# PV181 Laboratory of security and applied cryptography

## Introduction to Applied Cryptography

### Part 2, seminar 3: Asymmetric cryptography & OpenSSL

David Rajnoha and Łukasz Chmielewski
(based on the slides of Marek Sys)
Email: 492758@mail.muni.cz and chmiel@fi.muni.cz
Consultations (Łukasz): A406, 9.00-11.00 on Fridays

CROCS
Centre for Research on Cryptography and Security

# The Topic for Today

- Working with **OpenSSL**
- **Public keys**, encryption, signing
- **Public Key Certificates** and the **Chain of Trust**
- In assignment, we will return to Python Cryptography package
  - https://cryptography.io/en/latest/

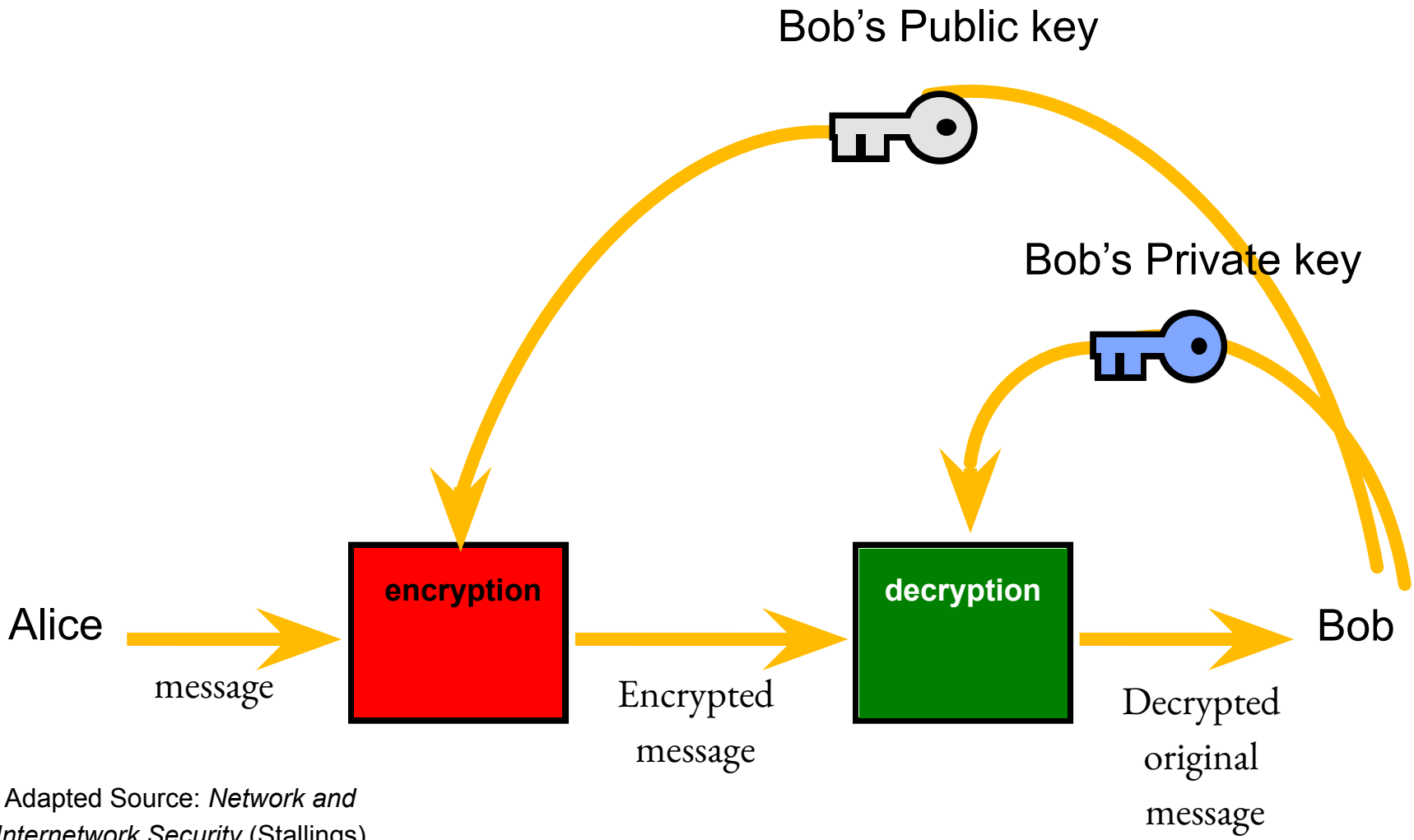
# Plan for Today

- Crypto with OpenSSL
- Digital certificates
- Certificates with OpenSSL
- Trust Models
- Building the chain of trust

