CRYPTOGRAPHY AT WORK: PKI & TLS

Introduction to Computer and Network Security

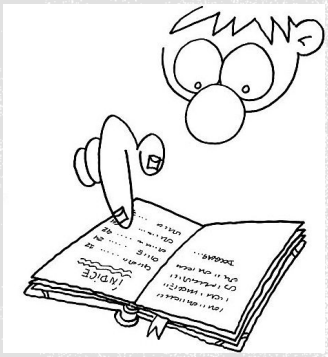
Silvio Ranise [silvio.ranise@unitn.it or *ranise@fbk.eu*]

UNIVERSITÀ DI TRENTO

FONDAZIONE BRUNO KESSLER

CONTENTS

- Public Key Cryptography reloaded
  - Confidentiality
  - Integrity
    - Signatures and hash functions
- Public Key Infrastructure
  - Digital Certificates
  - Infrastructure in brief
- SSL & TLS 1.2 and 1.3
  - Handshake
  - Some security problems
  - How TLS 1.3 addressed security issues



1

PUBLIC KEY CRYPTO (PKC): PROS & CONS

- Pros
  - **Private key is only known by the owner** (less risk)
  - Confidentiality by encrypting with Receiver's Public key
  - Non-Repudiation by encrypting with Sender's Private key
- Cons
  - **Algorithms are 2 – 3 orders of magnitude slower than those for symmetric encryption**
    - *Typically used in an initial phase of communication and then secret keys are generated for bulk encryption*
  - How are Public keys made available to the other people?
  - There is still a problem of Authentication!!!
    - **Who ensures that the owner of a key pair is really the person whose real life name is "Alice"?**

PUBLIC KEY CRYPTO (PKC): PROS & CONS

- Pros
  - **Private key is only known by the owner** (less risk)
  - Confidentiality by encrypting with Receiver's Public key
  - Non-Repudiation by encrypting with Sender's Private key
- Cons
  - **Algorithms are 2 – 3 orders of magnitude slower than those for symmetric encryption**
    - *Typically used in an initial phase of communication and then secret keys are generated for bulk encryption*
  - How are Public keys made available to the other people?
  - There is still a problem of Authentication!!!
    - **Who ensures that the owner of a key pair is really the person whose real life name is "Alice"?**

Assurance that the **sender** of information is provided with **proof of delivery** and the **recipient** is provided with **proof of the sender's identity**, so neither can later deny having processed the information.

?

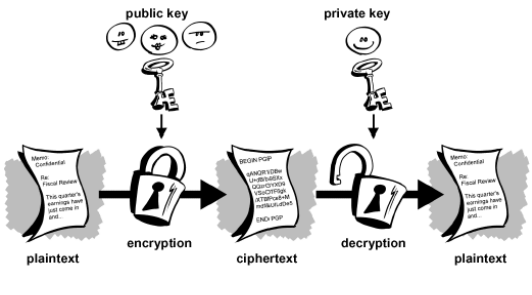
PKC: MITIGATING THE CONS

- **Digital signature** = a data item that vouches the origin and the integrity of a message

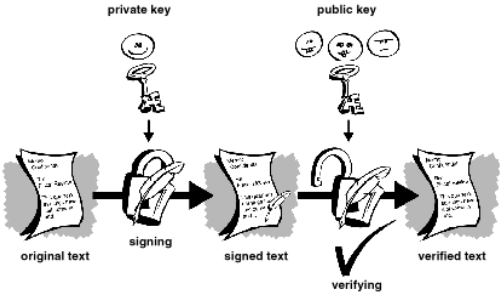
- Usage of digital signatures
  - The originator of a message uses the Private Key to sign the message and sends it together with its digital signature to a recipient
  - The recipient uses the Public Key of the sender to verify the origin of the message and that it has not been tampered with while in transit

4

RECALL PKC



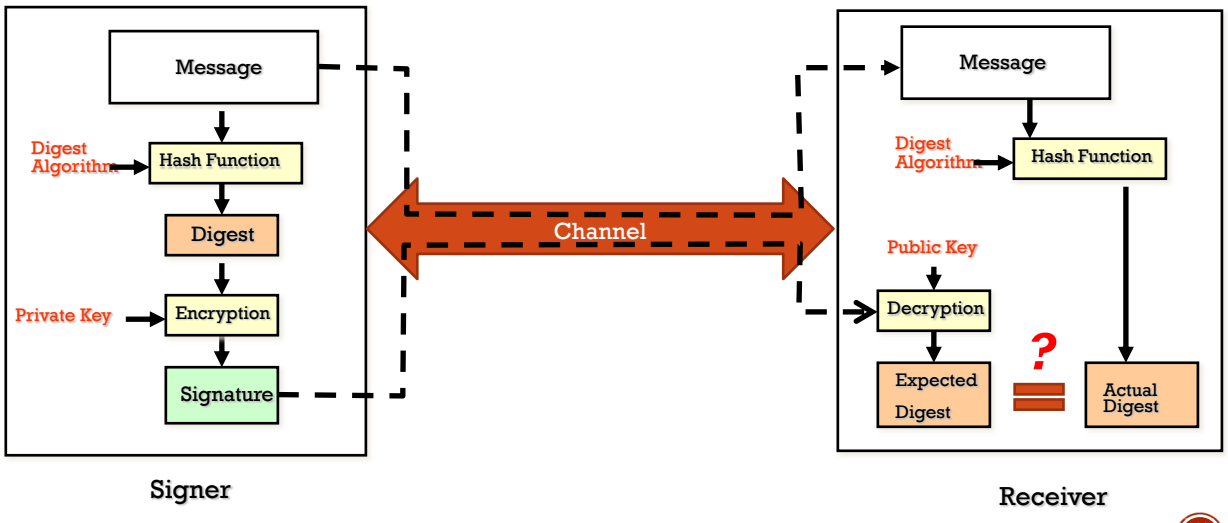
Confidentiality



Integrity

5

IN REALITY…



6

PROBLEMS WITH DIGITAL SIGNATURES

- Recall that the usage of some hash functions is deprecated because they are considered weak
- Even assuming hash functions are not a source of security issues…
- … there is still a problem linked to the "Real Identity" of the signer
  - Why should I trust who the Sender claims to be?
- In other words
  - *Who guarantees that the one using the key is entitled to do so?*

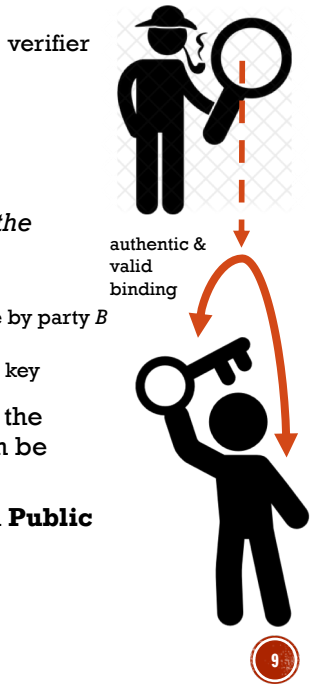
7

8

PUBLIC KEY INFRASTRUCTURE (PKI)

Or an attempt to solve the main problem underlying digital signatures…

OVERVIEW



- **Main requirement** on *public key* is to be ***authentically bound*** *to the identity of the party controlling the corresponding private key*
  - Not fulfilling this leads to impersonation attacks, .e.g.,
    - party *A* might receive a digital signature on message *M* supposedly generate by party *B* using its private key but where the signature is generated by party *C*
    - party *A* would verify the signature using *C*'s public key thinking it is using *B*'s key
- **Another requirement** is that parties relying on the correctness of the *binding between public keys and corresponding party identities* can be assured that the binding is *still valid*
- Satisfying these two requirements is typically done by setting up a **Public Key Infrastructure**

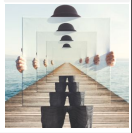
9

CERTIFICATES FOR BINDING PUBLIC KEYS AND IDENTITIES

- Initial proposal → certificates as **bulletin boards**
  - public keys and identities are listed
  - This requires trust in the provider of the bulletin board service
- Non scalable and inflexible approach adopted in mobile and Internet of Things applications is to **hardcode the public key in the software** of the party
  - Main security issues is to protect the stored public key from tampering attempts during software updates or substituting in the code repository of the software provider…
- Widely adopted solution for Internet applications and services is the use of **digital certificates**, i.e. digital objects containing data including the identity and the public key that are **signed by a Trusted Third Party** (TTP) attesting the correctness of the binding between the identity and the public key
  - Notice that this **moves the problem of checking** the correctness of the **binding between identity and public key** to the **distribution** and keeping up to date the **TTP's public key to check the authenticity of the certificate**
  - This problem can be deferred several (but **finitely many times**) and thus building a certificate chain whose trustworthiness must be lead back to a root certificate that must be self signed

See next slide

Need to avoid infinite regression



CWE™ (Common Weakness Enumeration) is a community-developed list of software and hardware weakness types. https://cwe.mitre.org/index.html

CWE-321: USE OF HARD-CODED CRYPTOGRAPHIC KEY

- The use of a hard-coded cryptographic key significantly increases the possibility that encrypted data may be recovered.
- Example

```
int VerifyAdmin(char *password) {
if (strcmp(password,"68af404b513073584c4b6f22b6c63e6b"))
{
   printf("Incorrect Password!\n");
   return(0);
}
printf("Entering Diagnostic Mode...\n");
return(1);
}
```

