PV181 Laboratory of security and applied cryptography

Introduction to Applied Cryptography

Part 2, seminar 3: Asymmetric cryptography & OpenSSL

Łukasz Chmielewski
chmiel@fi.muni.cz

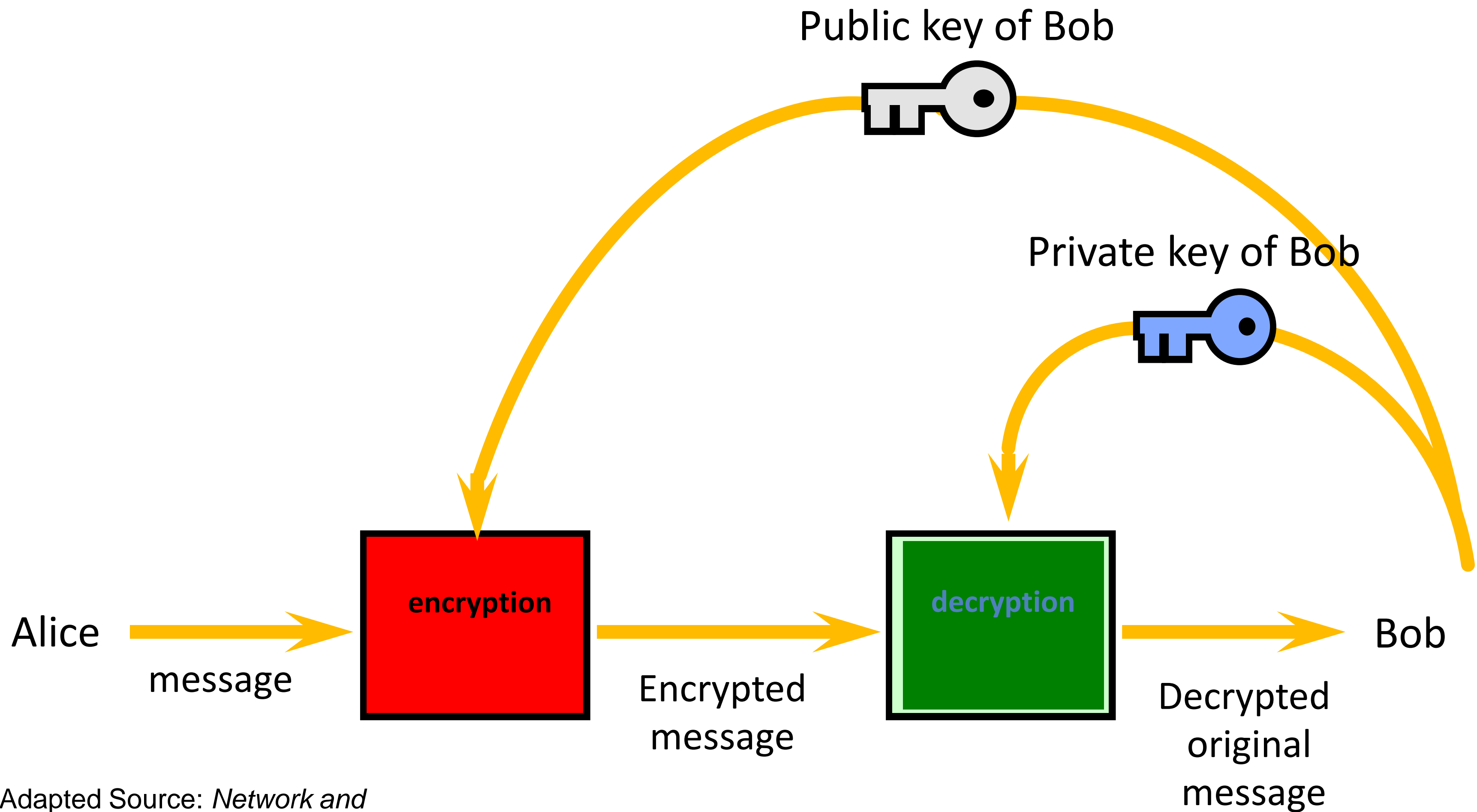
Public vs private key cryptography

- Private (symmetric)
  - both parties share secret (**private**)
  - Pros: fast encryption
  - Cons: key distribution requires **secure channel**
- Public (asymmetric)
  - one key is **public**
  - Pros - key distribution – **insecure** channel is OK
  - Cons - slow encryption
- Practice - private + public:
  - **public** used to establish key for **private** key system

