Cryptography
Week #11:
PKI

Rogério Reis, rogerio.reis@fc.up.pt
2023/2024
DCC FCUP

December, 15th 2023

Why PKI?

Why PKI

All PK cryptography primitives assume public-keys are authentic.

If not true, protocols are vulnerable to man-in-the-middle attacks.

In the real-world this problem can be solved in an ad-hoc way:

- manually confirm public-key belongs to intended party
- systems (e.g., GPG/PGP) supporting ad-hoc PK authentication

When legal/regulatory coverage is required $\Rightarrow$ PKI:

- Technical standards: which algorithms/encoding formats to use
- Regulations: how technical standards should be used
- More Regulations: responsibilities and rights of involved parties
- Laws: formal guarantees and penalties wrt regulations

Public-key certificates

Public-key certificates

Goal:

- Alice sends Bob a public key pk over an insecure channel
- Bob must be able to check Alice holds associated secret key

Trivial solution:

- Bob has authenticated channel to Trusted-Third-Party (TTP)
- Alice has previously proved to TTP that she owns pk (how?)
- Bob asks TTP (on-line) if pk belongs to Alice

Public-key certificates

Goal:

- Alice sends Bob a public key pk over an insecure channel
- Bob must be able to check Alice holds associated secret key

Trivial solution:

- Bob has authenticated channel to Trusted-Third-Party (TTP)
- Alice has previously proved to TTP that she owns pk (how?)
- Bob asks TTP (on-line) if pk belongs to Alice

Problems in practice:

1. How does Bob build authenticated channel to TTP?
2. What happens if TTP is off-line?
3. How do Bob and Alice get to work with the same TTP?
4. What does "Trust" in TTP mean?

Public-key certificates (2)

Public-key certificates use signatures to solve points 1 and 2:

- TTP is called a *Certification Authority* (CA)
- Alice proves to CA that she owns pk
  - By signing a certificate request (PKCS#11)
  - Because CA itself provides secret key to Alice

Public-key certificates (2)

Public-key certificates use signatures to solve points 1 and 2:

- TTP is called a *Certification Authority* (CA)
- Alice proves to CA that she owns pk
  - By signing a certificate request (PKCS#11)
  - Because CA itself provides secret key to Alice
- CA provides/checks data Alice wants on certificate:
  - Alice identity + public key
  - CA-specific information: serial number, issuer identity
  - Validity (start and end dates)
- CA signs data as a byte-encoded ASN.1 data structure.

PK Certificate := Alice's data and PK + CA signature

Trust in certificate $\leq$ Trust in CA

Public-key certificates (3)

What is ASN.1 (see here for some examples)?

- Abstract Syntax Notation 1: platform/language independent
- Legacy specification language from networking standards
- Standards use ASN.1 to specify data structures (packets)
- DER (Distinguished Encoding Rules) specify byte encoding

How do certificates solve points 1 and 2:

- Digital signature guarantees certificate is authentic to Bob
- CA can be off-line: Bob can get certificate via Alice!

Public-key certificates (3)

What is ASN.1 (see here for some examples)?

- Abstract Syntax Notation 1: platform/language independent
- Legacy specification language from networking standards
- Standards use ASN.1 to specify data structures (packets)
- DER (Distinguished Encoding Rules) specify byte encoding

How do certificates solve points 1 and 2:

- Digital signature guarantees certificate is authentic to Bob
- CA can be off-line: Bob can get certificate via Alice!

So can certificates be sent over insecure channels?

Public-key certificates (3)

What is ASN.1 (see here for some examples)?

- Abstract Syntax Notation 1: platform/language independent
- Legacy specification language from networking standards
- Standards use ASN.1 to specify data structures (packets)
- DER (Distinguished Encoding Rules) specify byte encoding

How do certificates solve points 1 and 2:

- Digital signature guarantees certificate is authentic to Bob
- CA can be off-line: Bob can get certificate via Alice!

So can certificates be sent over insecure channels?

Other natural questions:

- How does Bob know CA and verifies the CA signature?
- What are Alice/Bob actually trusting the CA to do?