# 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

# Asymmetric cryptosystem – RSA




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

# RSA with OpenSSL

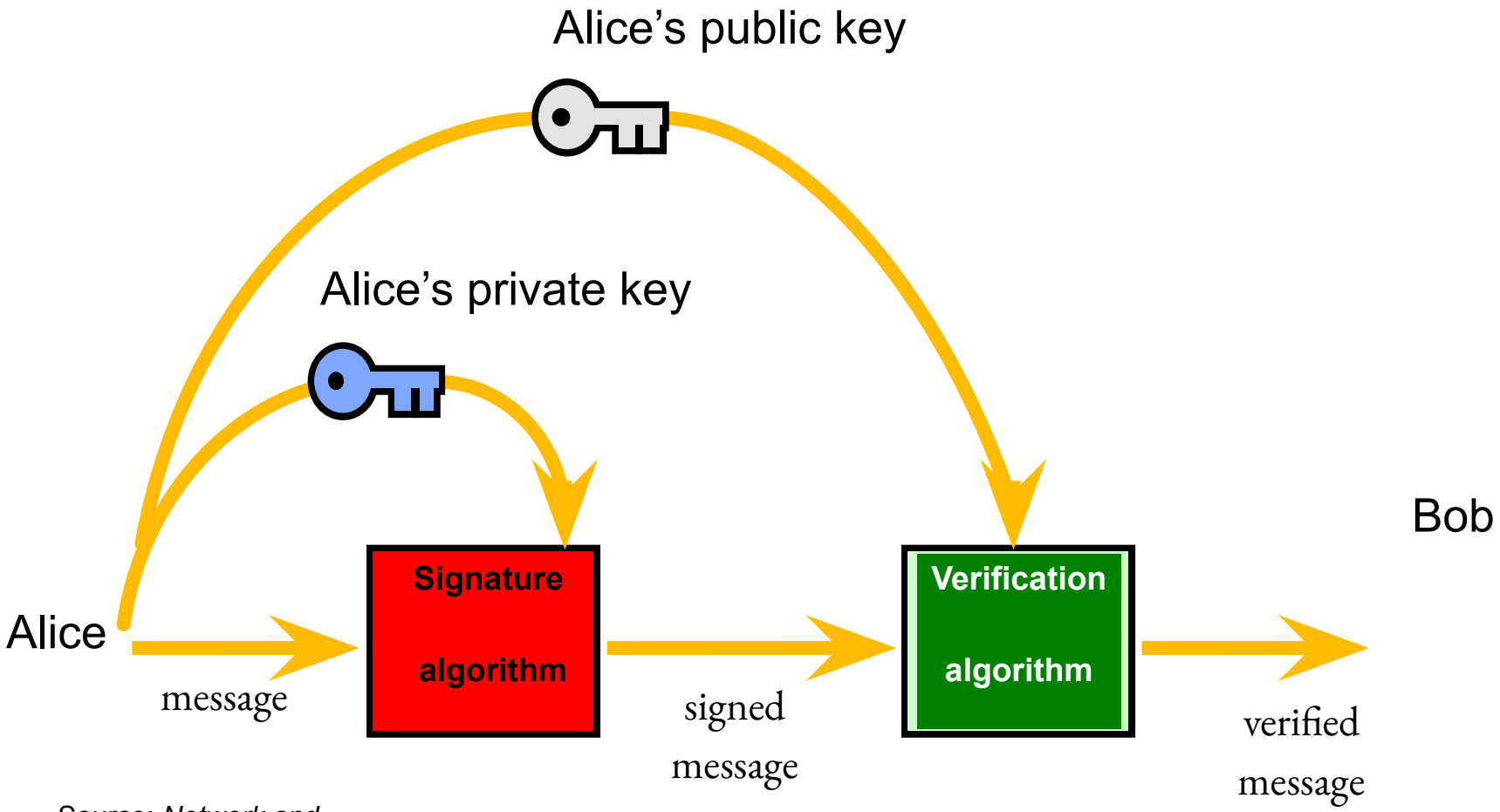
- Generate key pairs
  `openssl genrsa`
- Extract the public key
  `openssl rsa`
- Encrypt and decrypt
  `openssl pkeyutl`

# Task 1: OpenSSL genrsa

- Generate a RSA4096 key using OpenSSL
- Extract the public key from the generated key
- Encrypt bob.txt into ciphertext.txt
- Decrypt back ciphertext.txt

- Try to encrypt file alice.txt with Alice's public key into ciphertext.txt – does it work?