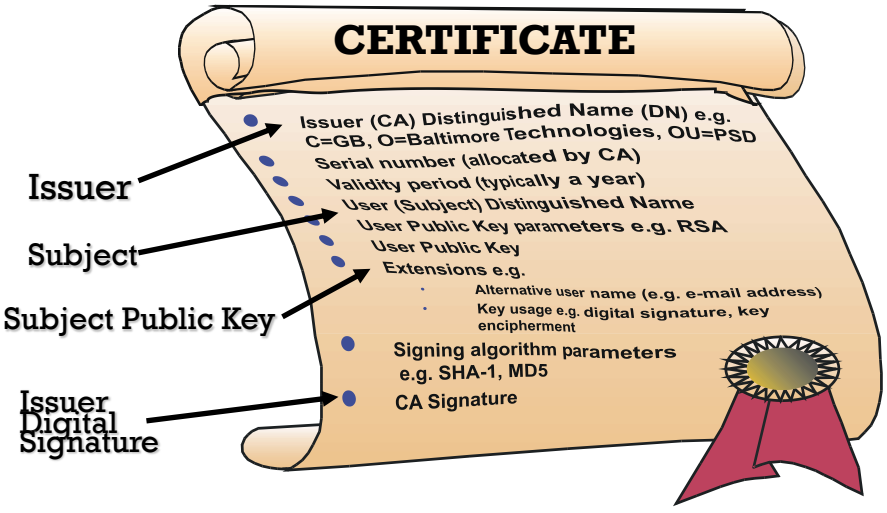
11

DIGITAL CERTIFICATES

- **Digital Certificate** = binding between an entity's Public Key and one or more Attributes concerning its Identity
- The entity can be a Person, a Hardware Component, a Service, etc.
- A Digital Certificate is issued (and signed) by someone
  - Usually the issuer is a **Trusted Third Party** (**TTP**) also called **Certificate Authority (CA)**
- Indeed, TTPs should be trusted, i.e. we should trust the digital certificates they are issuing

12

STRUCTURE OF A DIGITAL CERTIFICATE



13

X.509: A STANDARD FOR CERTIFICATES

Version		Version of X.509 to which the Certificate conforms
Serial Number		A number that uniquely identifies the Certificate
Signature Algorithm ID		The names of the specific Public Key algorithms that the CA has used to sign the Certificate (Ex.- RSA with SHA-1)
Issuer (CA) X.500 Name		The identity of the CA Server who issued the Certificate
Validity Period		The period of time for which the Certificate is valid with start date and expiration date
Subject X.500 Name		The owner's identity with X.500 Directory format (Ex.- cn=auser, ou=SP, o=Alphawest)
Subject Public Key Info	Algorithm ID	The Public Key of the owner of the Certificate and the specific Public Key algorithms associated with the Public Key
	Public Key Value	
Issuer Unique ID		Information used to identify the issuer of the Certificate
Subject Unique ID		Information used to identify the Owner of the Certificate
Extension		Additional information like Alternate name, CRL Distribution Point (CDP)
CA Digital Signature		The actual digital signature of the CA

14

CERTIFICATES: QUESTIONS & ANSWER

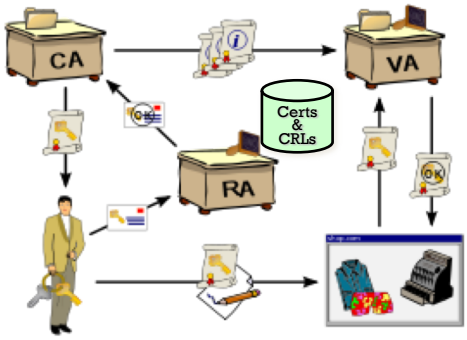
- How are Digital Certificates Issued?
- Who is issuing them?
- Why should I Trust the Certificate Issuer?
- How can I check if a Certificate is valid?
- How can I revoke a Certificate?
- Who is revoking Certificates?

One answer:
PUBLIC KEY INFRASTRUCTURE

15

PUBLIC KEY INFRASTRUCTURE (PKI)

- A PKI is an **arrangement that binds public keys with respective identities of entities** (like people and organizations)
  - An entity must be uniquely identifiable within each CA domain on the basis of information about that entity
- The binding is established through a process of registration and issuance of certificates at and by a **Certificate Authority (CA)**
  - Automated or manual process depending on the level of assurance
- The PKI role that assures valid and correct registration is called a **Registration Authority** (**RA**)
  - RA is responsible for accepting requests for digital certificates and authenticating the entity making the request
- A third-party **Validation Authority** (**VA**) can provide an entity information on behalf of the CA
- **Certificate distribution system** = repository for certificates (e.g., LDAP) + Certificate Revocation List (CRL)



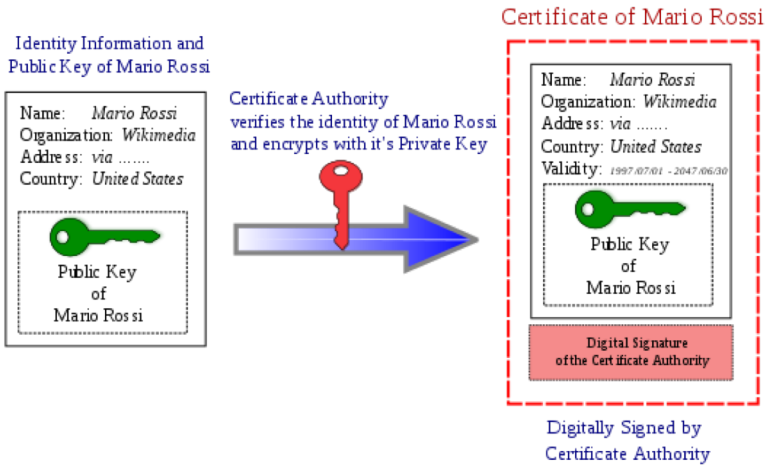
16

PKI SUMMARY

- PKI = system that provides for a TTP to vouch for user identities and allows binding of public keys to subjects
- Components:
  - The *root Certificate Authority* (CA) is the pinnacle of trust and the most significant element in the CA hierarchy. The root CA authorizes subordinate CAs
    - The *subordinate CA* is responsible for issuing certificates
  - The *Registration Authority* (RA) verifies information in a certificate request. Certificates are not issued until the RA validates the information
  - The *Cryptographic Practices Statement* (CPS) is a declaration of the security that the organization is implementing for all certificates issued by the CA holding the CPS. This statement tells potential partners or others relying on the security of the PKI system how well the security of the PKI system is being managed
  - The *Certificate Revocation List* (CRL) is a published list of certificates that have been revoked (that are no longer valid)

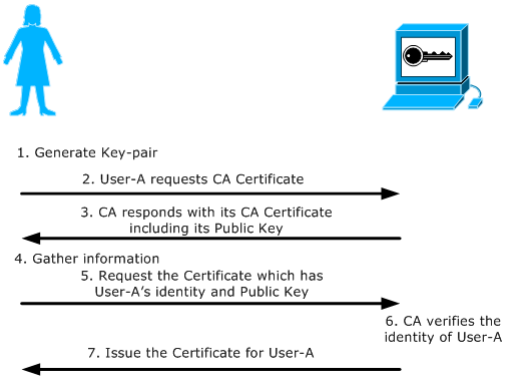
17

OBTAINING A CERTIFICATE



18

OBTAINING A CERTIFICATE (MORE DETAILS)



1. User-A generates a **Public and Private key-pair** or is assigned a key-pair by some authority in their organization
2. User-A first **requests the certificate of the CA Server**
3. The CA responds with its Certificate including its Public Key and its Digital Signature signed using its Private Key
4. User-A **gathers all information required by CA to obtain a certificate**; e.g., User-A email address, fingerprints, etc. that the CA needs to be certain that User-A claims to be who she is
5. **User-A sends a certificate request to the CA consisting of her Public Key and additional information. The certificate request is signed by User-A's Private Key.**
6. CA gets the certificate request, verifies User-A's identity and generates a certificate for User-A, binding her identity and her Public Key. The signature of CA verifies the authenticity of the Certificate
7. CA issues the certificate to User-A.