Asymmetric cryptography

- Two related keys – created by **one** party
  - different inverse operations (encryption - decryption, signing – signature verification)
- Properties - hard to compute private from public key
  - based on hard mathematical problems
- Hard problems and cryptosystems:
  - Integer factorization – RSA, Rabin, …
  - Discrete logarithm problem (DLP): ElGamal, EC, DSA, …
  - Others (DH, decoding,…) – Diffie-Helman, McElliece,…

Asymmetric cryptosystem



Adapted Source: *Network and Internetwork Security* (Stallings)

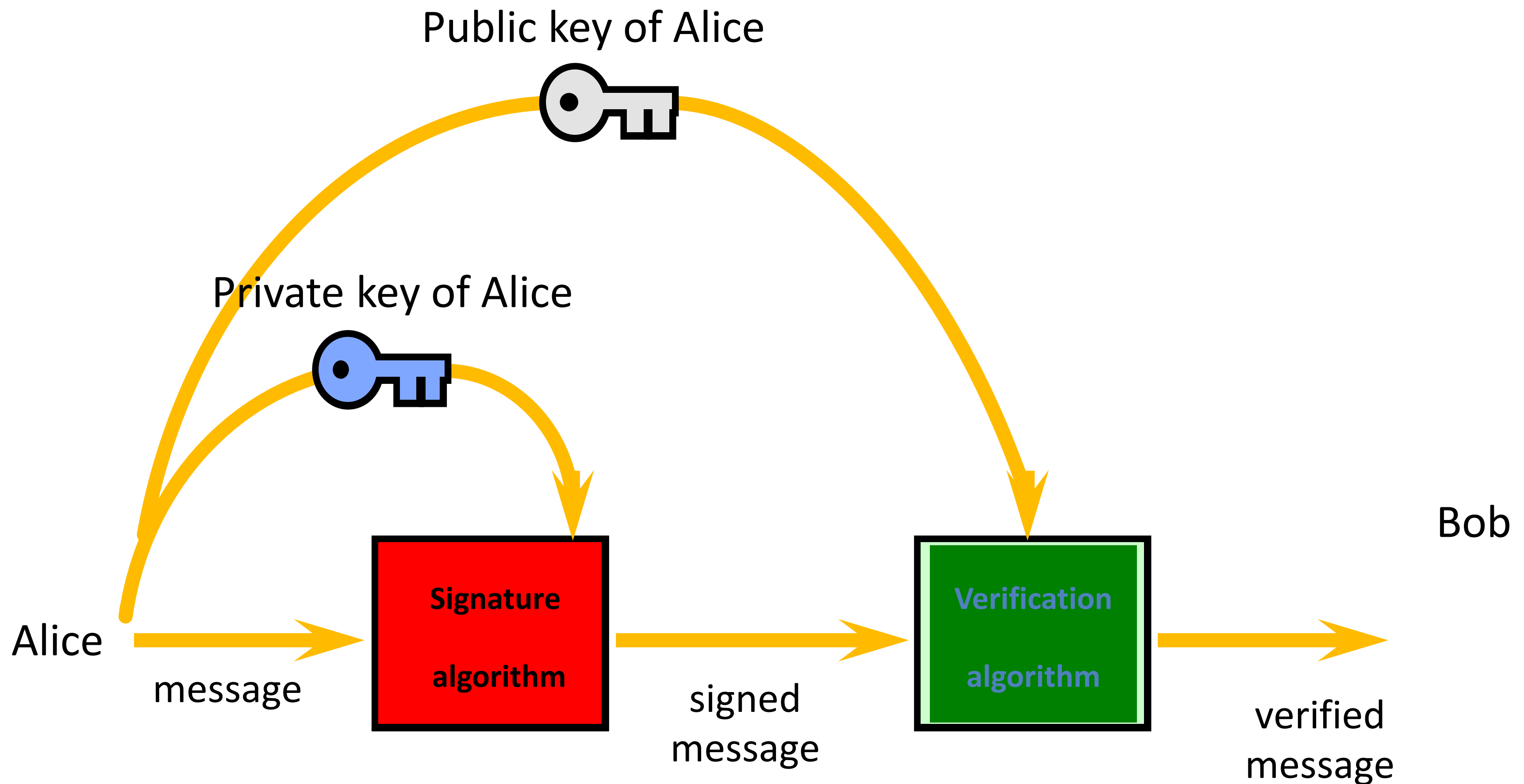
Asymmetric cryptosystem

- Bob generates both keys:
  - Public is sent to Alice
  - Private is kept secret
- Alice encrypts a message with her public key and sends it to Bob
- Bob decrypts the ciphertext using his private key
- Are big messages encrypted?
  - Usually not. Only symmetric keys are encrypted and those are used to encrypt big messages. Why?
    - Symmetric crypto is more efficient than asymmetric.