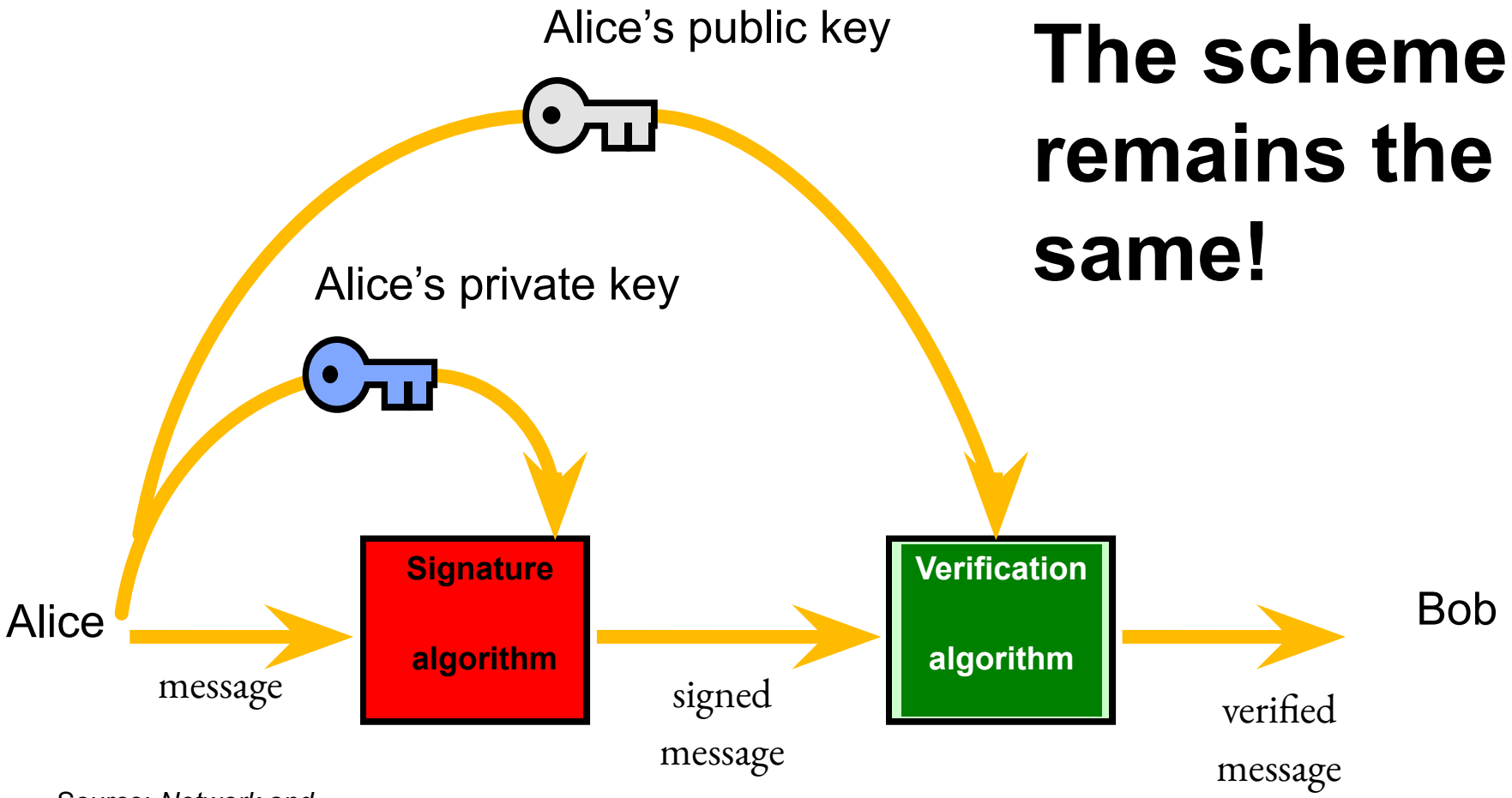
# Digital signature scheme




Source: *Network and Internetwork Security* (Stallings)

| | RSA | DSA | ECDSA |
|---|---|---|---|
| Keys | Public + Private Key Pair | Public + Private Key Pair | Public + Private Key Pair |
| Nonce Required | No | Yes | Yes |
| Underlying Principle | Prime Factorization | Discrete Logarithm | DL on Elliptic Curves |
| Equivalent Key Size | 3072 bits | 3072 bits | 256 bits |
| Signing Performance | Slower | Faster | Fastest |
| Verification Perf. | Faster | Slower | Fast |
| Vulnerabilities | Improper Padding, Small Key Sizes | Nonce Reuse, Weak RNGs | Nonce Reuse, Side-Channel Attacks |
| Security Level | High (with large key sizes) | High (but depends on nonce security) | High (with smaller key sizes) |
| Use Cases | General-purpose, widely adopted | Digital Signatures (FIPS compliant) | Modern applications (cryptocurrencies, SSL/TLS) |

# Digital signature scheme




**The scheme remains the same!**

Source: *Network and Internetwork Security* (Stallings)

# ECDSA with OpenSSL

- Generate key pairs
  `openssl ecparam`
- Generate corresponding public key
  `openssl ec`
- Sign and verify
  `openssl pkeyutl`
  `openssl dgst`

# Task 2: OpenSSL ECDSA

- Generate an ECDSA keypair using OpenSSL
  - It will be also used in further tasks.
- Sign a file using the private key
- Verify the signature using the public key
- What is the difference between `openssl pkeyutl` and `openssl dgst` ?

# Digital Certificates

- **Purpose**: Prove ownership of a public key.
- **Function**: Bind a public key to an identity (e.g., name, email).
- **Verification**: Signed by a trusted third party called a Certification Authority (CA).
- **Models**: Implemented through centralized and decentralized systems.

