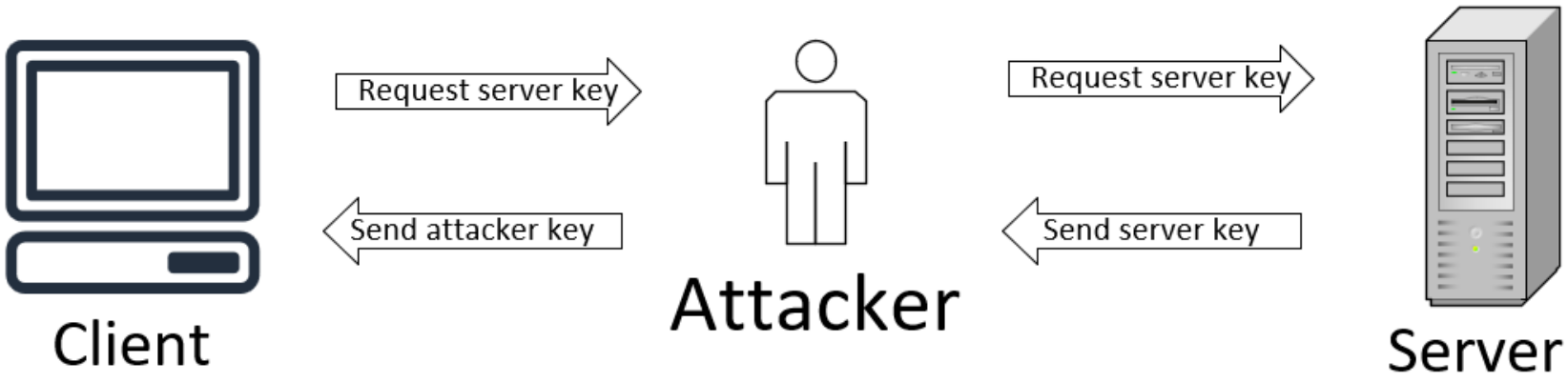
SSL/TLS Handshake



University of BRISTOL

MitM attack for SSL/TLS

- The attacker needs a copy of the certificate and a private key to masquerade as a known server
- Attacker can sniff the server messages and present the attacker's certificate
- The forged certificate can look like legitimate
- Man-in-the-middle (MitM) attack where the attacker eavesdrops on all communication



University of BRISTOL

What SSL/TLS doesn't do well?

- Using SSL is slower than an HTTP connection (handshake with public key)
- Overhead of encrypting and decrypting the data
- Doesn't work with transport layer protocols not connection-oriented, such as UDP
- SSL has no support for non-repudiation (what if the other party attach a message with invalid signature?)
- SSL doesn't protect against flaws in the application itself i.e. buffer overflow
- SSL cannot protect data before it is sent but only data in transit

Using SSL and the OpenSSL library

- OpenSSL is a cryptographic library able to implement many encryption algorithms, such as DES, AES and RSA
- OpenSSL used to be SSLeay created by Eric A. Young and Tim J. Hudson
- beginning in 1995
- OpenSSL first version was released as 0.9.1c in 1998
- OpenSSL is a cryptographic library and an SSL toolkit
- The SSL library provides all versions of SSL alongside with TLS
- Supports the most popular algorithms for symmetric and public key and hash algorithms
- OpenSSL a free SSL implementation and works on Unix Oss and Windows
- It has a feature of pseudorandom number generator (increase entropy)


```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

OpenSSL Files

- .KEY

  A file that has the private key

- .CSR (Certificate Signing Request)

  A file that is sent to the Certificate Authority with information and needs the private key

- .CRT (Certificate abbrev)

  It is the security certificate file, created to establish secure connections

- .PEM (Privacy Enhanced Mail)

  May include the public certificate or an entire certificate chain (public key, private key, certificate)

- .CRL (Certificate revocation list)

  A file used to de-authorize certificates before expiration

University of BRISTOL

Generating Public and Private Keys

Generating RSA Keys

Generate 2048 bit RSA Private Key saved as KEY1.pem

```
openssl genrsa -out KEY1.pem 2048
```

Generate 4096 bit RSA Private Key, encrypted with AES128

```
openssl genrsa -out KEY2.pem -aes128 4096
```

- Key size must be last argument of command
- Omit `-out <FILE>` argument to output to StdOut
- Other encryption algorithms are also supported:
  `-aes128, -aes192, -aes256, -des3, -des`

Generating DSA Keys:

Generate DSA Parameters File

```
openssl dsaparam -out DSA-PARAM.pem 1024
```

Generate DSA Keys file with Parameters file

```
openssl gendsa -out DSA-KEY.pem DSA-PARAM.pem
```

Generate DSA Parameters and Keys in one File

```
openssl dsaparam -genkey -out DSA-PARAM-KEY.pem 2048
```

See Inspecting section to view file contents.

Generating Certificate Signing Requests (CSRs) and Self-Signed Certificates

Generating CSRs:

Generate CSR with *existing* Private Key file

```
openssl req -new -key KEY.pem -out CSR.pem
```

Generate CSR and *new* Private Key file

```
openssl req -new -newkey <alg:opt> -nodes -out CSR.pem
```

Generating Self-Signed Certificates

Generate Certificate with *existing* Private Key file

```
openssl req -x509 -key KEY.pem -out CERT.pem
```

Generate Certificate and *new* Private Key file

```
openssl req -x509 -newkey <alg:opt> -nodes -out CERT.pem
```

Inspecting Certificate Signing Requests (CSRs) and Certificates

Viewing contents of Certs and CSRs

Viewing x509 Certificate as human readable Text

```
openssl x509 -in CERT.pem -noout -text
```

Viewing Certificate Signing Request (CSR) contents as Text:

```
openssl req -in CSR.pem -noout -text
```

Extracting Specific Info from Certificates

Extract specific pieces of information from x509 Certificates