Digital signature

- Asymmetric cryptography
  - Private key – signature generation (usually only **hash** of data is signed **not** data itself)
  - Public key – a verification procedure
- Data integrity + data origin + non-repudiation:
- Non-repudiation - correct signatures can be generated only by those with the private key – differently than for MAC!
- The digital signature itself does not give any guarantees concerning signing time.

Digital signature scheme



Source: *Network and Internetwork Security* (Stallings)

Digital signature

- Alice generates key pair
  - Public key is published (sent to Bob) for verification of signature
- Alice sign a document using her private key
- Bob use public key to verify the digital signature
- Classical examples: RSA, ECC
- PQC example: Dilithium

RSA: mathematics

1. Secret primes p, q: $n = p \cdot q$
2. Public exponent e:
$$\gcd(e, (p-1)) = \gcd(e, (q-1)) = 1$$
3. Private exponent d: $d \cdot e \equiv 1 \; mod \; \varphi(n)$

Encryption (public n, e): $E(m) = m^e \; mod \; n = c$

Decryption (private n, d): $D(c) = c^d \; mod \; n = m$

- RSA-1024: means n has 1024 bits and $m < n$
  - Is 1024 bit secure?

RSA: example

- Intentionally small numbers (**not** secure)
- We generate parameters:
  - $p = 17, q = 7, n = p \cdot q = 119$
- Public exponent is selected:
  - $e = 3$ is wrong, because $\gcd\big(3, (p-1)\cdot(q-1)\big) = \gcd(3{,}96) = 3$
  - $e = 5$, because $\gcd(5{,}96) = 1$
- Private exponent is computed:
  - $e \cdot d = 5 \cdot d = 1 \pmod{96}$ to have $d = 77$
- The public key is: $(n = 119, e = 5)$
- The private key is: $(n = 119, d = 77)$
- Encryption/decryption:
  - Message $m = \text{"C"} = 65$
  - Encryption $m' = 65^5 \bmod 119 = 46$
  - Decryption $m = 46^{77} \bmod 119 = 65$

RSA Padding example (PKCS#1 v1.5)

- Document
  - "00 01 02 03 04 05 06 07 07 06 05 04 03 02 01"
- Hash of the document (sha-1)
  - "b3 39 90 4c d2 a0 10 e6 19 37 eb e5 b5 83 37 8c 5d 10 51 95"
- Padded hash
  - "00 01 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 00 30 21 30 09 06 05 2b 0e 03 02 1a 05 00 04 14 b3 39 90 4c d2 a0 10 e6 19 37 eb e5 b5 83 37 8c 5d 10 51 95"

RSA in practice: Various Paddings

- $\mu(M)$ = 6b bb … bb ba || Hash(M) || 3x cc
  where $x = 3$ for SHA-1, 1 for RIPEMD-160
  - ANSI X9.31

- $\mu(M)$ = 00 01 ff … ff 00 || HashAlgID || Hash(M)
  - PKCS #1 v1.5

- $\mu(M) = 00 \,||\, H \,||\, G(H) \oplus [salt \,||\, 00 \ldots 00]$
  where H = Hash($salt$, M), $salt$ is random, and G is a mask generation function
  - Probabilistic Signature Scheme (PSS)

Hard problems

- Integer factorization
  - for **n** find divisor **p** of **n**
- Discrete logarithm problem:
  - in $Z_{\boldsymbol{p}}$, Elliptic curves (EC)
    for $y = g * g * \cdots * g = g^x$ find x
  - $*$ represents operation (*, +) for given domain (integers, EC)
  - Domain parameters:
    - $g, n = ord(g)$ - n should be large
    - params defining algebraic structure: $Z_{\boldsymbol{p}}$ or **EC**

Digital Signature Standard (DSS)