# Public Key Infrastructure (PKI)

- A framework of roles and procedures for managing digital certificates and public keys.
  - Issue, maintain, revoke, suspend, reinstate, and renew digital certificates.
  - Create and manage a repository for public keys.
- 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
- Certificate policies

# 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

# Self-signed certificate

- **Signed directly by the requester** rather than a trusted third party.
- **Contradicts the traditional role** of certificates, which typically rely on a trusted CA to establish credibility.
- **Testing Purposes**: Often used in development and testing environments where establishing trust is unnecessary.
- **Internal or Limited Use**: Suitable in scenarios where authority verification is not required, but the format of a certificate is necessary.

# 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

- Sign CSR to create certificate
  ```
  openssl x509 -req -in alice_domain.csr -CA ca_cert.crt -CAkey
  ca_key.pem -out alice_domain.crt -days 365
  ```
- 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.

# Task 3: Self-signed

- Generate a self-signed x509 certificate using OpenSSL for the key created in the task 2
  - Use sha256
  - Set the validity for 1 year
  - Use the interactive mode
- What is the option for self-signed certificate, what is the one for creating CSR?
- What additional information were you asked for?

# Task 4: Certificate Signing Request

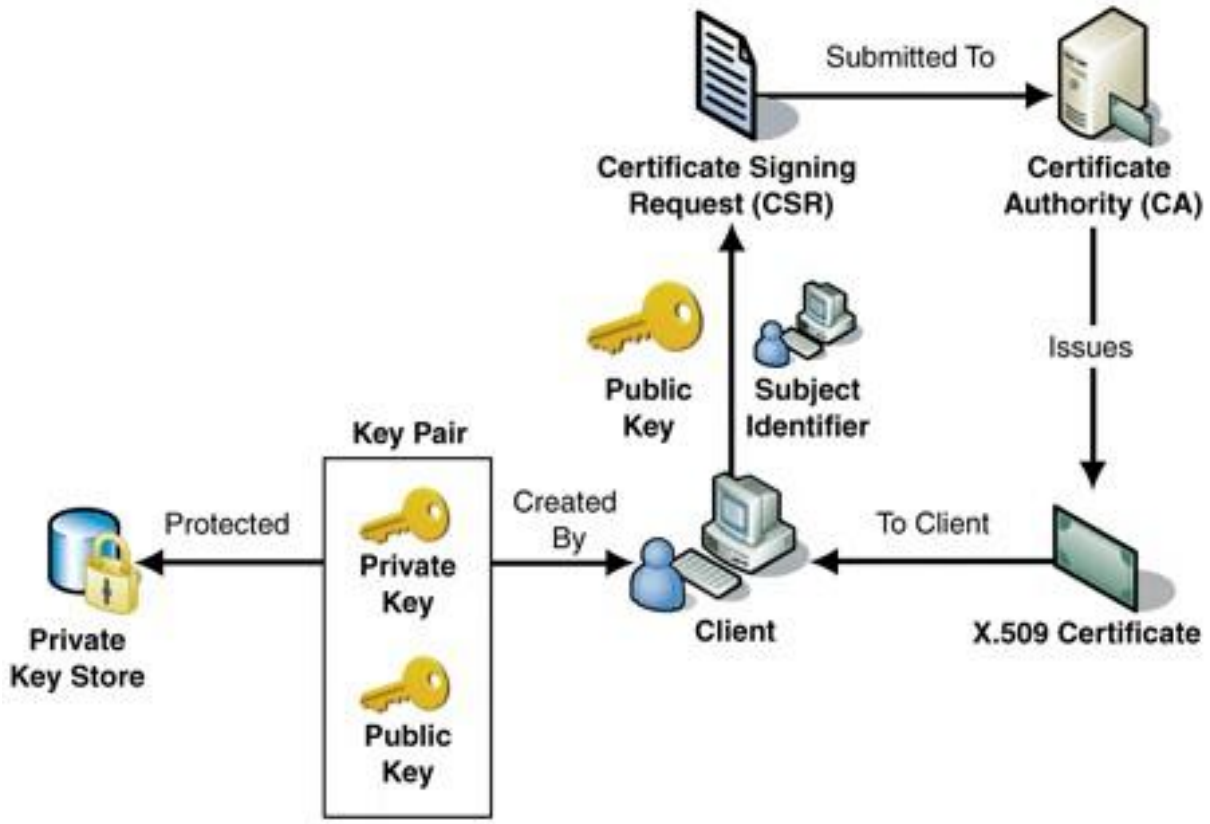
- Create a (CSR) for a new key:
  - Specify a Different Identity (CN, Organization...)
  - Use the -newkey option to generate a new key and the -keyout option to specify the key file's output location.
- Sign the CSR with the Self-Signed Certificate from Task 3
  - Use -CA and -CAkey options
- Verify the contents of the newly created certificate
  - Do you see different subject and issuer?