19

OBTAINING A CERTIFICATE (SOME MORE DETAILS)

- Let's focus on step 5 of the procedure to obtain a certificate, namely
  Request the certificate which has User-A's identity and public key
- This is also called the step in which a **Certificate Signing Request** (CSR) is generated
- The CSR contains
  - information identifying the requester which must be signed using the requester's private key (recall that the requester has preliminary generated a key pair)
  - the public key chosen by the requester
  - additional proofs of identity required by the certificate authority (the certificate authority may contact the applicant for further information)
  - a digital signature on the certification request information generated with the requester's private key
- The signature by the requester prevents an entity from requesting a bogus certificate of someone else's public key
  - Notice that the requester's private key is needed to produce, but it is not part of, the CSR

ACME (AUTOMATIC CERTIFICATE MANAGEMENT ENVIRONMENT)

- It is a protocol designed to automate the process of obtaining, renewing, and managing SSL/TLS certificates
- Developed by **Let's Encrypt** to enable secure HTTPS connections by providing free, automated certificates
  - https://letsencrypt.org/
  - A nonprofit Certificate Authority providing TLS certificates to 450 million websites
- Core Features
  - Automatically issues certificates for domain owners without manual intervention.
  - Validates domain ownership through methods such as DNS records or HTTP challenges
  - Automates certificate renewal before expiration, ensuring continuous HTTPS availability
  - Uses secure REST APIs for communication between clients and Certificate Authorities (CAs)
  - Allows certificate revocation when needed for security purposes

ACME (CONTD)

How ACME works

1. The client generates a key pair
2. The client proves control of the domain (e.g., challenge)
3. The client requests a certificate for the domain
4. The CA verifies the request and issues the certificate
5. The client can renew or revoke certificates as needed

Key Benefits

- **Automation**: Reduces human intervention, simplifying the deployment and management of certificates
- **Security**: Provides an easy path to enable encrypted connections, enhancing web security
- **Free Certificates**: Supported by Let's Encrypt, offering cost-effective SSL/TLS certificates

22

ACME, DOMAIN OWNERSHIP PROOF

- The client proves control of the domain (e.g., challenge)
- It ensures that **certificates are only issued to legitimate domain owners**

Methods of domain ownership proof

1. The client must host a specific file at http://<domain>/.well-known/acme-challenge/ containing a unique token provided by the Certificate Authority (CA)
   - The CA checks the file to confirm domain control
2. The client creates a specific DNS TXT record for the domain with a value provided by the CA
   - The CA verifies the DNS record to ensure the client has control over the DNS configuration
3. The client must configure a specific TLS certificate on the domain's server
   - This method proves domain control via a secure TLS channel

23

SOME REQUIREMENTS ON PKI (1)

- Standardized **naming mechanism** for parties
- **Parties** should be able to **prove** to TTP that they **own a given identity**
- **TTPs** need to **check** that parties **own a given identity**
- TTPs must be trustworthy in issuing certificates only to the "right" parties
  - The last three points above are subject to law and regulations
  - In the past, TTPs were involved in incidents resulting in certificates issued to the "wrong" parties
  - As a result, Google and other companies lunched an initiative for building a platform for checking "certificate transparency" that is able to monitor and audit certificates
    - Idea → several independent logs keep trace of the issued certificates
  - More on this can be found at https://certificate.transparency.dev/
- **Relying parties shall be able to check the time validity** window against available time sources to avoid that an **attacker can use an expired certificates whose associated private key he or she has compromised**

24

SOME REQUIREMENTS ON PKI (2)

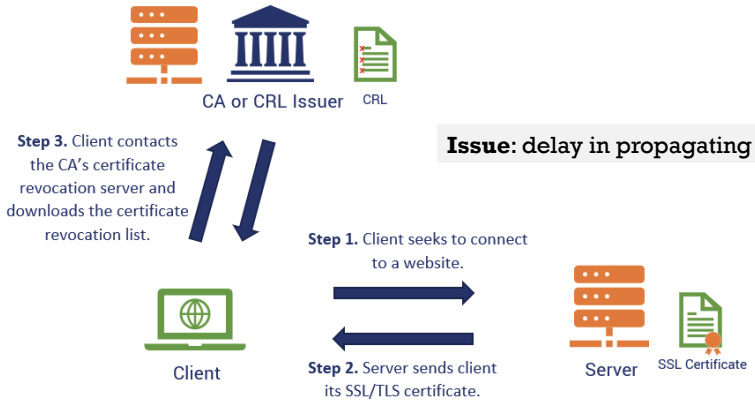
- Relying parties needs **reliable and timely source of information about the status of the certificates**
- Is the certificate still valid or has it been revoked for some reason?
- Typically this is done by either
  - distributing lists of revoked certificates (called **Certificates Revocation Lists, CRLs**) or
  - the relying parties with the TTP that the certificate that it intends to use is still valid (this is done via the **Online Certificate Status Protocol, OCSP**)
- CRLs perform the validity check locally but **there can be time windows in which the check are not updated because of the synching time required to distribute updated versions**
- OCSP requires high bandwidth availability because checks are not performed locally but the checks are always up to date

See next slide

25

CERTIFICATE REVOCATION LISTS

How to Check a Certificate's Revocation Status Using a CRL



Issue: delay in propagating updates to CRLs…

26

SOME REQUIREMENTS ON PKI (3)

- **Software** at relying party **validating certificates shall work correctly**
  - Given the complexity of the X509 standard, this is far from obvious
  - Have a look at the corresponding RFC at https://datatracker.ietf.org/doc/html/rfc5280, it is more than 150 pages long specifying complex data structures and encoding languages*!*
  - Not surprisingly, several different bugs have been found in software implementations
    - See for instance the "goto fail" bug by Apple in which a coding error (repeated line) made it possible to bypass all correctness check on certificates
      - More details at, e.g., https://www.imperialviolet.org/2014/02/22/applebug.html
- Software implementing **cryptographic primitives shall be updated according to the latest known vulnerabilities**
  - Vulnerabilities in SHA1 were found already in 2005 and cryptographers started to deprecate its usage
  - In 2017, public collisions were made available and confirmed in 2019
  - Despite the early warning more than 10 years before, only in 2017 web browsers stopped using SHA1 although certificates using SHA1 are still in usage today in payment systems*!*

27

CERTIFICATE VALIDATION (1)

- **Certificate Validation** is the process of verifying the authenticity and integrity of an SSL/TLS certificate before establishing a secure connection
- It ensures that the certificate is valid (not expired or revoked), issued by a trusted authority, and its **domain matches the requested domain**
- **Steps in Certificate Validation**
  - The client follows the certificate chain from the server's certificate to a trusted **Root Certificate Authority (CA)**
  - Each certificate in the chain is signed by the preceding CA. The client verifies each signature to ensure authenticity
  - The client checks that the certificate is within its validity period and that it hasn't been revoked (using methods like OCSP or CRLs)
  - The client verifies that the certificate's **Common Name (CN)** or **Subject Alternative Name (SAN)** matches the domain being accessed

28

CERTIFICATE VALIDATION (2)

- **Certificate Validation** is the process of verifying the authenticity and integrity of an SSL/TLS certificate before establishing a secure connection
- It ensures that the certificate is valid (not expired or revoked), issued by a trusted authority, and its **domain matches the requested domain**
- **Steps in Certificate Validation**
  - The client follows the certificate chain from the server's certificate to a trusted **Root Certificate Authority (CA)**
  - Each certificate in the chain is signed by the preceding CA. The client verifies each signature to ensure authenticity
  - The client checks that the certificate is within its validity period and that it has not been revoked (using methods like CRLs)
  - The client verifies that the certificate's **Common Name (CN)** or **Subject Alternative Name (SAN)** matches the domain being accessed

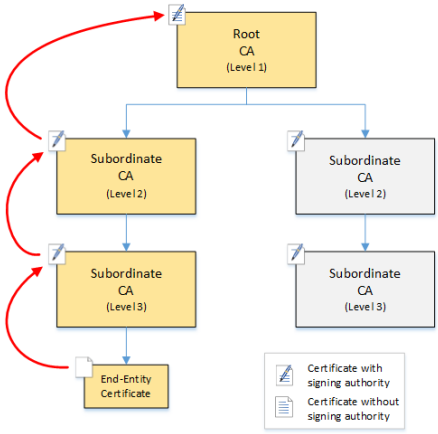
29

CHAIN OF TRUST (1)

- A **Chain of Trust** is a hierarchical structure used to validate digital certificates
- It links a **server's certificate** to a **Root Certificate Authority (CA)** via intermediate CAs, forming a trust path

Why is a Chain of Trust Needed?

- Instead of trusting each certificate individually, clients trust a small number of **Root CAs**, allowing for **scalable certificate management**
- It verifies that the certificate was issued by a **trusted CA** and not a malicious entity, ensuring **the certificate's authenticity**
- **Intermediate CAs** can issue certificates on behalf of Root CAs, enabling **distributed issuance** while maintaining security



30

CHAIN OF TRUST (2)

- To understand how many and which CAs browsers trust, have a look at the following link

https://certifi.io/

- It is characterized as a trust database of Root Certificates for humans
- Its goal is to validate the trustworthiness of certificates while verifying the identity of TLS hosts
- It contains the certificates of the Root CAs trusted by Mozilla
- Have a look at the following link for the whole database with information reported in human readable formats

https://ccadb.my.salesforce-sites.com/mozilla/IncludedCACertificateReport

31

32

SSL & TLS: INTRODUCTION

SSL & TLS

- SSL = Secure Sockets Layer.
  - Developed by Netscape in mid 1990s.
  - SSLv2 now deprecated; SSLv3 still widely supported
- TLS = Transport Layer Security.
  - IETF-standardised version of SSL.
  - TLS 1.0 = SSLv3 with minor tweaks, RFC 2246 (1999)
  - TLS 1.1 = TLS 1.0 + tweaks, RFC 4346 (2006)
  - TLS 1.2 = TLS 1.1 + more tweaks, RFC 5246 (2008)
  - **TLS 1.3** = major changes + removed insecure features, RFC 8446 (2018)

- SSL = Secure Sockets Layer.
  - Originally shipped in Netscape Navigator 1.1
  - Developed as a way to secure communications between the client and server on the web
- TLS = Transport Layer Security.
  - De facto standard for Internet security
  - Same SSL protocol design, different algorithms
  - "The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications"
  - In practice, used to protect information transmitted between browsers and Web servers

SSL & TLS

Now deployed in every web browser

- SSL = Secure Sockets Layer.
  - Developed by Netscape in mid 1990s.
  - SSLv2 now deprecated; SSLv3 still widely supported
- TLS = Transport Layer Security.
  - IETF-standardised version of SSL.
  - TLS 1.0 = SSLv3 with minor tweaks, RFC 2246 (1999)
  - TLS 1.1 = TLS 1.0 + tweaks, RFC 4346 (2006)
  - TLS 1.2 = TLS 1.1 + more tweaks, RFC 5246 (2008)
  - TLS 1.3 = major changes + removed insecure features, RFC 8446 (2018)