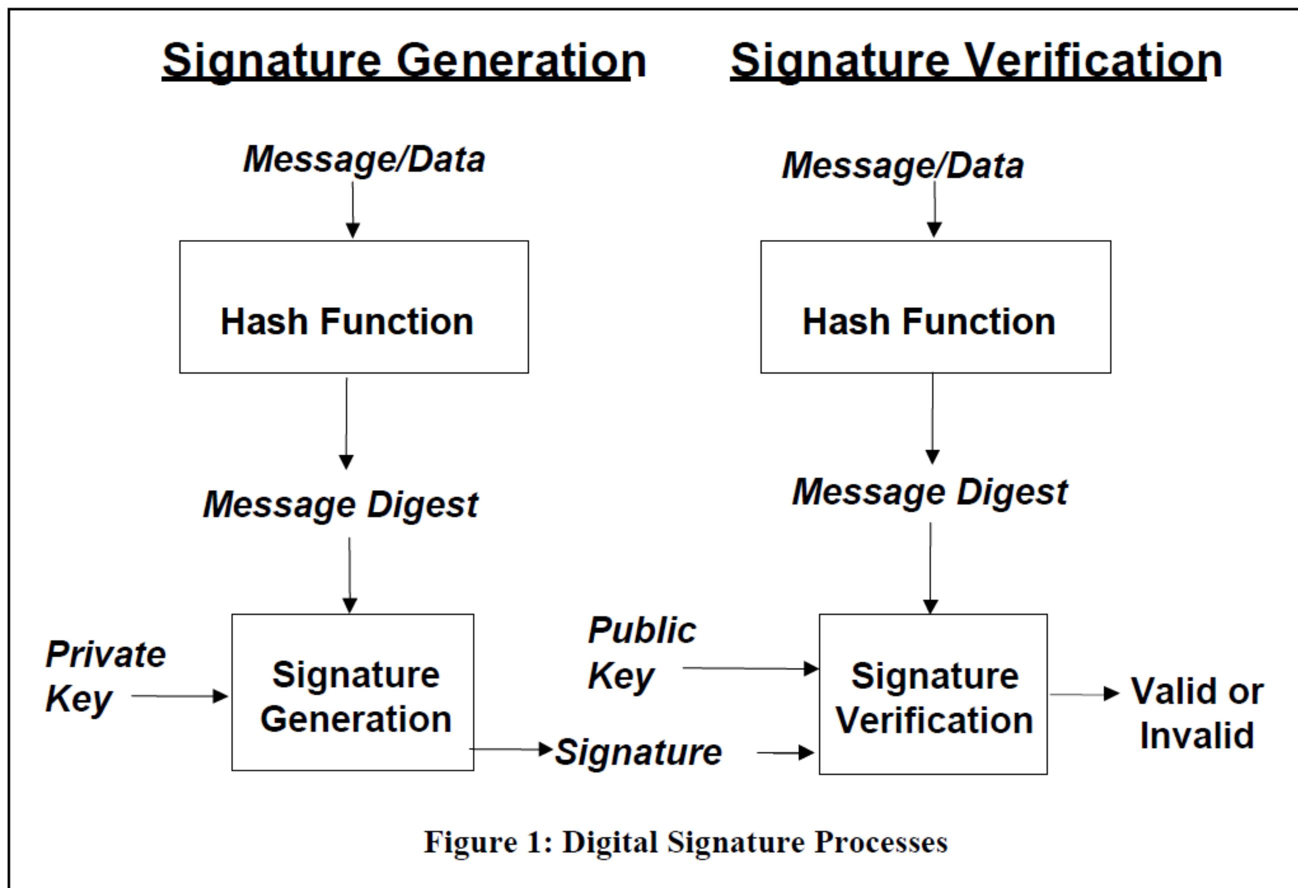


Figure 1: Digital Signature Processes

Digital Signature Algorithm (DSA)

- Proposed in 1991 by NIST
- In 1994 the selection procedure for Digital Signature Standard (DSS) was concluded – DSA (Digital Signature Algorithm) was selected.
- Modified version of ElGamal algorithm, based on discrete logarithm in Z_p.
- Became FIPS standard FIPS 186 in 1993.
- Slightly modified in 1996 as FIPS 186-1.
- Extended in 2000 as FIPS 186-2.
- Updated in 2009 as FIPS 186-3 (new key sizes).

- Now NIST FIPS 186-3 supports RSA & DSA & ECDSA.

DSA: keys

- Key generation
  - Choose random **x**, such that $0 < x < q$.
  - Calculate **y** = $g^x \bmod p$.
- Private key: **x**.
- Public key: y & (p, q, g).

DSA: math recall

- Signature generation
  - Generate a random per-message value **k** such that $0 < k < q$.
  - Calculate **r** = $(g^k \bmod p) \bmod q$
  - Calculate **s** = $(k^{-1}(H(m) + x*r)) \bmod q$
  - The signature is (r, s).
- Signature verification
  - **w** = $(\mathbf{s})^{-1} \bmod q$
  - **u1** = $(\mathbf{H(m)}*w) \bmod q$
  - **u2** = $(\mathbf{r}*w) \bmod q$
  - **v** = $((g^{u1}*y^{u2}) \bmod p) \bmod q$
  - The signature is valid if **v = r**
- For DSA (1024,160) the signature size will be 2x160 bits.