# Task 5: Building ~~Chain~~ Loop of Trust

- Create pairs
- Create CSR
- Send CSR to your partner
- Verify their CSR
- Issue their certificate
- Send the certificate back

For the certificate sharing, you can use the shared folder: https://drive.google.com/drive/folders/1-CQBWajT2JaG3SZtlaSjRaqH41Z_MrET?usp=drive_link

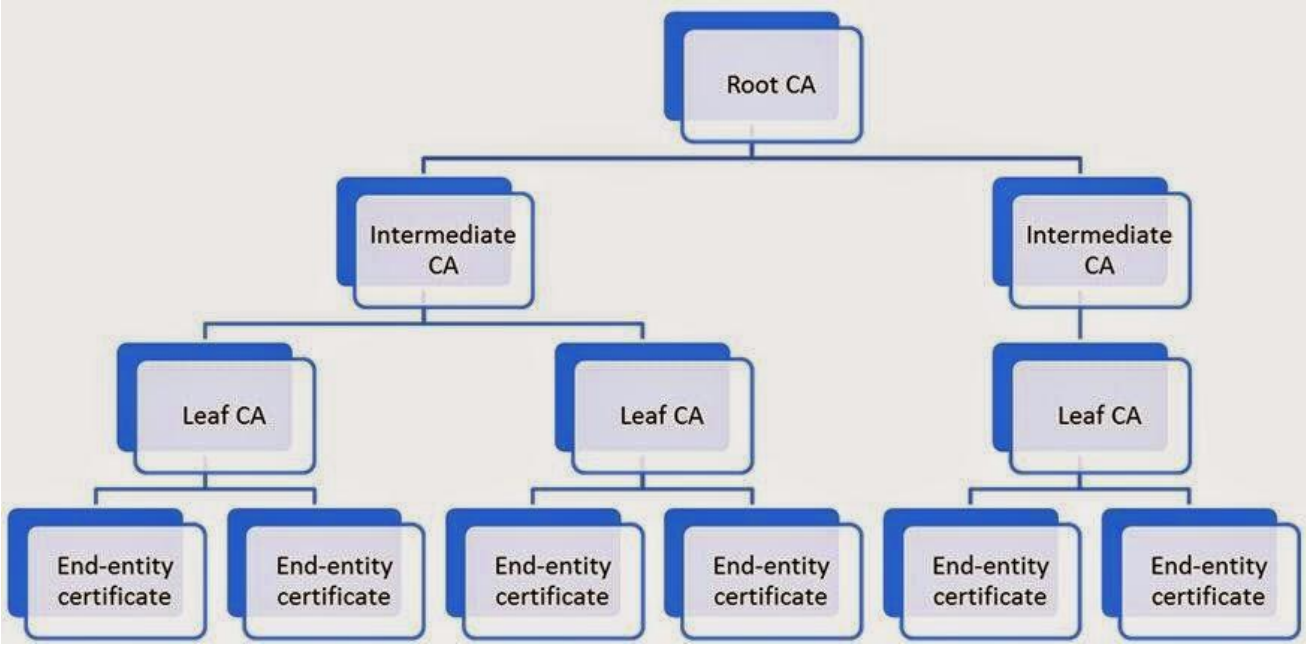
# Certificate issuing



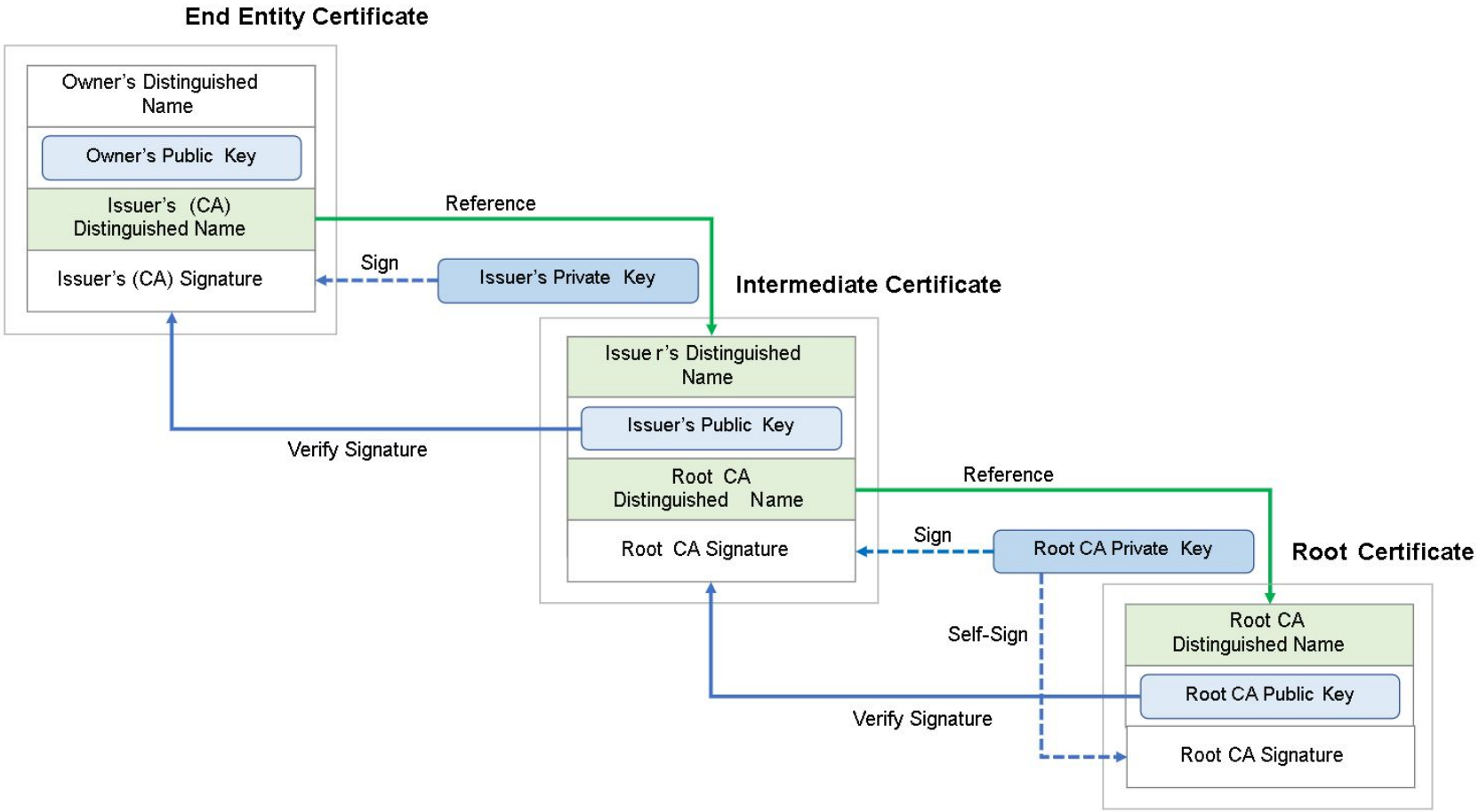

https://help.bizagi.com/process-modeler/en/index.html?cloud_auth_certificates.htm