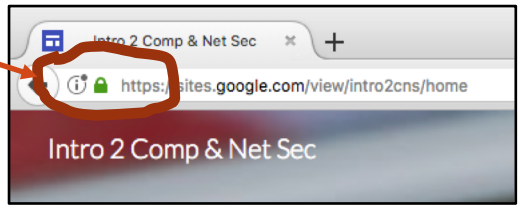
- SSL = Secure Sockets Layer.
  - Originally shipped in Netscape Navigator 1.1
  - Developed as a way to secure communications between the client and server on the web
- TLS = Transport Layer Security.
  - De facto standard for Internet security
  - Same SSL protocol design, different algorithms
  - "The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications"
  - In practice, used to protect information transmitted between browsers and Web servers

SSL & TLS IN THE BROWSER

The lock icon is the SSL indicator

Intended goal:

- Provide user with identity of page origin
- Indicate to user that page contents were not viewed or modified by a **network attacker**



HTTPS provides

- authentication of the website and associated web server with which one is communicating
  - protection against man-in-the-middle attacks
- bidirectional encryption of communications between a client and server
  - protection against eavesdropping and tampering (or forging) with the contents of the communication

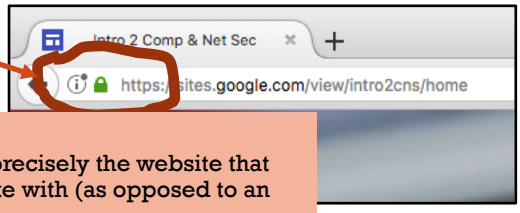
- HTTP = HyperText Transfer Protocol
- HTTPS = HTTP over TLS/SSL or HTTP Secure
- Main motivations for HTTPS:
  - authentication of the visited website
  - protection of the privacy and integrity of the exchanged data

SSL & TLS IN THE BROWSER

The lock icon is the SSL indicator

Intended goal:

- Provide user with identity of page origin
- Indicate to user tha
viewed or modified



Reasonable guarantee that
- one is communicating with precisely the website that one intended to communicate with (as opposed to an impostor)
- the contents of communications between the user and site cannot be read or forged by any third party

HTTPS provides
- authentication of the server with which o
  - protection against man-in-the-middle attacks
- bidirectional encryption of communications between a client and server
  - protection against eavesdropping and tampering (or forging) with the contents of the communication

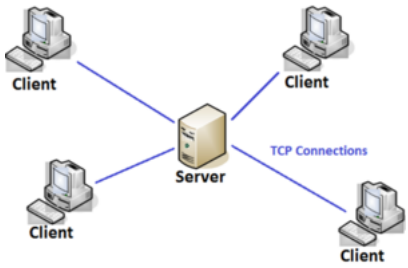
sfer Protocol
- HTTPS = HTTP over TLS/SSL or HTTP Secure
- Main motivations for HTTPS:
  - authentication of the visited website
  - protection of the privacy and integrity of the exchanged data

37

TLS IN CLIENT-SERVER ARCHITECTURES

CLIENT-SERVER ARCHITECTURE & TCP

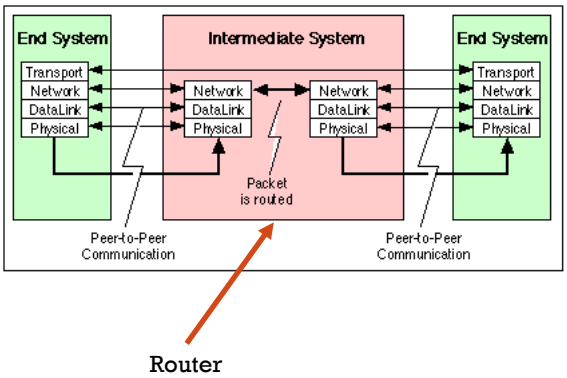
- It is an architecture of a computer network in which many **clients request and receive** service from a centralized **server**
- Clients provide an interface to allow a computer user to request services of the server and to display the results the server returns
- Servers wait for requests to arrive from clients and then respond to them
- Servers typically provide a standardized transparent interface to clients so that these need not be aware of the specifics of the system that is providing the service
- The **Transmission Control Protocol** (**TCP**) establishes a two-way connection between a server and a single client; it **provides reliable byte stream transmission of data** with error checking and correction, and message acknowledgement



Reliable in the sense of safety not security, i.e. it is able to handle accidental errors not those induced by an attacker with a strategy

CLIENT-SERVER ARCHITECTURE & TRANSPORT LAYER PROTOCOL

- A client or server application interacts directly with a **transport layer protocol** to establish communication and to send or receive information
- The transport protocol then uses lower layer protocols to send or receive individual messages
  - A computer needs a complete stack of protocols to run either a client or a server
- The transport layer protocol provides transparent transfer of data between end systems (also called end points) using the services of the network layer (e.g., TCP/IP)



TLS AT A GLANCE

Public Key cryptography

Used by client and server to
1.Negotiate cipher suite
2.Authenticate
3.Establish keys used in the Record Protocol

Handshake protocol

Record protocol

Symmetric cryptography

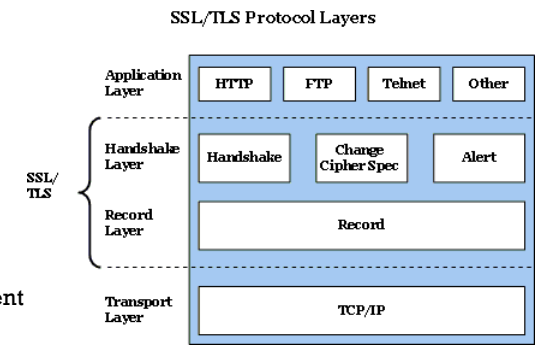
Provides confidentiality and integrity/authenticity of application layer data using keys from Handshake Protocol

TLS OVERVIEW

- TLS consists of two main protocols

- **Handshake** protocol
  - Use public-key cryptography to establish a shared secret key between the client and the server

- **Record** protocol
  - Use the secret key established in the handshake protocol to protect communication between the client and the server
  - Focus on the handshake protocol



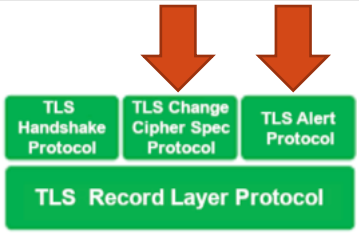
TLS HANDSHAKE PROTOCOL (IN A NUTSHELL)

- Two parties: client and server
- Negotiate version of the protocol and the set of cryptographic algorithms to be used
  - Interoperability between different implementations of the protocol
- Authenticate client and server (optional)
  - Use digital certificates to learn each other's public keys and verify each other's identity
- Use public keys to establish a shared secret

TLS RECORD PROTOCOL (IN A NUTSHELL)

- Provides Confidentiality and (Message) Integrity
- How?
- On the **send** side
  - **fragmenting** the data into blocks
  - applying **authentication** and **encryption** primitives to each block
  - handing the block to TCP for transmission over the network
- On the **receive** side
  - blocks are decrypted
  - verified for message integrity
  - reassembled
  - delivered to the higher-level protocol

TLS: ADDITIONAL PROTOCOLS



- **TLS Change Cipher Protocol**
  - It is used to change the encryption being used by the client and server
  - It is normally used as part of the handshake process to switch to symmetric key encryption.

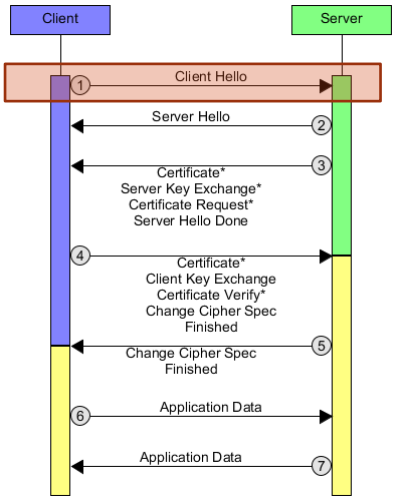
- **TLS Alert Protocol**
  - The primary use is to report the cause of failure
  - Examples
    - invalid messages received
    - messages that cannot be decrypted
    - connections closed

45

TLS 1.2: SOME DETAILS OF THE HANDSHAKE

https://tools.ietf.org/html/rfc5246

TLS HANDSHAKE PROTOCOL: STEP BY STEP



Step 1

- The Client Hello message contains:
  - the **version of the protocol** that the client wants to use
  - list of supported **cipher suite**
  - …

A **cipher suite** is a set of algorithms that help secure a network connection that uses TLS/SSL

TLS CIPHER SUITES (1)

Cipher suite = a structure describing the algorithms that a machine supports in order for two machines to decide which algorithms to use to secure their connection

- A cipher suite is a set of algorithms that help secure a network connection that uses TLS. Usually, it includes
  - a key exchange algorithm
  - a bulk encryption algorithm
  - a message authentication code (MAC) algorithm
- Cipher suites can include **signatures** and an **authentication** algorithm to help authenticate the server or the client
- Available hundreds of cipher suites that contain different combinations of algorithms above
  - Some cipher suites offer better security than others
- A list of cipher suites is available at
  - https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4