```
openssl x509 -in CERT.pem -noout -dates
openssl x509 -in CERT.pem -noout -issuer -subject
```

Other items you can extract:				
	`-modulus`	`-pubkey`	`-ocsp_uri`	`-ocspid`
	`-serial`	`-startdate`	`-enddate`	

University of BRISTOL

Some Known attack against SSL/ TLS (CVEs?)

- Downgrade attack

  The attacker tries to make the system to use an older insecure version protocol, cryptographic algorithm with known vulnerabilities

- CRIME attack (Compression Ratio Info-leak Made Easy)

  The attacker uses a vulnerability that exploits the use of data compression in HTTPS connections, observing the size of compressed HTTPS responses

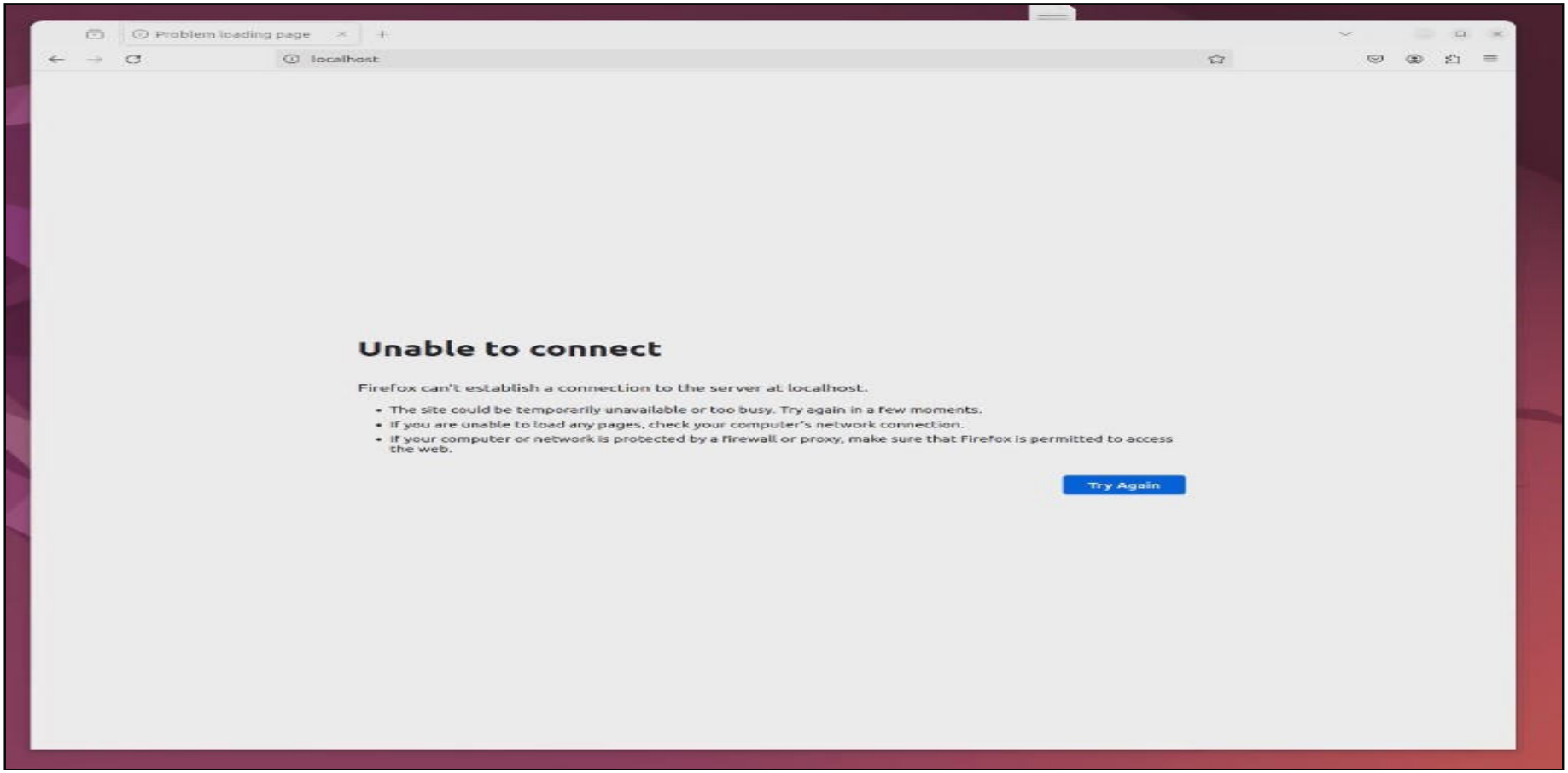
- BREACH attack (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext)

  The attacker uses a vulnerability to view encrypted traffic and force the victim to send HTTP request to a vulnerable server

https://www.openssl.org/docs/manmaster/man1/openssl.html

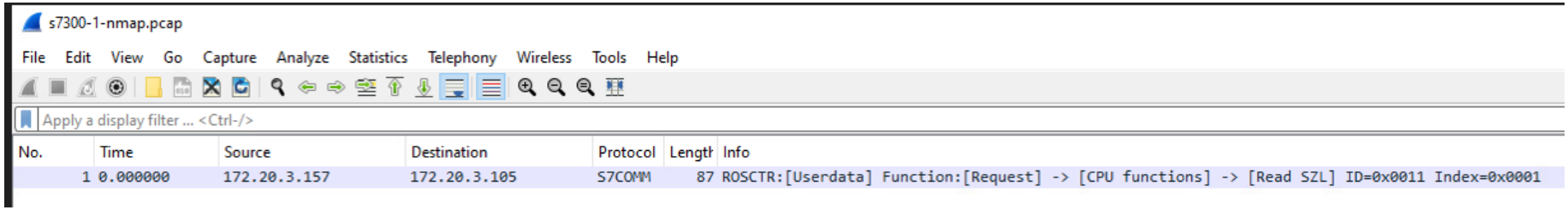
University of BRISTOL

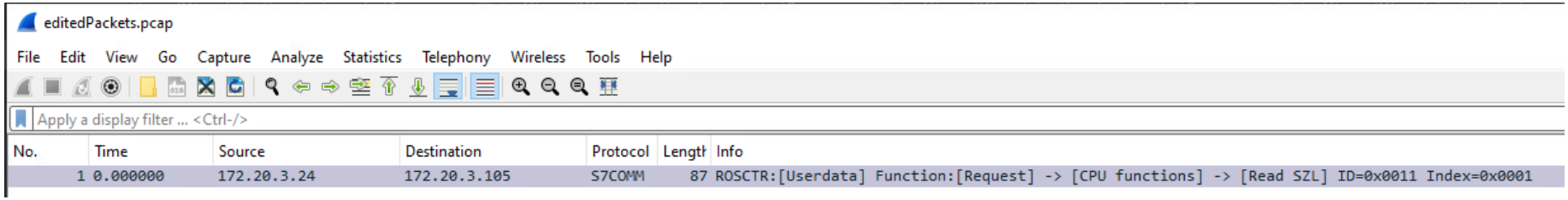
OpenSSL basics: Demo with SSL/TLS on Apache web server



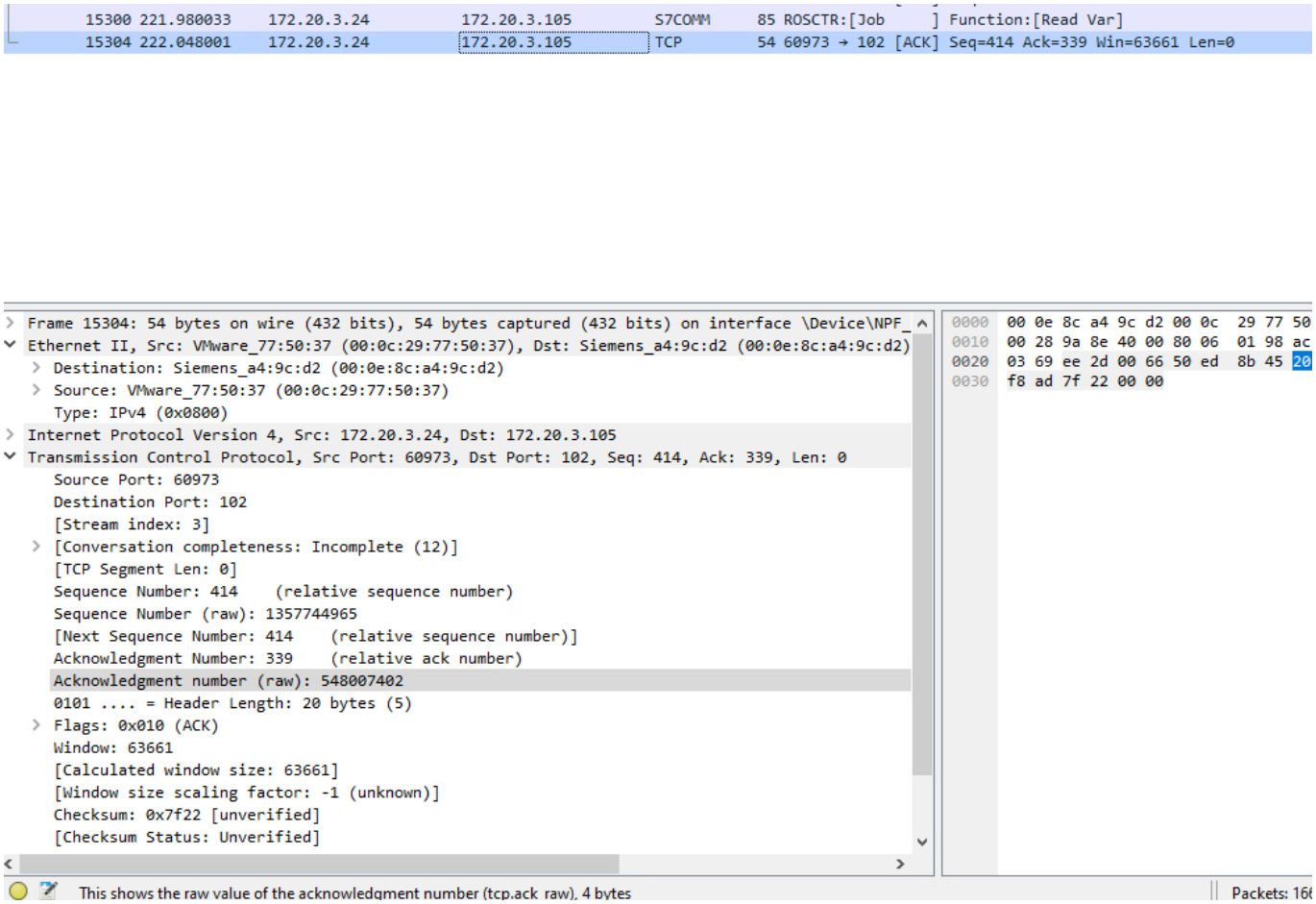
University of BRISTOL

TCP sequence prediction attack



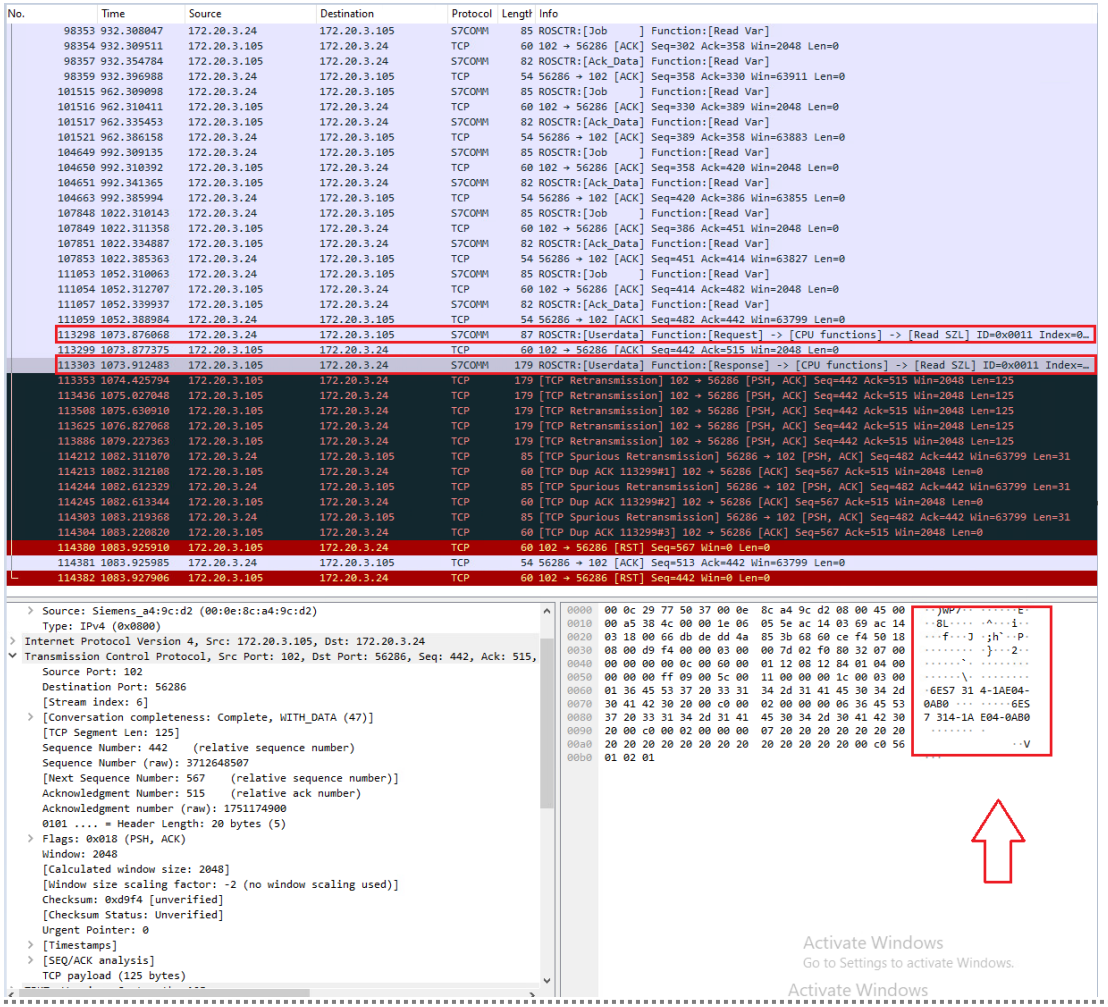


TCP sequence prediction attack



University of BRISTOL

TCP sequence prediction attack