Elliptic curve DSA (ECDSA)

- Elliptic curves invented by Koblitz & Miller in 1985.
- ECDSA proposed in 1992 by Vanstone
- Became ISO standard (ISO 14888-3) in 1998
- Became ANSI standard (ANSI X9.62) in 1999

- ECDSA is a version of DSA based on elliptic curves.
- More about this topic later…

Digital certificate

Digital certificate

- is used to prove ownership of the public key
- binds a public key to identity (identity, email,…)
- **Public key certificate** is **signed** by a trusted third party – Certification Authority (CA)
- two models: centralized and decentralized

Digital certificate – typical use case

- Two-way authentication Alice and Bob can verify each other's public key and identity with their corresponding certificates obtained from CA.
- Alice and Bob get each other's key through the corresponding certificates and not directly.
- In practice, business transactions rely on one-way authentication
- Example
  - When a client (my laptop) establishes a connection with Amazon, it is essential that the client authenticates the website; The company does not really care who the client is as long as the payment information is correct.
  - The client will request Amazon's certificate, verify its validity and then send the encrypted session key to Amazon's website.

Trust models

- Public key infrastructure (PKI)
  - centralized – hierarchy of CA's
  - cert signed by party
  - used in web browsers
  - standard X.509
- Web of trust
  - decentralized model
  - signed by many parties
  - used in PGP, GPG
  - standard OpenPGP

Public Key Infrastructure (PKI)

- set of roles and procedures:
  - issue, maintain, administer, revoke, suspend, reinstate, and renew digital certificates
  - create and manage a public key repository
- Certification Authority (CA) – stores, issues, signs certs
- Registration Authority (RA) – verifies the identity, could be part of CA
- Central directory– cert requests issued and revoked,
- Management system
- Cert policy

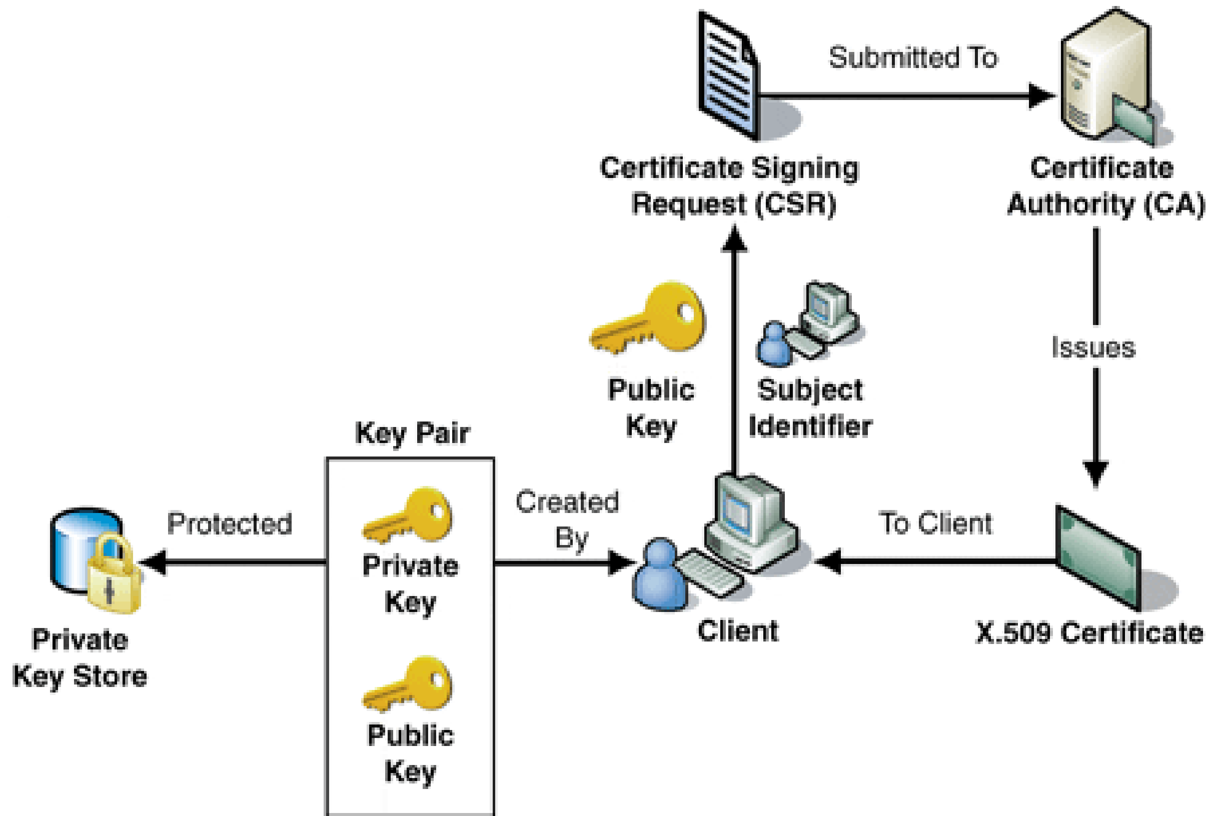
X.509 PKI certificate

- Certification Authority – trusted third party
- Certificate revocation lists (CRL) – certificates no longer be trusted (compromised key, CA,…)
- RFC5280 – defines format and semantics of certs and CRLs
- X.509 versions 1,2,3