TLS CIPHER SUITES (2)

A **message authentication code** (**MAC**) is a short piece of information used to authenticate a message, i.e. to confirm that the message came from the stated sender (authenticity) and has not been tampered in transit (integrity)

- Naming convention
- Example: ***TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256***
  - **TLS** defines the protocol that this cipher suite is for
  - **ECDHE_RSA** indicates the key exchange algorithm being used
    - The key exchange algorithm is used to determine if and how the client and server will authenticate during the handshake
  - **AES_128_GCM** indicates the block cipher being used to encrypt the message stream, together with the block cipher mode of operation
  - **SHA256** indicates the message authentication algorithm which is used to authenticate a message

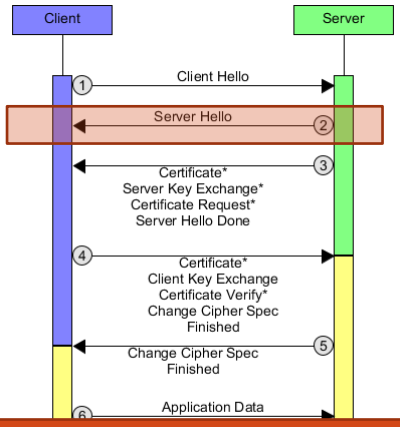
Elliptic Curve Diffie Hellman Ephemeral for key negotiation (note: ephemeral guarantees that no session key is used twice)

RSA for certificate (signature) validation

Example: **HMAC** (Hash-based Message Authentication Code)

- Sender hashes the content of messages combining the resulting hash with a key (which is negotiated during the TLS handshake) into a short message authentication code that is sent alongside the message
- Receiver repeats the process to ensure that the message has not been altered in transit

TLS HANDSHAKE PROTOCOL: STEP BY STEP

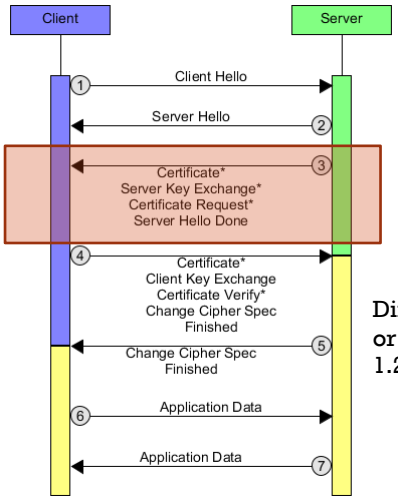


Step 2

- The server hello message contains
  - **chosen protocol** (supported by both parties);
  - **chosen cipher suite** (supported by both parties);
  - **session ID**: a freshly-generated value that will identify the new session
  - …

A **session** is a series of interactions between two communication end points that occur during the span of a single connection. Typically, one end point requests a connection with another specified end point and if that end point replies agreeing to the connection, the end points take turns exchanging commands and data ("talking to each other"). The session begins when the connection is established at both ends and terminates when the connection is ended.

TLS HANDSHAKE PROTOCOL: STEP BY STEP



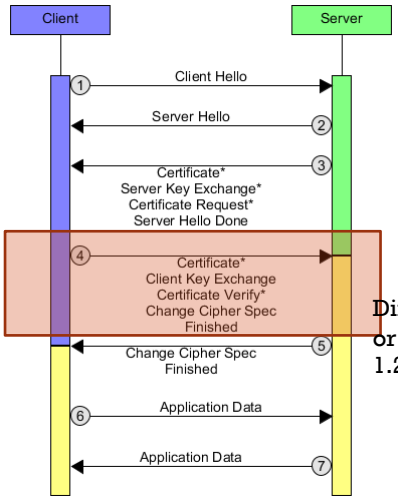
Diffie-Hellman or RSA in TLS 1.2

Step 3

- Certificate: If the client requested an authenticated connection, the server must send a X.509 certificate
- **Server Key Exchange = message sets the premaster secret that will be later used to generate the Master Secret**
- Certificate Request: a non-anonymous server can send this message if it requires the client to be authenticated

50

TLS HANDSHAKE PROTOCOL: STEP BY STEP



Diffie-Hellman or RSA in TLS 1.2

Step 4

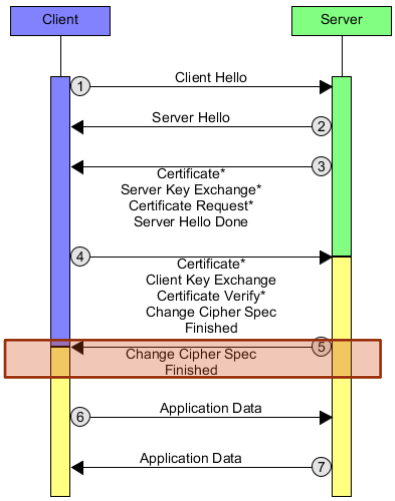
- Certificate: if the client has received a CertificateRequest message, it must send a X.509 certificate
- Client Key Exchange = message sets the **pre-master secret** that will be later used to generate the **master secret**
- Certificate Verify: message that provides explicit verification of the client certificate

51

ON PRE-MASTER & MASTER SECRETS

- The **pre-master key/secret** is the value obtained from the key exchange
  - Example: when using Diffie-Hellman, the pre-master secret is g^(ab) mod p
- Depending on the cryptographic parameters, the size of pre-master keys can vary
- To simplify the generation of session keys, it is important to identify a fixed-length value from which to derive what we call **master secret/key** whose length is fixed and equal to 48 bytes
- The master secret is derived from the pre-master key by using a **Pseudo Random function** (PRF), i.e. an efficient and deterministic function which returns a **pseudorandom** output indistinguishable from **random** sequences
  - Main difference w.r.t. pseudorandom number generators is that they can accept any input data
- **From the master secret both client and server derive multiple session keys by using again the PRF**: half of the session keys is used for message integrity and the other half for message confidentiality
  - **Principle**: *do not use the same key for both encryption and signing/message authentication*

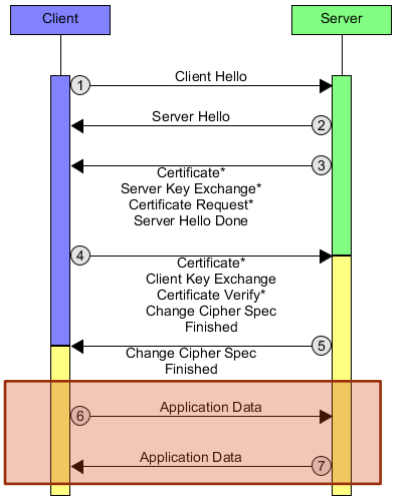
TLS HANDSHAKE PROTOCOL: STEP BY STEP



Step 5

- Change Cipher Spec = message (consisting of a single byte of value 1) sent by both parties. Once received, the participant transitions to the agreed cipher suite.
- Finished = generated by hashing the entire handshake and sent by both parties. It is used to signal the completion of the algorithm.

TLS HANDSHAKE PROTOCOL: STEP BY STEP

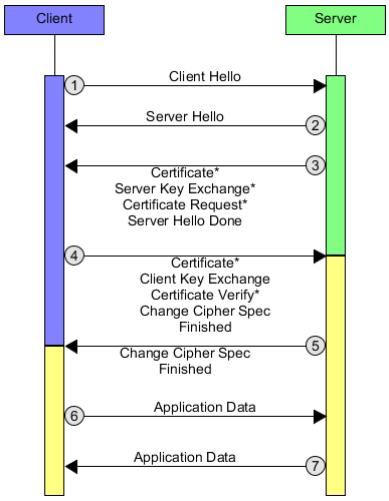


Steps 6-7

- From the reception of the Change Cipher Spec message onward, every message sent (both by server and client) will be encrypted using the shared key

Record protocol

CERTIFICATE VERIFICATION



- Step 3
  - TLS client verifies the server's digital certificate
- Step 4
  - TLS server verifies the client's certificate
- **How SSL and TLS provide identification, authentication, confidentiality, and integrity?**

HOW TLS PROVIDES AUTHENTICATION

- For **server authentication**, the client uses the server's public key to encrypt the data that is used to compute the secret key. The server can generate the secret key only if it can decrypt that data with the correct private key.
- For **client authentication**, the server uses the public key in the client certificate to decrypt the data the client sends during step 4 of the handshake. The exchange of finished messages that are encrypted with the secret key (steps 6 and 7 in the overview) confirms that authentication is complete.
- If any of the authentication steps fail, the handshake fails and the session terminates.
- The exchange of digital certificates during the SSL or TLS handshake is part of the authentication process.

WHAT HAPPENS DURING CERTIFICATE VERIFICATION

- In steps 3 and 4, the TLS client verifies the server's certificate, and the TLS server verifies the client's certificate. There are four aspects to this verification:

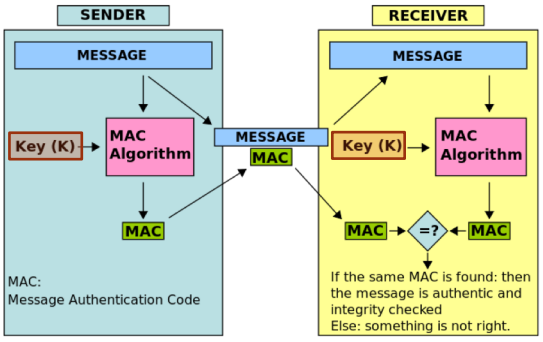
1. The digital signature is checked
2. The certificate chain is checked in case of intermediate CA certificates
3. The expiry and activation dates and the validity period are checked
4. The revocation status of the certificate is checked

HOW TLS PROVIDES CONFIDENTIALITY

- SSL and TLS use a combination of symmetric and asymmetric encryption to ensure message privacy
- During the TLS handshake, the TLS client and server agree an encryption algorithm and a shared secret key to be used for one session only.
- **All messages transmitted between the SSL/TLS client and server are encrypted using that algorithm and key, ensuring that the message remains private even if it is intercepted.**
- SSL/TLS supports a wide range of cryptographic algorithms
  - This is true up to version 1.2 as in version 1.3 the number of cipher suites has been dramatically reduced to eliminate most weak primitives
- Because SSL and TLS use asymmetric encryption when transporting the shared secret key, there is no key distribution problem.