No.	Time	Source	Destination	Protocol	Length	Info
98353	932.308047	172.20.3.24	172.20.3.105	S7COMM	85	ROSCTR:[Job] Function:[Read Var]
98354	932.309511	172.20.3.105	172.20.3.24	TCP	60	102 → 56286 [ACK] Seq=302 Ack=358 Win=2048 Len=0
98357	932.354784	172.20.3.105	172.20.3.24	S7COMM	82	ROSCTR:[Ack_Data] Function:[Read Var]
98359	932.396988	172.20.3.24	172.20.3.105	TCP	54	56286 → 102 [ACK] Seq=358 Ack=330 Win=63911 Len=0
101515	962.309098	172.20.3.24	172.20.3.105	S7COMM	85	ROSCTR:[Job] Function:[Read Var]
101516	962.310411	172.20.3.105	172.20.3.24	TCP	60	102 → 56286 [ACK] Seq=330 Ack=389 Win=2048 Len=0
101517	962.335453	172.20.3.105	172.20.3.24	S7COMM	82	ROSCTR:[Ack_Data] Function:[Read Var]
101521	962.386158	172.20.3.24	172.20.3.105	TCP	54	56286 → 102 [ACK] Seq=389 Ack=358 Win=63883 Len=0
104649	992.309135	172.20.3.24	172.20.3.105	S7COMM	85	ROSCTR:[Job] Function:[Read Var]
104650	992.310392	172.20.3.105	172.20.3.24	TCP	60	102 → 56286 [ACK] Seq=358 Ack=420 Win=2048 Len=0
104651	992.341365	172.20.3.105	172.20.3.24	S7COMM	82	ROSCTR:[Ack_Data] Function:[Read Var]
104663	992.385994	172.20.3.24	172.20.3.105	TCP	54	56286 → 102 [ACK] Seq=420 Ack=386 Win=63855 Len=0
107848	1022.310143	172.20.3.24	172.20.3.105	S7COMM	85	ROSCTR:[Job] Function:[Read Var]
107849	1022.311358	172.20.3.105	172.20.3.24	TCP	60	102 → 56286 [ACK] Seq=386 Ack=451 Win=2048 Len=0