X.509 PKI certificate content

Serial Number: unique ID of cert
Subject: ID of entity
Signature algorithm:
Signature:
Issuer: verifier of info and issued cert
Valid–From: date cert is first valid from
Valid–To: expiry date
Key-Usage: purpose of PK (signature, cert signing, …)
Public Key:
Thumbprint algorithm: to compute hash of PK cert
Thumbprint (fingerprint): hash of abbreviated PK cert

Certificate issuing



https://help.bizagi.com/process-modeler/en/index.html?cloud_auth_certificates.htm

Types of Certificate

- Extended Validation (EV)
  - Issued only after rigorous identity verification to check the legitimacy of the applicant (organization). It may include the verification of the legal and physical existence of the organization; cross verification from other records (govt. and other public/private records).
  - Most expensive certificate and it may take several days to issue the certificate.
- Organization Validation(OV)
  - Less rigorous; only the existence and domain of the organization are verified.
- Domain Validation (DV)
  - Lowest level validation; Least expensive and issued in few minutes
  - Confirmed that an applicant has the right to use a specific domain name

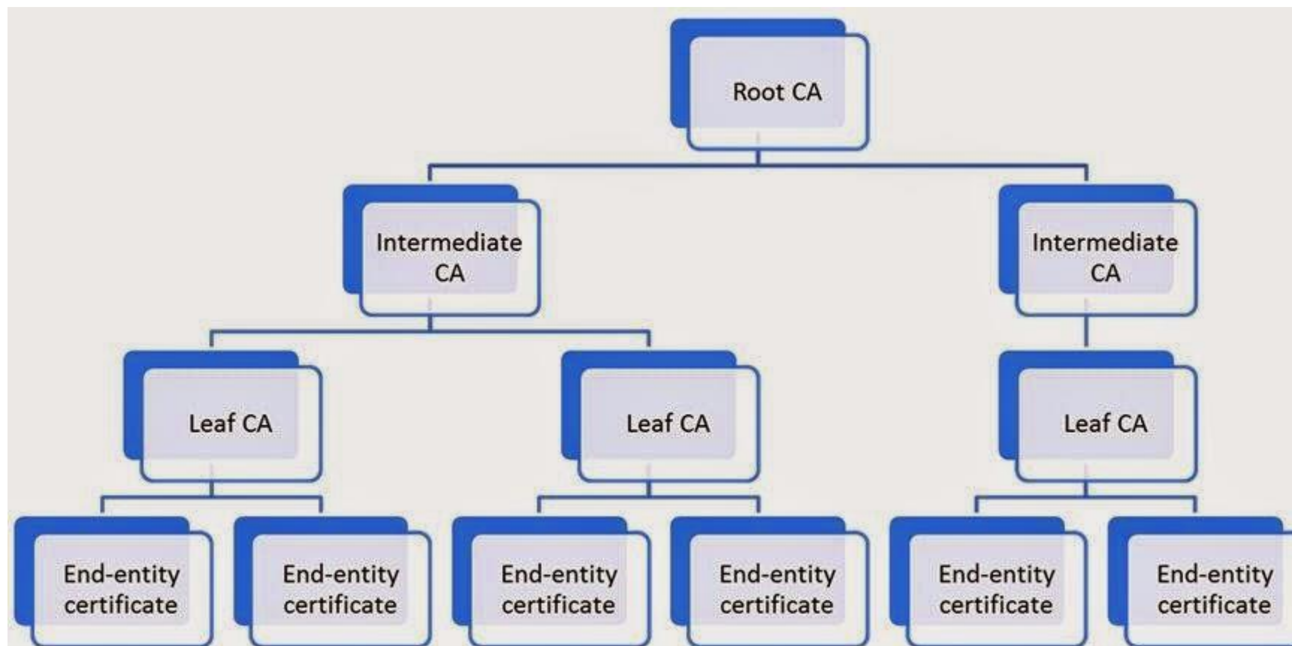
Certificate verification

Checking single cert:

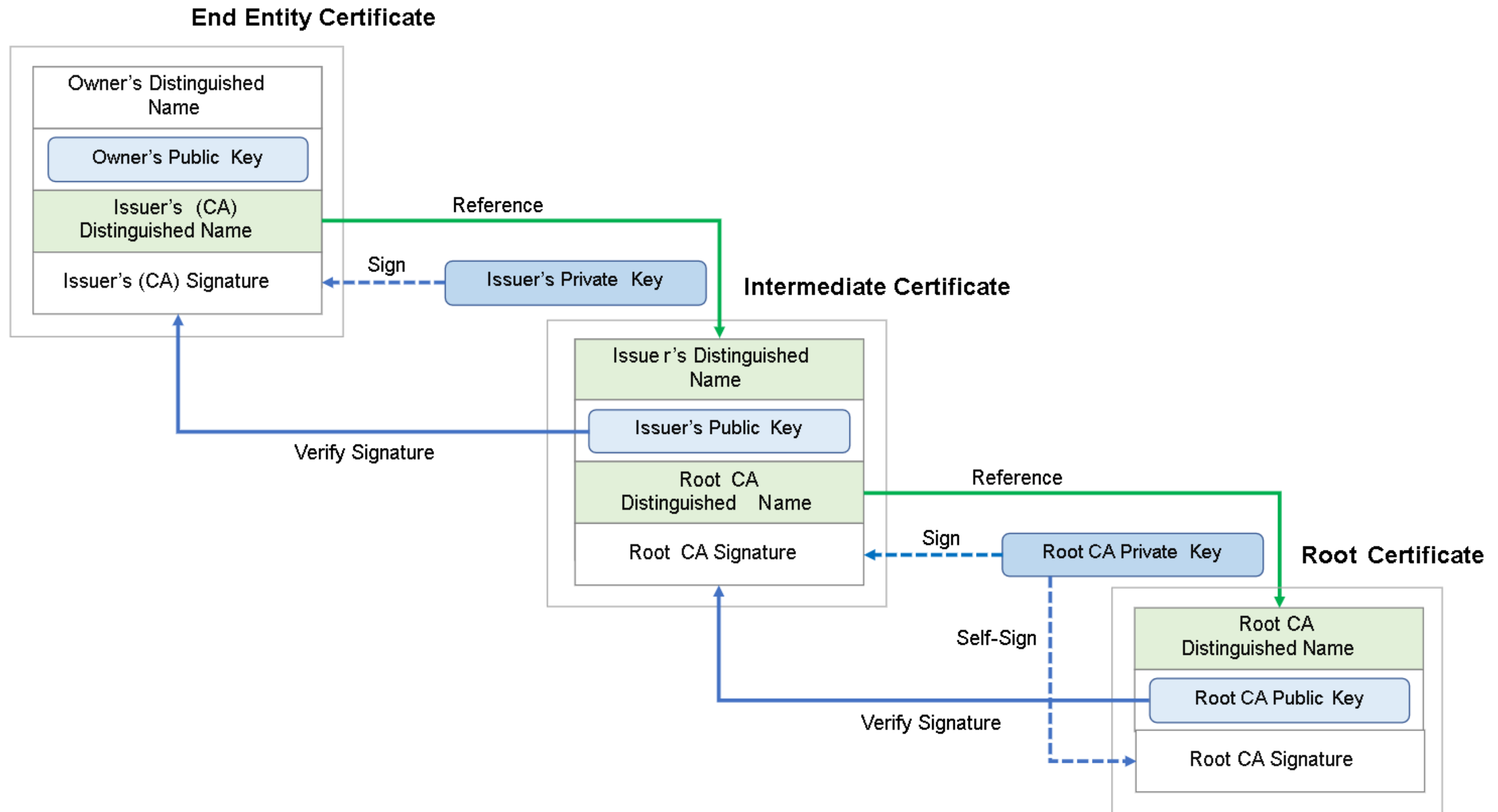
- current **date** against validity period
- current validity of CA public key
- signature of CA on cert
- check whether the certificate is revoked
- policies

Certificates hierarchy

- **root CA (trust anchor)** - self-signed certificate
- Intermediate CA's
- **End entity** – user certificate



Chain of trust



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

Certificate validation path

Input: cert path, trust anchor

Path validation:

1. Check all certs if still valid
2. Check revocation status of certs
3. Check issuer = of previous cert subject
4. Check policy constraints
5. …

Revocation

- Reasons for revocation
  - **key compromise** (most common), CA compromise, affiliation change,…
- Two states:
  - revoked – irreversibly for compromised private key
  - hold – unsure user about key compromising, can be reinstalled
- Checked using:
  - CRL – list of revoked certs
  - Online Certificate Status Protocol – on demand

