2. Cryptography

2.3 Public-Key Cryptography
(Asymmetric Cryptography)
Outline

- Asymmetric cryptography
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    - Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange
  - Key-distribution with asymmetric cryptography

- Annex (complementary):
  - RSA and Math foundations
  - El Gammal Foundations
  - ECC Foundations
Outline

- Asymmetric cryptography
  - Public-Key cryptography principles
    - Public-Key algorithms
    - RSA algorithm
      - Key-Pair Generation and Encryption/Decryption
    - Diffie-Hellman key exchange
    - Key-distribution with asymmetric cryptography
Secret-Key Cryptography

- Traditional **shared secret key** cryptography *(symmetric methods)*
  - One key, shared by the principals
  - if the key is disclosed communications are compromised (principle of the key NDA between principals involved)

- Also **symmetric crypto**
  - Does not protect sender from receiver forging a message & claiming is sent by sender (non-repudiation concerns)
  - Peer-authentication “in strict sense” not warranted
  - Perfect secrecy (forward vs. backward) limited
  - *Must relay on a “trust” KDC, sharing keys with principals (non-disclosure principles) for key distribution protocols*
    - Limitation for PFS, PBS with independent key-generation control by principals to establish secure channels
    - Limitations: Key generation and key maintenance control
Public-Key Cryptography

- Probably most significant advance and revolution in the 3000 year history of cryptography
- Whitfield Diffie & Martin Hellman, Stanford University in 1976
  - But principles known earlier in classified community
- Clever application of number theory concepts and functions

- Complements **rather than replaces** secret key crypto or symmetric algorithms
  - Practice: hybrid use
    (symmetric + asymmetric + secure hashing)
Public-Key Cryptography

- **Public-key/two-key/asymmetric** cryptography involves **two** keys (or a key-pair):
  - a **public-key**, known by anybody: can be used to **encrypt** messages, and **verify** signatures
    - Confidentiality
  - a **private-key**, known only to the recipient: used to **decrypt** messages, and **sign** (create) **digital signatures**
    - Authentication

- **In asymmetric** methods:
  - Those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures
  - **Considering the key pair, what is encrypted with one key pair, is decrypted by the other key of that pair (for well-known algorithms)**
  - Encryption and Decryption functions implemented by the same computation.
The use of public-key cryptography

- **Confidentiality and Authentication**
  - Verification by each principal, based on correct and non-repudiable associations (principal ID, PublicKey)
  - Or (principal ID, Public Key) certified associations

- **Key exchange**: Two sides can cooperate to exchange a session key (or security association parameters): hybrid use of asymmetric and symmetric cryptography
  - Ex., Key generated by Alice and distributed to Bob:
    \[
    \{ K_s \}_K_{pubBob} \ || \ \{ H(M) \}_K_{privAlice} \ || \ \{ M \}_{Ks}
    \]

Some asymmetric algorithms are suitable for all uses (authentication, confidentiality) others are specific to one

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Encryption/Decryption</th>
<th>Digital Signature</th>
<th>Key Exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Elliptic Curve</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Diffie-Hellman</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DSS</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
**Encryption using a Public-Key System**

- **Confidentiality**

![Diagram](image)
Authentication using Public-Key System

- Authentication

![Diagram of authentication using public-key system]

1. Bob's private key
2. Bob's public key
3. Alice's public key ring
4. Transmitted ciphertext
5. Decryption algorithm (reverse of encryption algorithm)
6. Plaintext output
Public-Key Cryptosystems

Confidentiality + Authentication

Bob

Alice
Hybrid use

- Example (pgp)

Confidentiality + Authentication
Public-Key Method + Symmetric Encryption + Cryptographic hash

Note: Compression always before encryption!
Properties of Public-Key Cryptography (1)

1. Computationally feasible (easy) for a principal B to generate a pair:
   - public key: $K_{pub}$, $K_{pubB}$, $K_{ub}$
   - private key: $K_{priv}$, $K_{privB}$, $K_{Rb}$

2. Easy for sender to generate cipher text using a public-key

3. Easy for the receiver to decrypt ciphertext using private key

Possible notations:

$$C = \{M\}_K_{pub}$$
$$M = \{C\}_K_{priv}$$
$$M = D_{K_{Rb}}(C) = D_{K_{Rb}}[E_{K_{Ub}}(M)]$$
Properties of Public-Key Cryptography (2)

4. Computationally infeasible to determine private key ($K_{priv}$) knowing public key ($K_{pub}$)

   computationally infeasible to find decryption key knowing only algorithm & encryption key

4. Computationally infeasible to recover message $M$, knowing $K_{pub}$ and ciphertext $C$

5. Either of the two keys can be used for encryption, with the other used for decryption (depending on the algorithms):

$$M = \{ \{M\}K_{pub}\}K_{priv} = \{\{M\}K_{priv}\}K_{pub}$$

$$M = D_{K_{Rb}}[E_{K_{Ub}}(M)] = D_{K_{Ub}}[E_{K_{Rb}}(M)]$$
What means “easy” or “unfeasibility”

• Easy: something solved in polynomial time as a function of input length
  - Input: $n$ bits $\Rightarrow$ function proportional to $n^a$, with $a = \text{fixed constant}$
  - Functions of class $P$

• Unfeasibility: if the effort to compute grows faster than polynomial time
Outline

• Asymmetric cryptography
  - Public-Key cryptography principles
  - Public-Key algorithms
    - RSA algorithm
      • Key-Pair Generation and Encryption/Decryption
    - Diffie-Hellman key exchange
    - Key-distribution with asymmetric cryptography
  - Annex (complementary): RSA and Math foundations
Math of public-key encryption