# 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 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

# 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

# Introduction to Asymmetric Cryptography (Recall Slides)

## READ BEFORE THE SEMINAR

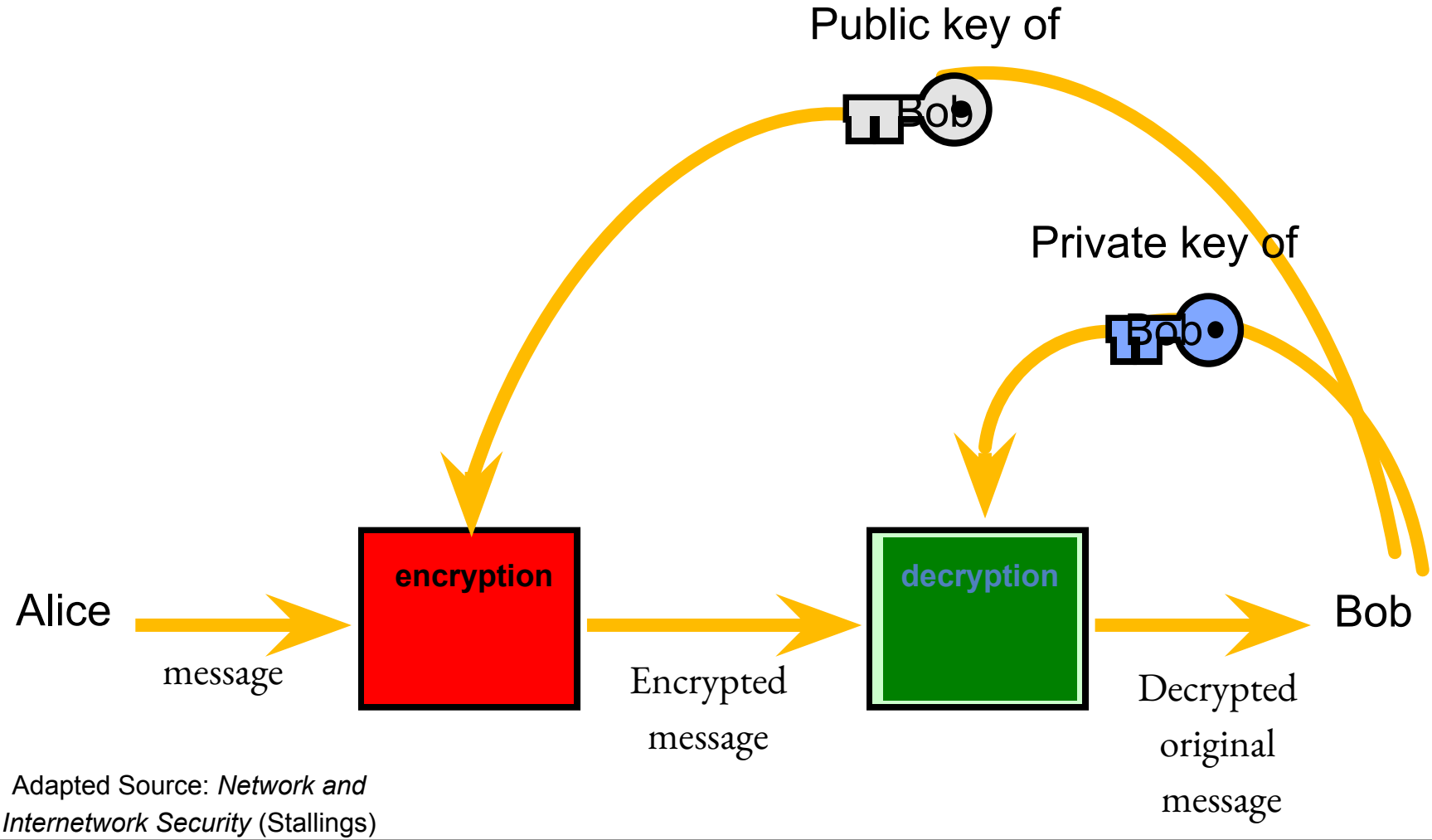