107851	1022.334887	172.20.3.105	172.20.3.24	S7COMM	82	ROSCTR:[Ack_Data] Function:[Read Var]
107853	1022.385363	172.20.3.24	172.20.3.105	TCP	54	56286 → 102 [ACK] Seq=451 Ack=414 Win=63827 Len=0
111053	1052.310063	172.20.3.24	172.20.3.105	S7COMM	85	ROSCTR:[Job] Function:[Read Var]
111054	1052.312707	172.20.3.105	172.20.3.24	TCP	60	102 → 56286 [ACK] Seq=414 Ack=482 Win=2048 Len=0
111057	1052.339937	172.20.3.105	172.20.3.24	S7COMM	82	ROSCTR:[Ack_Data] Function:[Read Var]
111059	1052.388984	172.20.3.24	172.20.3.105	TCP	54	56286 → 102 [ACK] Seq=482 Ack=442 Win=63799 Len=0
113298	1073.876068	172.20.3.24	172.20.3.105	S7COMM	87	ROSCTR:[Userdata] Function:[Request] -> [CPU functions] -> [Read SZL] ID=0x0011 Index=0…
113299	1073.877375	172.20.3.105	172.20.3.24	TCP	60	102 → 56286 [ACK] Seq=442 Ack=515 Win=2048 Len=0
113303	1073.912483	172.20.3.105	172.20.3.24	S7COMM	179	ROSCTR:[Userdata] Function:[Response] -> [CPU functions] -> [Read SZL] ID=0x0011 Index=…
113353	1074.425794	172.20.3.105	172.20.3.24	TCP	179	[TCP Retransmission] 102 → 56286 [PSH, ACK] Seq=442 Ack=515 Win=2048 Len=125
113436	1075.027048	172.20.3.105	172.20.3.24	TCP	179	[TCP Retransmission] 102 → 56286 [PSH, ACK] Seq=442 Ack=515 Win=2048 Len=125
113508	1075.630910	172.20.3.105	172.20.3.24	TCP	179	[TCP Retransmission] 102 → 56286 [PSH, ACK] Seq=442 Ack=515 Win=2048 Len=125