Verifying a Public-Key Certificate

Suppose Alice sends Bob a public-key certificate with:

- Alice's identity and public key
- A validity period (start and end dates)
- Some additional meta-information
- All signed by certification authority CA

Verifying a Public-Key Certificate

Suppose Alice sends Bob a public-key certificate with:

- Alice's identity and public key
- A validity period (start and end dates)
- Some additional meta-information
- All signed by certification authority CA

This is what Bob should do:

1. Check Alice's identity is correct (e.g., DNS name for server)
2. Check current time is within validity period
3. Check meta-information makes sense for application
4. Check CA is *trustworthy* to certify this public-key
5. Obtain CA's public key and verify signature in certificate

The first three are self-explanatory. PKI solves 4 and 5.

Sanity check: did you understand how this works?

Who sends and who receives/validates a PK certificate in:

- Asymmetric encryption:

Sanity check: did you understand how this works?

Who sends and who receives/validates a PK certificate in:

- Asymmetric encryption:
  - Public key belongs to receiver
  - Sender must get certificate beforehand
- Digital signatures

Sanity check: did you understand how this works?

Who sends and who receives/validates a PK certificate in:

- Asymmetric encryption:
  - Public key belongs to receiver
  - Sender must get certificate beforehand
- Digital signatures
  - Public key belongs to signer
  - OK to sign and send certificate along (M, σ)
- Key agreement

Sanity check: did you understand how this works?

Who sends and who receives/validates a PK certificate in:

- Asymmetric encryption:
  - Public key belongs to receiver
  - Sender must get certificate beforehand
- Digital signatures
  - Public key belongs to signer
  - OK to sign and send certificate along (M, σ)
- Key agreement
  - If mutually authenticated, then both must send certificates
  - What happens usually in TLS?

Sanity check: did you understand how this works?

Who sends and who receives/validates a PK certificate in:

- Asymmetric encryption:
  - Public key belongs to receiver
  - Sender must get certificate beforehand
- Digital signatures
  - Public key belongs to signer
  - OK to sign and send certificate along (M, σ)
- Key agreement
  - If mutually authenticated, then both must send certificates
  - What happens usually in TLS?

Example: in S/MIME (signed email) clients usually

- Allow signing a message as soon as personal certificate installed
- Needs signed message from Alice before allowing encryption
- Does this make sense?

Technical details about public-key certificates

Standardized in X.509 and transposed to internet by IETF

Important data structures have unique **object identifiers**

Technical details about public-key certificates

Standardized in X.509 and transposed to internet by IETF

Important data structures have unique **object identifiers**

Current version is 3, which includes basic fields:

- subject, issuer, validity, public key info, serial

Extensions (attachments), some of which may be marked *critical*

- all extensions carry an object identifier
- if marked critical but not recognized $\Rightarrow$ reject!

Technical details about public-key certificates

Standardized in X.509 and transposed to internet by IETF

Important data structures have unique **object identifiers**

Current version is 3, which includes basic fields:

- subject, issuer, validity, public key info, serial

Extensions (attachments), some of which may be marked *critical*

- all extensions carry an object identifier
- if marked critical but not recognized $\Rightarrow$ reject!

Important extensions:

- Subject/authority key identifier: fingerprint of public key
- Basic constraints: flag that signals special CA certificate
- Key usage: CA can restrict purpose of certificate

Public Key Infrastructure

Public Key Infrastructure

> *A public key infrastructure (PKI) is a set of roles, policies, hardware, software and procedures needed to create, manage, distribute, use, store and revoke digital certificates. [Wikipedia]*

All of these components serve a purpose and follow rules so that:

- A certificate user (end entity) can be assured
- By a trustworthy certification authority
- That a PK belongs to another end entity (person, server, . . .)
- And can be used for a given purpose
- Under well-defined rights/responsibilities for all parties