HOW TLS PROVIDES INTEGRITY

- SSL and TLS provide data integrity by using a **Message Authentication Code** (**MAC**)
- **Use of TLS does ensure data integrity, provided that the CipherSpec in the channel definition uses a suitable hash or MAC algorithm**
- Use of MD5 and SHA1 is strongly discouraged as this is now very old and no longer secure for most practical purposes



https://en.wikipedia.org/wiki/Message_authentication_code

- MAC functions are similar to hash functions but have different security requirements
- MACs are similar to digital signatures but differ in that MACs are both generated and verified by using the **same key**

S. Ranise - Security & Trust (FBK)

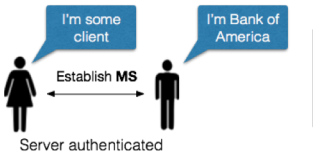
SHORT DIGRESSION ON MAC

- A message authentication code (MAC) is a tag used to confirm that
  - the message came from the stated sender (authenticity) and
  - has not been tampered with (integrity)
- The MAC allows verifiers to detect any changes to the message content
- A **MAC algorithm** is a **family of cryptographic functions** – parameterized by a symmetric key – that can be used to provide data origin authentication, as well as data integrity, by producing a MAC on arbitrary messages

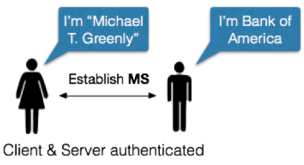
60

IN A NUTSHELL

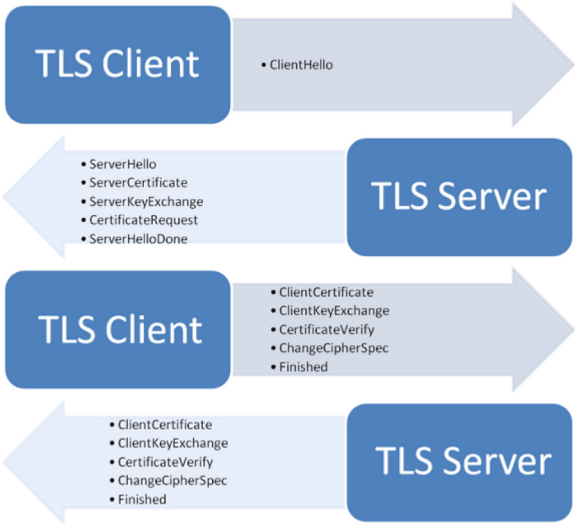
- *TLS is a protocol for establishing secure (Transport layer) communications between two parties, usually denoted as a Client and a Server*
- Focus on TLS handshake (see right)



Server authenticated



Client & Server authenticated



61

A RECENT EXTENSION TO ENHANCE PRIVACY (1)

- **Encrypted Client Hello (ECH)** is a new proposed standard available at

  https://datatracker.ietf.org/doc/draft-ietf-tls-esni/

- It masks the **Server Name Indication** (SNI) that is used to negotiate a TLS handshake
- Whenever a user visits a website that has ECH enabled, no one except for the user and the website owner will be able to determine which website was visited

- Before SNI, when making a TLS connection, the client had no way to specify which site it was trying to connect to
- If one server hosts multiple sites on a single listener, the server has no way to know which certificate to use in the TLS protocol
- For more details on SNI, see https://en.wikipedia.org/wiki/Server_Name_Indication

A RECENT EXTENSION TO ENHANCE PRIVACY (2)

- **Encrypted Client Hello (ECH)** is a new proposed standard available at

  https://datatracker.ietf.org/doc/draft-ietf-tls-esni/

- It masks the **Server Name Indication** (SNI) that is used to negotiate a TLS handshake
- Whenever a user visits a website that has ECH enabled, no one except for the user and the website owner will be able to determine which website was visited

- Before SNI, when making a TLS connection, the client had no way to specify which site it was trying to connect to
- If one server hosts multiple sites on a single listener, the server has no way to know which certificate to use in the TLS protocol
- For more details on SNI, see https://en.wikipedia.org/wiki/Server_Name_Indication

AN OLDER EXTENSION TO ENHANCE PRIVACY

- Whenever a user visits a website, the operating system needs to know which IP address to connect to. This is done through a DNS request
- DNS by default is unencrypted, meaning anyone can see which website you're asking about
- To help users shield these requests from intermediaries, servers have adopted DNS over HTTPS (DoH)

https://en.wikipedia.org/wiki/DNS_over_HTTPS

- A step further is Oblivious DNS over HTTPS which prevents even the webserver from seeing which websites a user is trying to connect to

https://datatracker.ietf.org/doc/rfc9230/

TLS VULNERABILITIES

Disclaimer: we can give only intuitions underlying the attacks as the technical details may be quite involved…

BE CAREFUL!

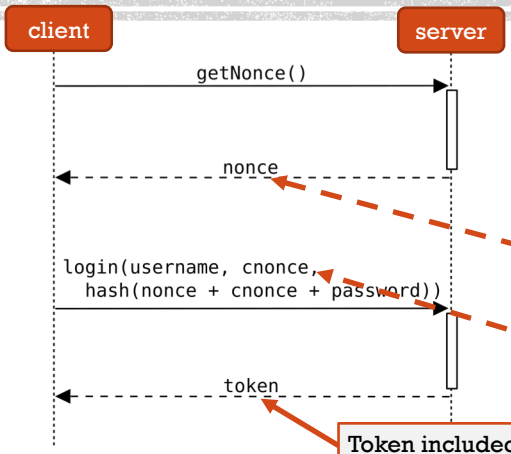
- Despite being the basic building block of Internet security…
- … TLS suffers from some security issues for a few reasons
  - Backward compatibility
    - The protocol still supports weak cipher suites and broken hash functions
  - Logical flaws
    - A set of logical loopholes can be used to "trick" both client and server
  - Implementation issues
    - Libraries (including the widely used **Openssl**) contains bugs and some of them can be exploited to mount attacks

Openssl is considered an important building block of the Internet. You can read more at https://www.openssl.org/

KEY ESTABLISHMENT: REPLAY ATTACKS & NONCES

- ClientHello offers many different cipher suites, choice of which to use is made by server
- ClientHello and ServerHello also contain 32-byte nonces
  - 28-byte random values + 4-byte time encoding
- Nonces are signed by the server in Diffie-Hellman based cipher suites and involved in key derivation
- Nonce are important for security to prevent session replay attacks forcing reuse of session keys
- Nonces avoid attacks aiming to induce client and server to regenerate the same key of a previous session by replying the messages of the previous session containing the parameters to derive the session key using the Diffie-Hellman key exchange

DIGRESSION ON NONCES (1)



- A **nonce** is an arbitrary number that can be used just once in a cryptographic communication
- Typically, it is a (pseudo-)random number issued by one of the parties in a protocol to ensure that old communications cannot be reused in replay attack

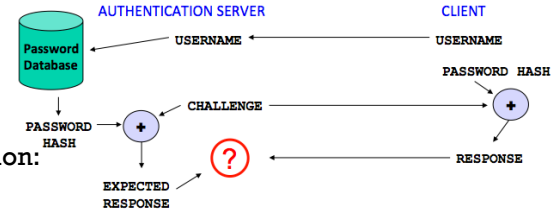
Example: typical use of nonces to ensure **freshness** in authentication protocols is shown on the left

- Without the server's nonce and client's cnonce an attacker could reuse a previously generated login(…) message to authenticate as the client

Token included by the client in all the requests to the server so that the server knows which client is asking a given resource

DIGRESSION ON NONCES (2)

- Challenge–response authentication is a family of protocols in which one party presents a question ("challenge") and another party must provide a valid answer ("response") to be authenticated
- Typically
  - Party presenting challenge: server
  - Party providing response: client
- The simplest example is password authentication:
  - challenge is asking for the password and
  - valid response is the correct password
- Obvious extension
  - Use One Time Password as 2nd factor authentication



TLS is a fundamental pre-requisite to implement this kind of solution over the Internet!