113625	1076.827068	172.20.3.105	172.20.3.24	TCP	179	[TCP Retransmission] 102 → 56286 [PSH, ACK] Seq=442 Ack=515 Win=2048 Len=125
113886	1079.227363	172.20.3.105	172.20.3.24	TCP	179	[TCP Retransmission] 102 → 56286 [PSH, ACK] Seq=442 Ack=515 Win=2048 Len=125
114212	1082.311070	172.20.3.24	172.20.3.105	TCP	85	[TCP Spurious Retransmission] 56286 → 102 [PSH, ACK] Seq=482 Ack=442 Win=63799 Len=31
114213	1082.312108	172.20.3.105	172.20.3.24	TCP	60	[TCP Dup ACK 113299#1] 102 → 56286 [ACK] Seq=567 Ack=515 Win=2048 Len=0
114244	1082.612329	172.20.3.24	172.20.3.105	TCP	85	[TCP Spurious Retransmission] 56286 → 102 [PSH, ACK] Seq=482 Ack=442 Win=63799 Len=31
114245	1082.613344	172.20.3.105	172.20.3.24	TCP	60	[TCP Dup ACK 113299#2] 102 → 56286 [ACK] Seq=567 Ack=515 Win=2048 Len=0
114303	1083.219368	172.20.3.24	172.20.3.105	TCP	85	[TCP Spurious Retransmission] 56286 → 102 [PSH, ACK] Seq=482 Ack=442 Win=63799 Len=31
114304	1083.220820	172.20.3.105	172.20.3.24	TCP	60	[TCP Dup ACK 113299#3] 102 → 56286 [ACK] Seq=567 Ack=515 Win=2048 Len=0
114380	1083.925910	172.20.3.105	172.20.3.24	TCP	60	102 → 56286 [RST] Seq=567 Win=0 Len=0
114381	1083.925985	172.20.3.24	172.20.3.105	TCP	54	56286 → 102 [ACK] Seq=513 Acк=442 Win=63799 Len=0
114382	1083.927906	172.20.3.105	172.20.3.24	TCP	60	102 → 56286 [RST] Seq=442 Win=0 Len=0