# Public vs private key cryptography

- 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)

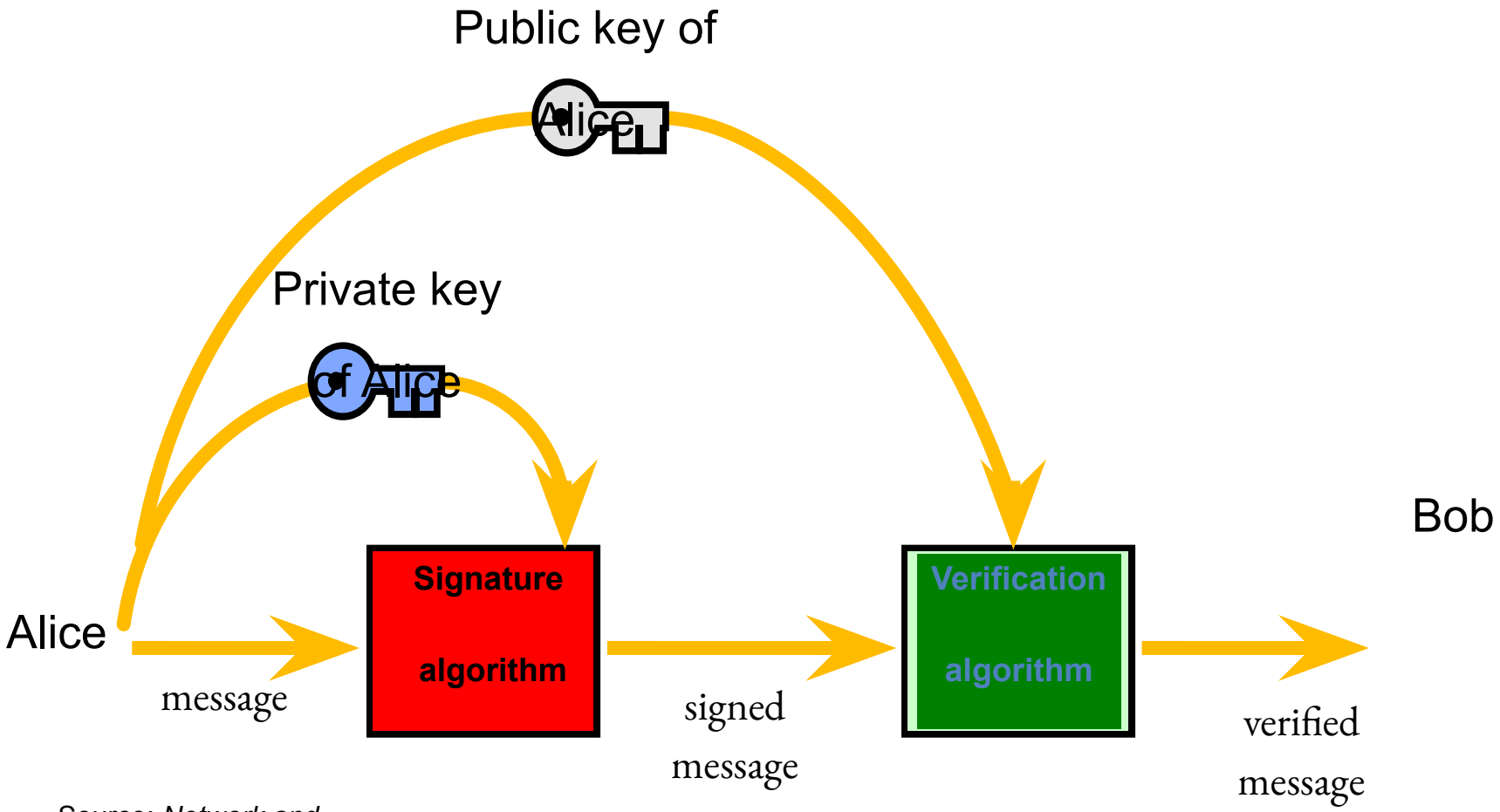
# Asymmetric cryptosystem

- 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 :
2. Public exponent :

3. Private exponent :

Encryption (public ):

Decryption (private ):

- RSA-1024: means  has 1024 bits and
  - Is 1024 bit secure?