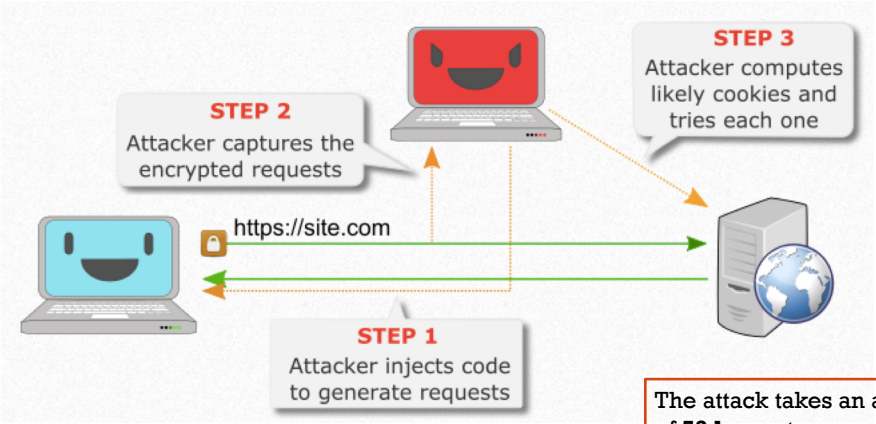
RC4NOMORE (1)

NOMORE is acronym for Numerous Occurrence MOnitoring & Recovery Exploit

- Remember that RC4 is a stream cipher whose software implementation is straightforward (it consists of shuffling the elements stored in an array of bytes)
- It was, until 2017, one of the most widely used cipher in deployments of TLS (record protocol)
- In 2013, there were first evidence of the fact that RC4 was weak because an attacker can guess certain values of the key stream…
- In 2015, the exploitation of such biases was shown possible in practice and in 2017 the RC4NOMORE attack showed the practicality of discovering repeated text in messages such as **tokens used for authentication**
  - These tokens are called **cookies** and are widely used in web applications
- The idea is to **use the weaknesses in the RC4 cipher to speed up the guessing of a token so that the attacker can authenticate as the victim**

RC4NOMORE (2)

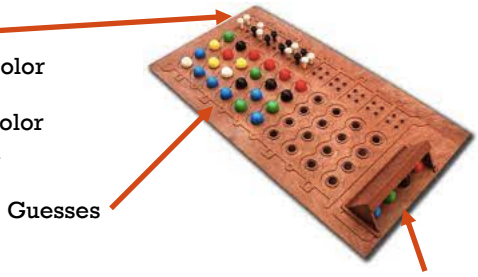
The picture is taken from the following web site https://www.rc4nomore.com/ which is dedicated to the RC4NOMORE attack where you can get many more details…



The attack takes an average of **52 hours** to succeed and a maximum of **75 hours**

RC4NOMORE (3)

Feedback
- black peg = correct color and position
- white peg = correct color but incorrect position

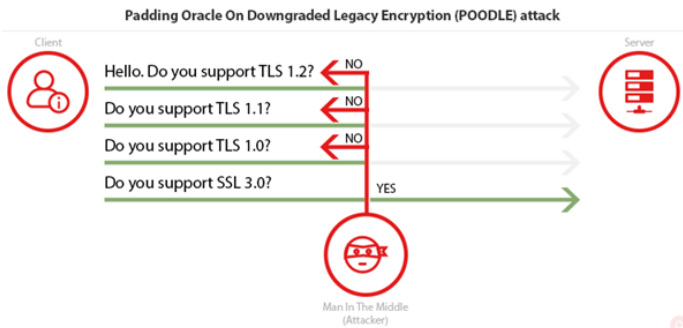


- Intuition: attack is similar to game Mastermind
- The key pegs represent the token
- Code injected at step 1 is generating guesses
- Comparison of ciphertext of the token with encrypted guess corresponds to feedbacks

- As a consequence of this attack, all major browsers deprecated the use of RC4 in TLS… nowadays, RC4 is almost never used in TLS connections

POODLE (2014)

Padding Oracle On Downgraded Legacy Encryption



- It exploits two features
  - some servers/clients still support SSL 3.0 for backward compatibility with legacy systems
  - a vulnerability in SSL 3.0 related to block padding
- Attack idea
  - The client initiates the handshake and sends a list of supported SSL/TLS versions
  - An attacker intercepts the traffic, performing a man-in-the-middle (MITM) attack, and impersonates the server until the client agrees to **downgrade** the connection to SSL 3.0
  - At this point, the attacker can exploit a weakness in one of the block ciphers (namely, the Cipher Block Chaining mode with padding) supported by SSL 3.0 to guess cookies and other sensitive information
- Full details at https://www.openssl.org/~bodo/ssl-poodle.pdf

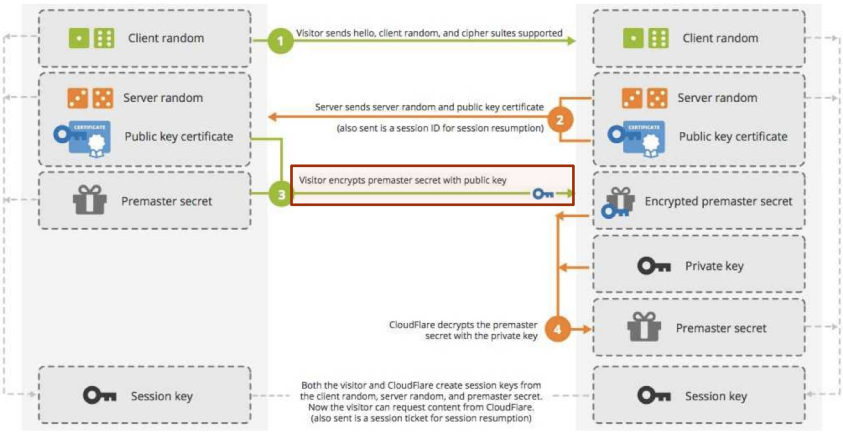
Similar to the BEAST (*Browser Exploit Against SSL/TLS*) attack of 2011

BLEICHENBACHER ATTACK & ROBOT (1)

- In 1998, Bleichenbacher discovered a devastating man-in-the-middle attack on SSL, the predecessor of TLS
- Specifically, Bleichenbacher presented an attack on RSA encryption **exploiting the padding** which in turn allows a man-in-the-middle adversary against SSL to recover the pre-master secret and thence the application keys (**when RSA is used to exchange the session key and not Diffie-Hellman as we have seen before**)
- The weakness is that the server answers to ciphertext with wrong padding saying that this is the case while it answers positively when the padding is correct…
- … an attacker can guess messages and as soon as the server says that one is correctly padded, he/she knows to be on the right track to guess the plaintext
- TLS incorporates an ad hoc fix to mitigate this attack, namely decryption failures are hidden from the attacker or, equivalently, the server answers correctly and incorrectly padded messages in the same way

DIGRESSION ON TLS USING RSA FOR SESSION KEY EXCHANGE

client / visitor



server / cloudflare

Picture adapted from
https://developers.cloudflare.com/internet/protocols/tls

BLEICHENBACHER ATTACK & ROBOT (2)

- ROBOT = Return Of Bleichenbacher's Oracle Threat
- In 2017, we have witnessed the return after almost 20 years of the same type of vulnerabilities based on different server behaviors when RSA cryptography fails
- We are not going to dive into the details that can be found at the following web site

  https://www.robotattack.org/
- We just mention that the authors created a tool to systematically scan TLS servers and check their behavior in presence of RSA failures
- The most important takeaways from this work is that the **best mitigation is to deprecate the use of RSA for key exchange in TLS!**

Quite recently (2023), a new incarnation of Bleichenbacher's attack called Marvin has been discovered… details are available at https://people.redhat.com/~hkario/marvin/

HEARTBLEED (2014)

- Found in the *heartbeat* extension of the popular OpenSSL library used to keep a connection alive as long as both parties are still there
- The client sends a heartbeat message to the server with a payload that contains *data* and the *size* of the data (and *padding*)
- The server must respond with the same heartbeat request, containing the data and the size of data that the client sent
- If **the client sent false data length**, the **server would respond with the data received by the client and random data from its memory to meet the length** requirements **specified by the sender**
- **Random data from server memory can be the private key of the server, credentials, sensitive documents, credit card numbers, emails, …**

TLS VULNERABILITIES IN A NUTSHELL

- Over time, **weaknesses of some ciphers** may become known and they should be deprecated as soon as cryptographers say that they are no more safe to use since "attacks can only get better"
  - **Example**: biases in the key stream generation of RC4 has lead to RC4NOMORE attack
- Sometimes the problem is not a weakness in the ciphers but **how the cipher is used** that make the resulting ciphertext amenable to so called **covert channel attack**, i.e. the normal behavior of a system may leak information about the data being processed in unintended/unexpected ways
  - **Example**: a TLS server reacting in different ways to wrong or correct padding used in combination with RSA can reveal some information about the content of messages (in particular those parts that are repeated in every message such as authentication tokens) and is the basis of **Bleichenbacher's attack and its variants such as ROBOT**
- Sometimes the problem is the **combination of a weak cipher and a glitch in the logic underlying the exchange of messages**, i.e. in parts of the **protocol**
  - **Example**: the possibility of inducing the selection, during **renegotiation**, of a cipher whose weaknesses an attacker knows how to exploit in order to extract plaintexts (e.g., authentication tokens) as it is the case in the **POODLE attack** exploiting weaknesses in block ciphers used in particular modes of operation

MITIGATIONS

- Once identified, the above vulnerabilities (as well as many others) were patched or mitigations were suggested
- Several problems can be avoided by proper configuration of the TLS server
- However, configuring such servers is not easy, especially by IT professionals without proper training and expertise in security issues
- To assist this task, automated tools are available that, given the URL of the TLS server, return a list of potential vulnerabilities
- One such tool is TLSAssistant which besides identifying potential vulnerabilities can also provide suggestions on how to fix them or automatically produce fixes to the configuration
  - The tool is available at https://st.fbk.eu/tools/TLSAssistant.html