```
> Source: Siemens_a4:9c:d2 (00:0e:8c:a4:9c:d2)
  Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.20.3.105, Dst: 172.20.3.24
v Transmission Control Protocol, Src Port: 102, Dst Port: 56286, Seq: 442, Ack: 515,
    Source Port: 102
    Destination Port: 56286
    [Stream index: 6]
  > [Conversation completeness: Complete, WITH_DATA (47)]
    [TCP Segment Len: 125]
    Sequence Number: 442    (relative sequence number)
    Sequence Number (raw): 3712648507
    [Next Sequence Number: 567    (relative sequence number)]
    Acknowledgment Number: 515    (relative ack number)
    Acknowledgment number (raw): 1751174900
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 2048
    [Calculated window size: 2048]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0xd9f4 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
    TCP payload (125 bytes)
```

```
0000  00 0c 29 77 50 37 00 0e  8c a4 9c d2 08 00 45 00   ··)wP7·· ······E·
0010  00 a5 38 4c 00 00 1e 06  05 5e ac 14 03 69 ac 14   ··8L···· ·^···i··
0020  03 18 00 66 db de dd 4a  85 3b 68 60 ce f4 50 18   ···f···J ·;h`··P·
0030  08 00 d9 f4 00 00 03 00  00 7d 02 f0 80 32 07 00   ········ ·}···2··
0040  00 00 00 00 0c 00 60 00  01 12 08 12 84 01 04 00   ······`· ········
0050  00 00 00 ff 09 00 5c 00  11 00 00 00 1c 00 03 00   ·····\·· ········
0060  01 36 45 53 37 20 33 31  34 2d 31 41 45 30 34 2d   ·6ES7 31 4-1AE04-
0070  30 41 42 30 20 00 c0 00  02 00 00 00 06 36 45 53   0AB0 ··· ·····6ES
0080  37 20 33 31 34 2d 31 41  45 30 34 2d 30 41 42 30   7 314-1A E04-0AB0
0090  20 00 c0 00 02 00 00 00  07 20 20 20 20 20 20 20    ·····  ·
00a0  20 20 20 20 20 20 20 20  20 20 20 20 20 00 c0 56   ·····V
00b0  01 02 01
```

University of BRISTOL

Public Key Infrastructure (PKI)