PKI Architecture

```
+---+
| C |                       +------------+
| e | <-------------------->| End entity |
| r |       Operational     +------------+
| t |       transactions          ^
| i |      and management         |  Management
| f |       transactions          |  transactions        PKI
| i |                             |                     users
| c |                             v
| a | =======================  +--+------------+  ==============
| t |                          ^               ^
| e |                          |               |         PKI
|   |                          v               |      management
| & |                       +------+           |       entities
|   | <---------------------|  RA  |<----+     |
| C |  Publish certificate  +------+     |     |
| R |                                    |     |
| L |                                    |     |
|   |                                    v     v
| R |                                +------------+
| e | <------------------------------|     CA     |
| p |   Publish certificate          +------------+
| o |   Publish CRL                     ^      ^
| s |                                   |      |  Management
| i |                +------------+     |      |  transactions
| t | <--------------| CRL Issuer |<----+      |
| o |   Publish CRL  +------------+            v
| r |                                      +------+
| y |                                      |  CA  |
+---+                                      +------+
```

Operational/Management transactions

How do certificates go around?

Operational/Management transactions

How do certificates go around?

Operational protocols specify how certificates are:

- stored in repositories (e.g., LDAP)
- transferred to client software (HTTP, FTP, MIME)
- encoded in non-ambiguous formats

Operational/Management transactions

How do certificates go around?

Operational protocols specify how certificates are:

- stored in repositories (e.g., LDAP)
- transferred to client software (HTTP, FTP, MIME)
- encoded in non-ambiguous formats

You have seen several instances of operational protocols:

- In TLS the RFC specifies how certificates are exchanged
- In S/MIME certificates are included in the PKCS#7 attachments
- OS certificates are managed via standard cryptographic modules

PKI Management: Initialization

We asked an important question before:

- How do users get to *know* a CA
- How does Bob verify a CA signature in a certificate?

Answer:

- All public keys are encoded in X.509 certificates
- **Some certificates contain the public keys of CAs**

PKI Management: Initialization

We asked an important question before:

- How do users get to *know* a CA
- How does Bob verify a CA signature in a certificate?

Answer:

- All public keys are encoded in X.509 certificates
- **Some certificates contain the public keys of CAs**
- Bob obtains the CA's public key from a certificate
- Bob uses the CA's PK to verify signature on Alice's certificate
- If certificate OK $\Rightarrow$ Bob can use Alice's public key

PKI Management: Initialization

We asked an important question before:

- How do users get to *know* a CA
- How does Bob verify a CA signature in a certificate?

Answer:

- All public keys are encoded in X.509 certificates
- **Some certificates contain the public keys of CAs**
- Bob obtains the CA's public key from a certificate
- Bob uses the CA's PK to verify signature on Alice's certificate
- If certificate OK $\Rightarrow$ Bob can use Alice's public key

Therefore, Alice's public key is authenticated if:

- Bob has certificate for CA that issued Alice's certificate
- Bob trusts CA to have checked data on Alice's certificate

PKI Management: Initialization (2)

How does Bob know to trust CA?

PKI Management: Initialization (2)

How does Bob know to trust CA?

In the simplest settings:

- Bob gets certificate directly from CA
- Bob implicitly trusts CA certificate

Examples:

- We get many CA certificates pre-installed in OS
- Portuguese citizen's card is certified by state-run CA

These are examples of initialization operations.

Key generation, if done by the end entity, also part of initialization.

PKI Management: Registration and Certification

Registration Authorities (RA):

- Front-end: direct contact with end-entities
- Responsible for checking data that goes into certificates
- Responsible for ensuring (unique) entity possesses secret key

Certification Authorities:

- Back-end: infrastructure where certificates are signed
- Typically high-security: air gaps, physical security, etc.

PKI Management: Registration and Certification

Registration Authorities (RA):

- Front-end: direct contact with end-entities
- Responsible for checking data that goes into certificates
- Responsible for ensuring (unique) entity possesses secret key

Certification Authorities:

- Back-end: infrastructure where certificates are signed
- Typically high-security: air gaps, physical security, etc.

Example: Portuguese Citizen's Card

- RA is Registo Civil, Loja do Cidadão, etc.
- CA is deployed in protected facilities at INCM
- CA generates keys, signs certificates and issues smartcards
- RA delivers them to citizens after physical identification

PKI Management: Revokation

Certificates outside of validity dates are, by definition, invadid.

What happens if they need to be invalidated?

- E.g., lost secret key, data breach, meta-data becomes incorrect.

PKI Management: Revokation

Certificates outside of validity dates are, by definition, invadid.