# RSA: example

- Intentionally small numbers (**not** secure)
- We generate parameters:

- Public exponent is selected:
  - 3

- Private exponent is computed:

- The public key is:
- The private key is:
- Encryption/decryption:
  - Message
  - Encryption
  - Decryption 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 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($M$) || 3$x$ 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 $Z_p$, 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: $Z_p$ or **EC**

# Digital Signature Standard (DSS)

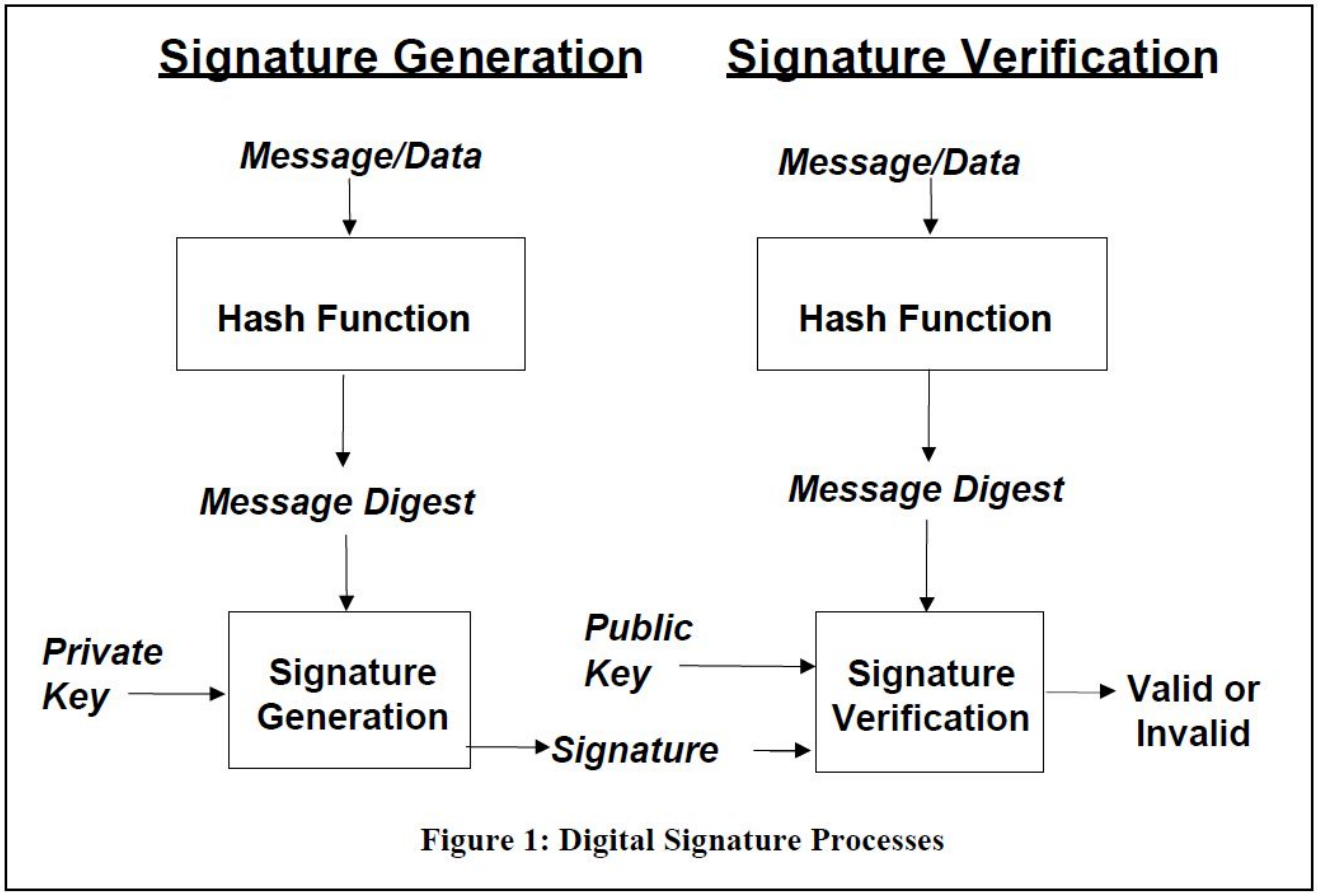



Figure 1: Digital Signature Processes

# 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 .
- 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.