Seminar Tasks

- After you do an exercise discuss the result in pairs.
- Try to explain what is happening.
- Two parts:
  - OpenSSL command line tool
  - python
- Good luck! ☺

OpenSSL and public-key crypto

- A command line tool for using various cryptographic functions
- Can be used for (within this course)
  - Create and manage public and private keys
  - Calculation of Message digest
  - Public-key cryptographic operations
  - Encryption and decryption with Ciphers
  - Creation of X.509 certificates, CSRs and CRLs

Public Key Encryption / Decryption

- Generate key pairs:
  `openssl genrsa -aes128 -out alice_private.pem 1024`
- Extract the public key
  `openssl rsa -in alice_private.pem -pubout > alice_public.pem`
- Encrypt with Alice's public key
  `openssl pkeyutl -encrypt -inkey alice_public.pem -pubin -in alice.txt -out outfile.enc`
- Decrypt with Alice's private key
  `openssl pkeyutl -decrypt -inkey alice_private.pem -in outfile.enc > test.txt`
- Try it with 2048 bit RSA key. Try to generate DSA key pair with 3DES. How does it compare to RSA?

OpenSSL: CSR and self-signed certificate

- Create a certificate signing request (CSR)

```
openssl req -key alice_private.pem -new -out alice_domain.csr
```

- Create both key and CSR with one command

```
openssl req -newkey rsa:4096 -keyout alice_private.pem -out
alice_domain.csr
```

- Create a self-signed certificate

```
openssl x509 -signkey alice_private.pem -in alice_domain.csr -req
-days 365 -out alice_domain.crt
```

- You can verify your CSR (this checks the signature of the file)

```
openssl req -text -in alice_domain.csr -noout -verify
```

  What happens when you modify the signature or any field of the certificate?
- Create a CSR with the key you have generated and self-sign it.
  - Which fields do you identify?
- An OpenSSL cook book (for quick reference): https://www.feistyduck.com/books/openssl-cookbook/

OpenSSL: CSR and certificate cont'd

- Check the (SSL) key and verify the consistency
  `openssl rsa -in alice-private.key -check -noout`
  (without printing the key)
- The hash values of the certificate and key; hash values can be compared to verify the certificate and key match
- `openssl x509 -noout -modulus -in alice_domain.crt | openssl sha256`
- `openssl rsa -noout -modulus -in alice_private.pem | openssl sha256`
- Try it with your key and self-signed certificate.

Python – RSA (hazmat)

- Let’s play first a bit with python.
- Have a look at demo.py.
  - There are tasks to be done. The goal is to do them till 3.
- Task 6 & 7 are only for enthusiast ☺
- Use: https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/

- Good luck!

Assignment 3

- This is a programming assignment. Please upload your scripts/code via the course webpage.
- The deadline for submission is October 11th, 8:00 (then -3 points per each started 24h).
- Please name the submission file as <uco_number>_hw3.zip. Put there both the python code and the openssl commands (e.g., in the readme file). If you have more files pack them together to the zip file.
- The code must contain comments so that it is reasonably easy to understand how to run the script for evaluating each answer.

Assignment 3 - Tasks

1. Use the openssl command tool to encrypt alice.txt with AES128 in OFB mode (with cryptographically secure IV and key). Do the same (with the same key and IV) using the python cryptography library (like in Assignment 1). Give the two outputs in two different files and attach them to your solution. Also, attach the openssl command that you used.
   Remark: make sure that your code would work for large files. **[3 points]**
2. Test that step 1 works fine with openssl (for decryption). Attach the command to your solution. **[1.5 point]**
3. Suppose you are a trusted CA and you receive Bob's CSR. Write a python script to generate Bob's certificate. Write a script that checks that the certificate and the key match. **[3 points]**
4. Suppose Alice wants to verify Bob's certificate issued by you (from step 3). Write a python function that takes the necessary inputs and verifies the validity of Bob's certificate. Your function must raise an error in case the certificate is not issued by you. **[2.5 points]**
5. **Extra:**
   Generate and send me an email (lukchmiel@gmail.com) encrypted with my public key and signed with your private key.
   For the sake of simplicity, I posted the public key in the study materials for the seminar (Lukasz Chmielewski lukchmiel@gmail.com-(0xF077D43514C58924)-public.asc).
   Make sure that I can learn your public key (e.g., attach it to the email).
   Describe briefly how you performed Step 5 in the email. **[1 point]**
   Note: do not use keys generated by third parties (e.g., websites)

Good luck!!!