80

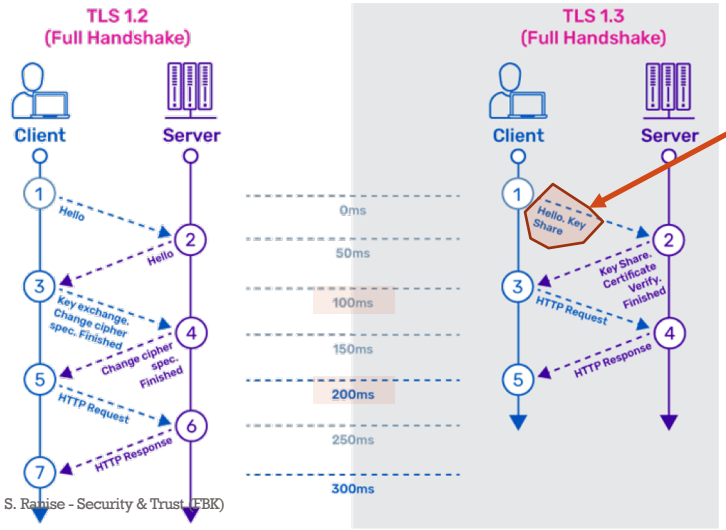
TLS 1.3

https://tools.ietf.org/html/rfc8446

GOALS OF TLS 1.3

- Clean up: Remove unsafe or unused features
- Security: Improve security with respect to modern techniques
- Privacy: Encrypt more of the protocol
- Performance: 1-RTT and 0-RTT handshakes
- Continuity: Backwards compatibility

FASTER HANDSHAKE



- With the Hello message, the client sends the cryptographic parameters for key exchange since **the only way to derive session keys is by using (a variant of) Diffie-Hellman**.
- This reduces the number of messages to exchange during the handshake since **there is no more need to agree on how to derive the session key**
- This makes the protocol more secure against covert channel attacks such as those based on Bleichenbacher's weakness
- More secure above means that Bleichenbacher's attacks are still possible even though TLS 1.3 does not support the weak key exchange mechanism based on RSA with padding
  - For details see https://www.nds.ruhr-uni-bochum.de/media/nds/veroeffentlichungen/2015/08/21/Tls13QuicAttacks.pdf

0/1 ROUND-TRIP TIME (0/1-RTT)

See next slides for the notions of Perfect Forward Secrecy (PFS) and ephemeral keys

- TLS relies on key exchanges to establish a secure session
- In earlier versions, keys could be exchanged during the handshake using one of two mechanisms: a static RSA key exchange or a Diffie-Hellman key exchange
- **In TLS 1.3, RSA has been removed**, along with all static (non-PFS) key exchanges, while retaining **ephemeral Diffie-Hellman keys**
- In addition to eliminating the security risk posed by a static key, which can compromise security if accessed illicitly, relying exclusively on the Diffie-Hellman family allows the **client** to **send the required cryptographic parameters for key generation already included in its "hello"**
- By eliminating an entire round-trip on the handshake, this saves time and improves overall site performance
  - This modality is called "one round-trip time" (1-RTT)
- In addition, when accessing a site that has been visited previously, a client can send data on the first message to the server by leveraging **pre-shared keys** (**PSK**) from the prior session
  - This modality is called "**zero round-trip time**" (0-RTT)

CIPHER SUITES GREATLY REDUCED

- Shrunk the size of the cipher suites used for encryption
  - **5 in TLS 1.3** against 319 in TLS 1.2
  1. TLS_AES_128_GCM_SHA256
  2. TLS_AES_256_GCM_SHA384
  3. TLS_CHACHA20_POLY1305_SHA256
  4. TLS_AES_128_CCM_SHA256
  5. TLS_AES_128_CCM_8_SHA256
- In TLS 1.2 and earlier versions, the use of ciphers with cryptographic weaknesses had posed potential security vulnerabilities
- TLS 1.3 includes support only for algorithms that currently have no known vulnerabilities, including …
- …any that do not support **Forward Secrecy**

Meaning of acronyms

AES = Advanced Encryption Standard
Block cipher: block size 128 bits, keys with various size

ChaCha20 = stream cipher
Poly1305 = Message Authentication Code

GCM = Galois Counter Mode
CCM = Counter with Cipher Block Chaining
Modes of operation for block cipher AES
They provide both data authenticity (integrity) and confidentiality

SHA = Secure Hash Algorithm
Family of cryptographic hash functions published by NIST and US FIPS
Generates digests of varying sizes

WHAT IS FORWARD SECRECY?

- **Forward secrecy** or **perfect forward secrecy** is a feature of specific key agreement protocols that gives assurances that **session keys** (used in the past) **will not be compromised even if long-term secrets used in the session key exchange are compromised**
- For HTTPS the long-term secret is typically the private signing key of the server
- **Forward secrecy protects past sessions against future compromises of keys or passwords**
- By generating a unique session key for every session a user initiates, the compromise of a single session key will not affect any data other than that exchanged in the specific session protected by that particular key
  - This by itself is **not sufficient for forward secrecy which additionally requires that a long-term secret compromise does not affect the security of past session keys**

ON EPHEMERAL DIFFIE-HELLMAN (1)

- Related to forward secrecy
  - The problem of the RSA method to pass keys is that a breach of the long term (private) keys would breach the keys previously used for secure communications
  - if server sends its public key to the client
    - client sends back a session key for the connection encrypted with the public key of the server
    - server will then decrypt this and determine the session
  - A leakage of the public key of the server would cause all the sessions which used this specific public key to be compromised
- With some key exchange methods (e.g., the "standard" Diffie-Hellman method), **the same key will be generated if the same parameters are used on either side**
- This can cause problems as an intruder could guess the key
- With ephemeral methods a different key is used for each connection, and, again, the leakage of any long-term would not cause all the associated session keys to be breached
- The problem with the Diffie-Hellman method is that the keys are not ephemeral, so we should avoid it in generating keys
- We need to use a suitable extension of the Diffie-Hellmann method called Ephemeral Diffie-Hellman…

ON EPHEMERAL DIFFIE-HELLMAN (2)

- **Ephemeral Diffie-Hellman** (DHE in the context of TLS) differs from the **standard (static) Diffie-Hellman** (DH) in the way that static Diffie-Hellman key exchanges always use the same Diffie-Hellman private keys
  - So, each time the same parties do a DH key exchange, they end up with the same shared secret
- When a key exchange uses Ephemeral Diffie-Hellman a temporary DH key is generated for every connection and thus the same key is never used twice
  - Due to increasing concerns about pervasive surveillance, key exchanges that provide Forward Secrecy are recommended
    - See, e.g., https://tools.ietf.org/html/rfc7525#section-6.3
- **Ephemeral Diffie-Hellman provides no authentication** because the key is different every time
  - Neither party can be sure that the key is from the intended party
- **Within TLS, DHE should be combined with an additional authentication mechanism…** this is done by using so-called **Authenticated Encryption with associated data** (AEAD) ciphers that we will not see in the course, it will be sufficient to know that AEAD ciphers are symmetric ciphers that can guarantee confidentiality together with authenticity and integrity

DISCLAIMER (1)

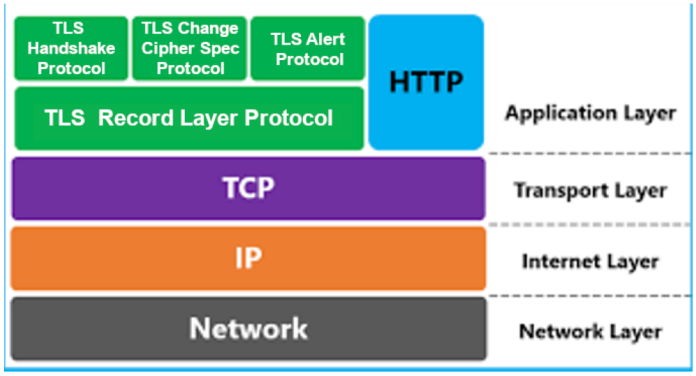
- We have **only scratched the surface** of the TLS specifications and security issues
- There are many more details about the parameters used during key exchange that we have omitted for the sake of brevity that play a key role in providing security
- There is a long history of attacks to TLS that we have omitted because they require the consideration of several different details that are impossible to consider in an introductory course such as this
- If you want to dive more in the nitty-gritty details of TLS and related vulnerabilities/attacks, you can consider the following website

  https://tlseminar.github.io/readings/

  which contains a useful collection of pointers to relevant materials on these topics

DISCLAIMER (2)

- A lot of security problems below TLS/SSL omitted such as
  - IP Security
  - Firewalls
  - IPsec/VPNs
  - Worms
  - DDOS
  - …



RECAP QUESTIONS

- What are the advantages and disadvantages of Symmetric and Asymmetric Cryptography?
- How can PKC be used to sign messages?
  - What is the role of hash functions in this process?
- What is a digital certificate and what are its main components?
- What is the Public Key Infrastructure (PKI) and which are its main entities?

RECAP QUESTIONS-CONT'D

- What is SSL/TLS? Where is it used?
- How does the handshake protocol of TLS work?
  - What is the role of Diffie-Hellman technique?
  - Can RSA be used for negotiating a key during the handshake? What is the main drawback of this method with respect to Diffie-Hellman?
- What is a MAC (Message Authentication Code)?
  - How is it different and similar to digital signatures?
- What are the potential security problems of TLS?
  - How does TLS 1.3 attempt to address some of the security issues in previous versions of TLS?