- PKI provides a means to establish trust binding public keys and identities with certificates
- With PKI we are sure that data are decrypted with corresponding private key
- If we combine this with a hash to create a signature, we can be sure that the encrypted data has not been tampered
- Certificates can be signed with the issuer private key with all info to validate the identity

Certification Authorities

- A private CA that issues certificates locally i.e., for an organization trusted by its members
- Public CAs that issue certificates publicly for members and must be trusted by the public (third party CA certificates)
- A CA must be trusted, to extend trust and the certificate includes the public key are freely distributed

University of BRISTOL

Let's Encrypt key principles

- A public CA that can automatically grand a browser-trusted certificate for an HTTPS server for free
- The prerequisite is to have a valid registered domain name and install a certificate management agent on the web server (Certbot)
- Free and open certificate authority (CA) by the Internet Security Research Group (ISRG)
- Provides security with TLS security best practices for admins to secure their websites
- Offers transparency, certificates issued will be publicly available for anyone to inspect

University of BRISTOL

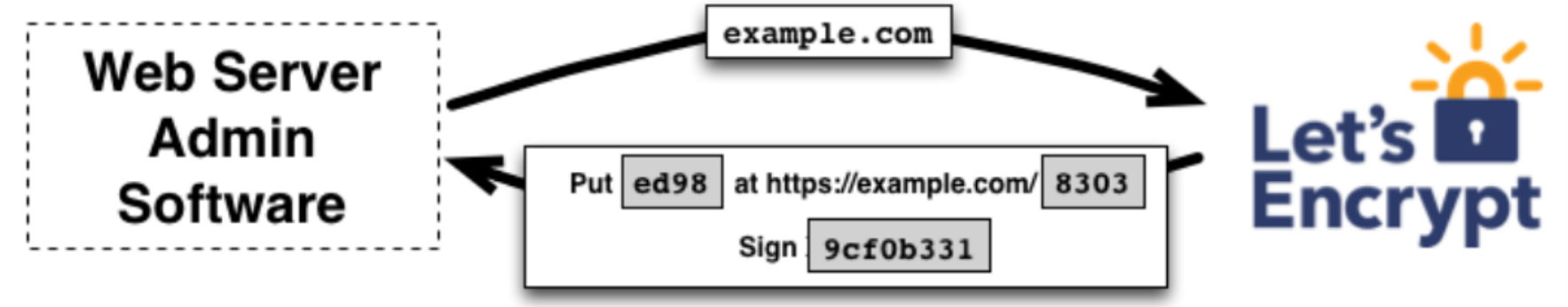
Let's Encrypt CA Basics

1. Let's Encrypt identifies the server admin with the public key. The installed agent generates a new key pair and informs Let's Encrypt that the server controls a domain
2. Let's Encrypt CA, will issue a set of challenges, for example:

   Provide the DNS record

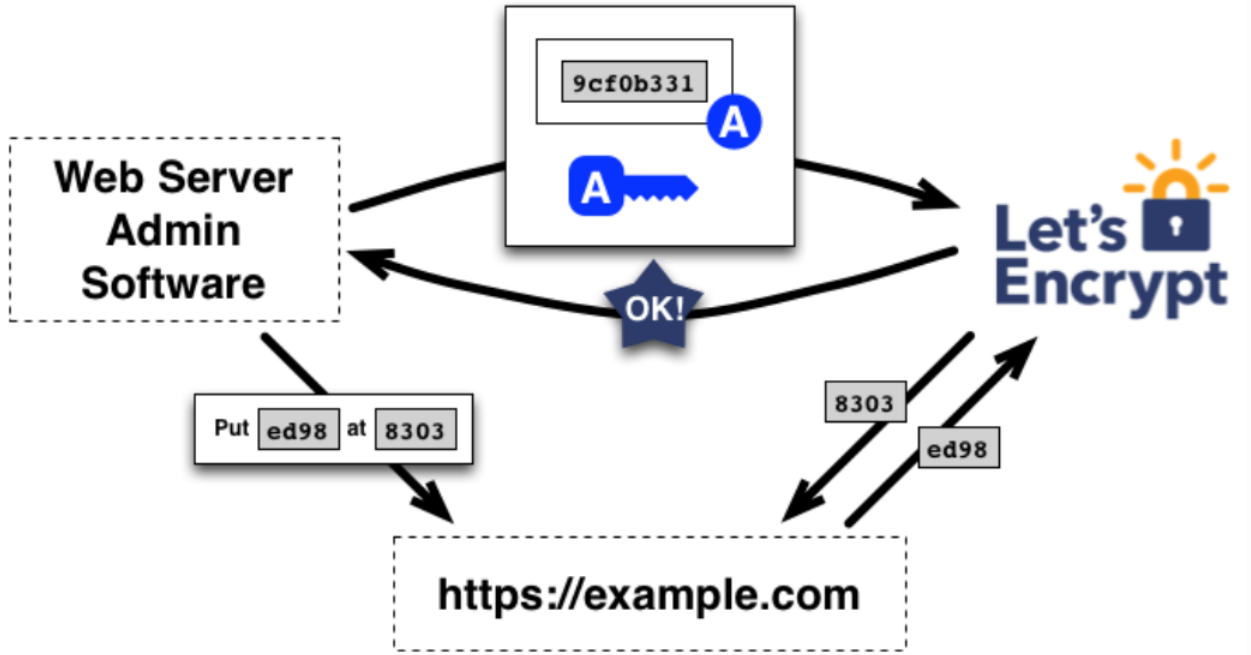
   Provide an HTTP resource