What happens if they need to be invalidated?

- E.g., lost secret key, data breach, meta-data becomes incorrect.

Certificates need to be **revoked** *while they still look valid*.

This is formally done using Certificate Revokation Lists (CRL):

- CA periodically publishes a black-list of revoked certificates
- Certificate consumers should check most-recent CRL
- Exceptional CRL may also be published, as best-effort

How do we get revokation information?

Certificate extensions typically indicate URLs for CRLs

Traditionally low support from client software

PKI Management: Revokation (2)

Three solutions used in the real-world.

1 - Trusted Service Provider Lists (TSL):

- up to date white list of trusted certificates
- closed small groups (e.g., banking) and high-security applications

2 - On-line Certificate Status Protocol (OCSP)):

- a trusted server checks CRLs for you
- usually managed by CAs themselves
- typically used in large organizational contexts (e.g., eGov)

3 - Certificate pinning:

- web servers/browsers/applications carry their own white lists
- identify *good* certificates for important entities (e.g., Google)

Certificate Chains and CA Hierarchy

We have seen a simple case: Bob trusts Alice's CA implicitly.

In general this is not the case:

Certificate Chains and CA Hierarchy

We have seen a simple case: Bob trusts Alice's CA implicitly.

In general this is not the case:

- Bob is initialized with certificates for *root* CAs
- Bob trusts implicitly in these CAs

Certificate Chains and CA Hierarchy

We have seen a simple case: Bob trusts Alice's CA implicitly.

In general this is not the case:

- Bob is initialized with certificates for *root* CAs
- Bob trusts implicitly in these CAs
- Certificates for root CAs are self-signed:
  - CA generates a key pair $(\mathsf{sk}, \mathsf{pk})$
  - CA creates its own certificate with subject = issuer = CA name
  - Certificate includes pk and CA signs it with sk

Certificate Chains and CA Hierarchy

We have seen a simple case: Bob trusts Alice's CA implicitly.

In general this is not the case:

- Bob is initialized with certificates for *root* CAs
- Bob trusts implicitly in these CAs
- Certificates for root CAs are self-signed:
  - CA generates a key pair $(\mathsf{sk}, \mathsf{pk})$
  - CA creates its own certificate with subject = issuer = CA name
  - Certificate includes pk and CA signs it with sk

Note: self-signed certificates can be generated by anyone.

Validating a self-signed certificate implies:

- belief that whoever owns that secret key is a CA
- belief that this CA only generates *good* certificates

Certificate Chains and CA Hierarchy (2)

Root CAs typically do not issue end-entity certificates.

- There is a hierarchy of CAs
- If CA A signs certificate of CA B
- Then trust in CA B $\leq$ trust in CA A

We can have many levels in this hierarchy/tree, so:

- To authenticate Alice's public key, Bob gets Alice's certificate
- To validate Alice's certificate, Bob gets certificate of Alice's CA
- Bob verifies that Alice's certificate is valid wrt Alice's CA

Bob still needs to decide whether to trust Alice's CA.

Trust = Alice's CA is descendent of Root CA trusted by Bob

Certificate Chains and CA Hierarchy (3)

Bob enters a loop starting with Current CA = Alice's CA.

The loop works as follows:

- If Bob implicitly trusts Current CA certificate: Accept!
- Else If Current CA is subordinate to some $\widehat{\text{CA}}$:
  - Bob gets $\widehat{\text{CA}}$ certificate
  - Bob verifies Current CA certificate is valid wrt $\widehat{\text{CA}}$
  - Bob re-enters loop with Current CA = $\widehat{\text{CA}}$
- Else Reject!

Note: this process fails if Bob cannot get certificates

- All certificates can be sent by Alice except the root of trust.

Certificate Policies

PKI can be used to give cryptography a legal meaning.

A *Certificate Policy* is a set of PKI operation rules:

- Rights and responsibilities of end-entities
- Rights and responsibilities of CAs

These rights and responsibilities can be written in law.

A certificate policy is assigned an object identifier (OID):

- Certificates can be flagged to comply with a policy

This implies an accreditation system:

- CA must be audited before it is authorized to use OID
- Any CA that uses OID without authorization is breaking the law


THE END