   Sign an arbitrary number (nonce) with private key



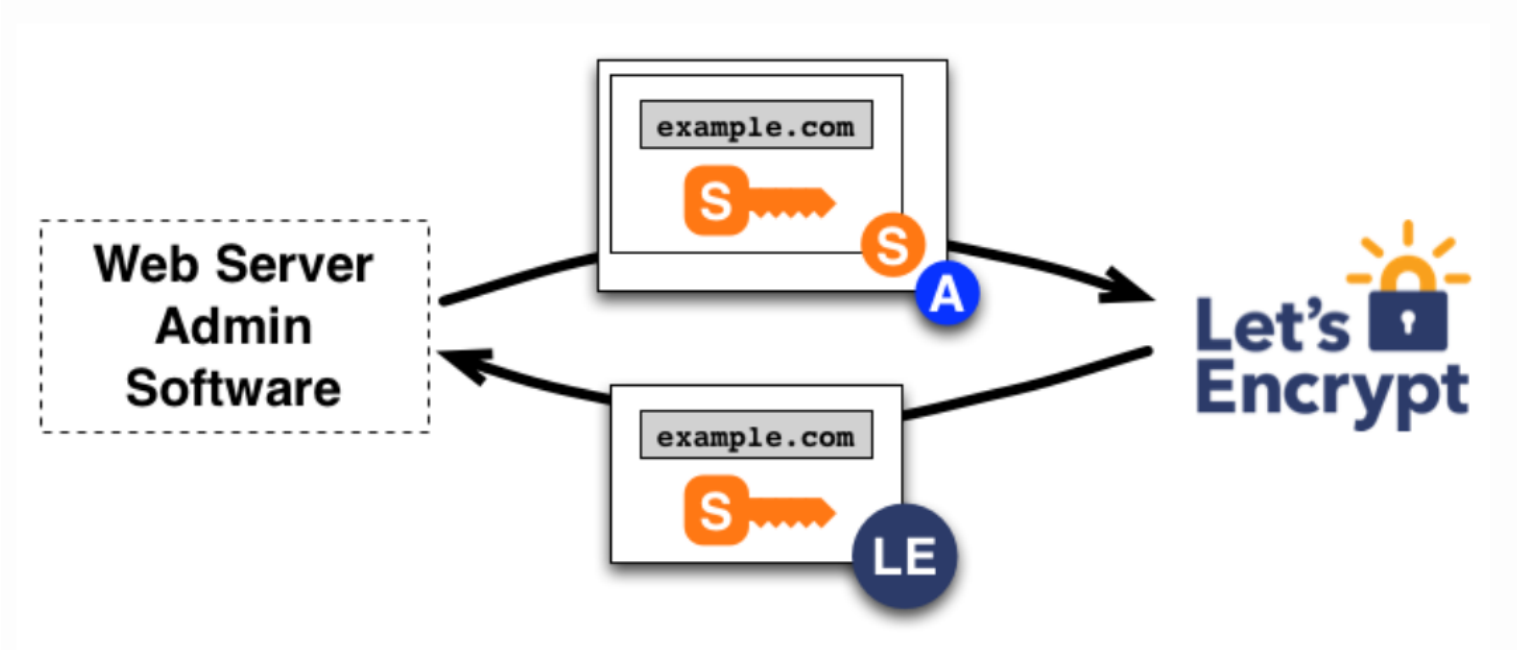
Let's Encrypt CA

4. The agent completes the tasks, and the CA validates the signature of the nonce and the task(s), grands the agent the ability to request, renew and revoke certificates



Let's Encrypt CA

5. The agent constructs a **Certificate Signing Request** with a signature (public key) and ask Let's Encrypt to issue a certificate for the domain with it's public key (whole again CSR signed with private key)

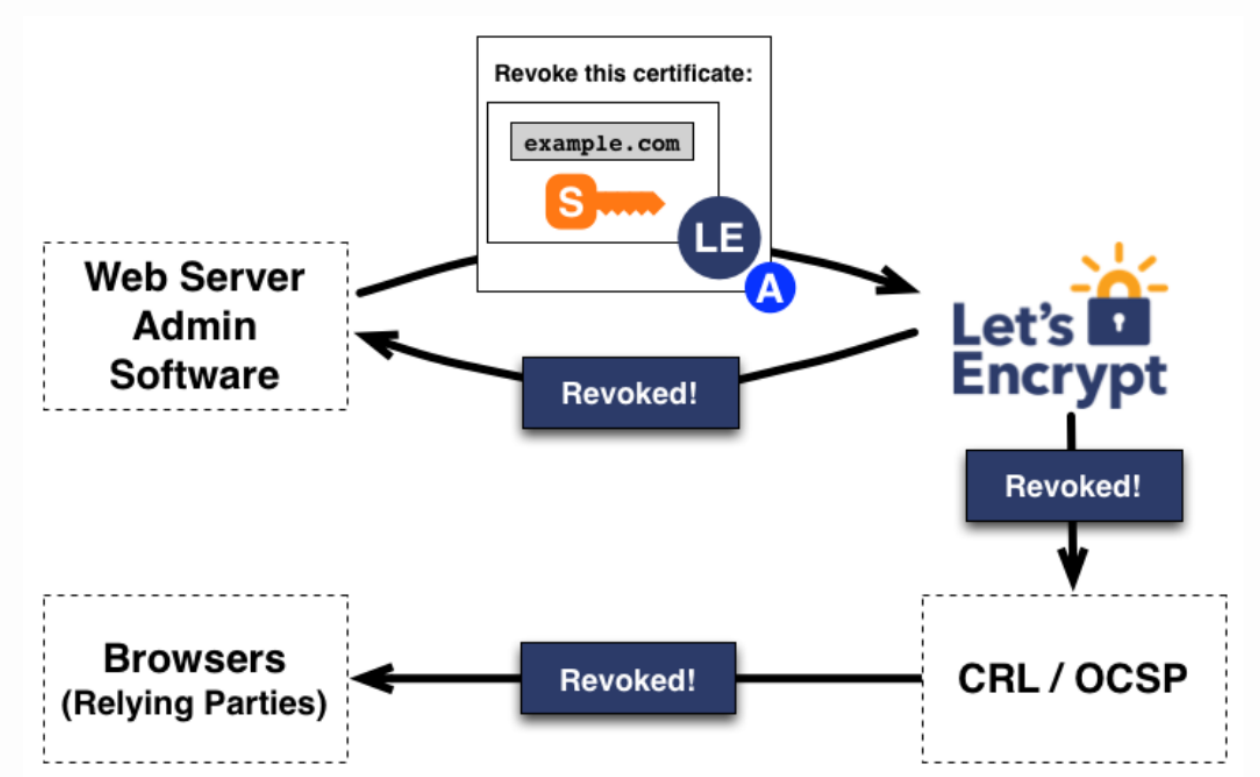


6. Let's Encrypt CA gets the request and verifies both signatures and then issues the certificate for the domain and sends it to the server

University of BRISTOL

Let's Encrypt CA

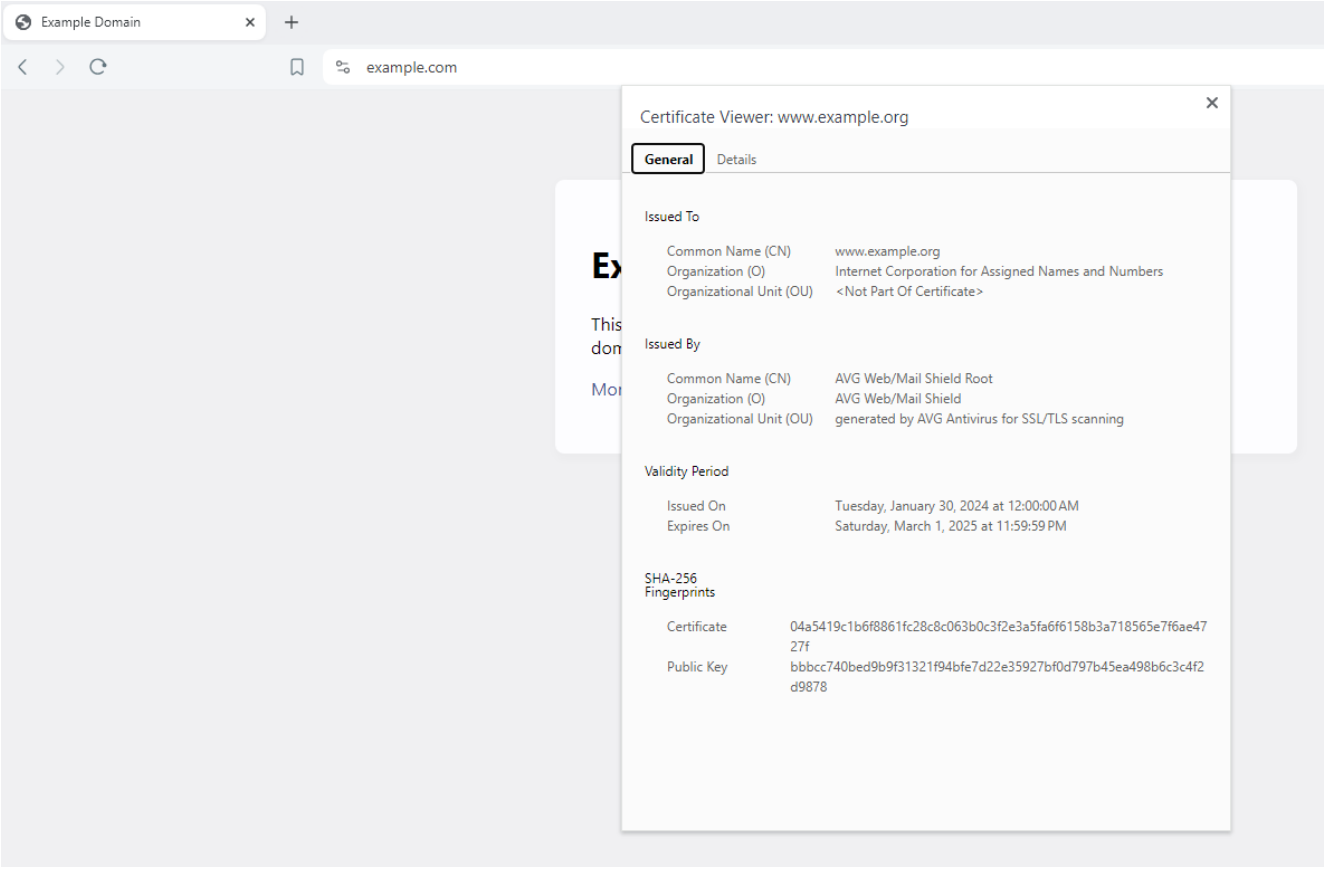
- Revocation works similarly, the agent signs a revocation request and then Let's Encrypt CA verifies the request and authorize, browsers then stop accepting the invalidate certificate



University of BRISTOL

Let's Encrypt CA

Let's see a public example which is implemented this way: https://example.com/



University of BRISTOL

Introduction to PGP - Overview

- PGP released in 1991 by Phil Zimmermann → de facto standard for secure exchange of information
- Today PGP has become an open standard known as OpenPGP
- PGP can encrypt messages online: email, plain text files etc.
- Close to military-grade symmetric and asymmetric encryption
- Relies on a private key (kept safe), integrity checking, message authentication and signed certificates
- PGP is slow therefore not considered for use in application

Suggested resources for further reading

Network Security with OpenSSL: Cryptography for Secure Communications

Authors: John Viega, Matt Messier , Pravir Chandra

Have any questions?