- Number theory
  - Prime numbers, prime factorization
  - Relatively primes and its properties:
    - Ex., GCD
  - Fermat theorem
  - Euler theorem and Euler Totient Function $\phi(n)$
  - Primality testing
    - Miller-Rabin algorithm and prime distribution or estimation
  - CRT (Chinese Reminder Theorem)
  - Modular arithmetic
  - Primitive roots of integers and primes
  - Discrete logarithms (inverse of exponentiation)
    - Find $i$, such that $b = a^i \pmod{p}$, or $i = d\log_a b \pmod{p}$
Public-Key algorithms (1)

RSA: Rivest, Shamir & Adleman, MIT, 1977

- Best known & widely used and implemented public-key scheme
  - Used as a block cipher or digital signatures
  - Digital signatures combining secure hash functions and
    standardized computations: ex., PKCS#N standards
  - Hybrid use with symmetric crypto: digitally signed and
    confidential symmetric key-envelopes, combined with
    symmetric encryption
- Based on exponentiation in a finite (Galois) field over integers
  modulo a prime
  - Feasible to compute $Y = X^K \mod N$ (knowing $K$, $X$ and $N$)
  - Impossible (computationally) to compute $X$ from $Y$, $N$ and $K$
  - nb. exponentiation takes $O((\log n)^3)$ operations (feasible)
- Uses large integers (eg. 1024, 2048, 4096 bits)
- Security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)
Public-Key algorithms (2)

• **Diffie-Hellman**
  - Exchange a secret key securely (secret key establishment) or key-agreement
  - Unfeasible solution of discrete logarithms (computational time and complexity)

• **El Gamal**
  - Block Cipher
  - Unfeasible solution of discrete logarithms (computational time and complexity)

• **Digital Signature Standard (DSS)**
  - Makes use of the SHA-1
  - For digital signatures (only), not for encryption or key exchange
  - Also implementable with different asymmetric algorithms
Public-Key algorithms (3)

Other public-key algorithms:
• Knapsack, Pohlig-Hellman, Rabin, McEliece, LUC, Finite Automaton

Public-Key signature algorithms:
• DSA variants, GOST, Discrete Logarithm Variants,
• Ong-Schnorr-Shamir, ESIGN

Foundation explained in:
Bruce Schneier, Applied Cryptography, Wiley, 2006
Elliptic Curve Cryptography

- Elliptic-Curve Cryptography (ECC)
  - Good for smaller bit size
  - Low confidence level yet, compared with RSA
  - Interesting properties, reputation growing
  - Very complex
- Majority of public-key crypto (RSA, D-H) use either integer or polynomial arithmetic with very large numbers/polynomials
- Imposes a significant load in storing and processing keys and messages
- An alternative is to use elliptic curves
- Offers same security with smaller bit sizes
- Newer, but not as well (crypt)analyzed
- Standardization problem: different ECC curves and characteristics
## Comparable Key Sizes for Equivalent Security

Computational effort for cryptanalysis

<table>
<thead>
<tr>
<th>Symmetric scheme (key size in bits)</th>
<th>ECC-based scheme (size of $n$ in bits)</th>
<th>RSA, DSA (modulus size in bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>112</td>
<td>512</td>
</tr>
<tr>
<td>80</td>
<td>160</td>
<td>1024</td>
</tr>
<tr>
<td>112</td>
<td>224</td>
<td>2048</td>
</tr>
<tr>
<td>128</td>
<td>256</td>
<td>3072</td>
</tr>
<tr>
<td>192</td>
<td>384</td>
<td>7680</td>
</tr>
<tr>
<td>256</td>
<td>512</td>
<td>15360</td>
</tr>
</tbody>
</table>
Outline

• Asymmetric cryptography
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    • Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange
  - Key-distribution with asymmetric cryptography

- Annex (complementary): RSA and Math foundations
The RSA Algorithm - Key Generation

Key pair generation (summary and simple example)

1. Select $p, q$ - $p$ and $q$ both prime (secrets)
2. Calculate $n = p \times q$
3. Calculate $\Phi(n) = (p - 1)(q - 1)$
4. Select integer $e$ - $\gcd(\Phi(n), e) = 1; 1 < e < \Phi(n)$
5. Calculate $d$ - $d = e^{-1} \mod \Phi(n)$
6. Public Key - $K_{pub} = \{e, n\}$
7. Private key - $K_{priv} = \{d, n\}$

1) Ex., 7, 17  
2) $n = 7 \times 17 = 119$  
3) $\varphi(n) = 6 \times 16 = 96$  
4) $e = 5, \gcd(96, 5) = 1, \text{ com } 1 < 5 < 96$  
5) $5xd = 1 \mod 96, \text{ com } d < 96 \quad d = 77$  
   $5 \times 77 = 385, \text{ notar que } 4 \times 96 + 1 = 385$

$K_{pub} = (5, 119)$  
$K_{priv} = (77, 119)$
The RSA Algorithm: Encryption/Decryption

**Encryption:** \( C = \{P\}_{K_{pub}} \)

- Plaintext: \( M < n \)
- Ciphertext: \( C = M^e \pmod{n} \)

**Decryption:** \( P = \{C\}_{K_{priv}} \)

- Ciphertext: \( C \)
- Plaintext: \( M = C^d \pmod{n} \)

**Example:**

- Ciphertext: \( C = 66 \)
- Plaintext: \( M = 19 \)

Encryption:

- Plaintext: 19
- Ciphertext: 66

Decryption:

- Ciphertext: 66
- Plaintext: 19

\( K_{pub} = (5, 119) \)

\( K_{priv} = (77, 119) \)
Another RSA Example - Key Setup

1. **Select primes:** $p=17 \& q=11$ (secrets)
2. **Compute** $n = pq = 17 \times 11 = 187$
3. **Compute** $\varphi(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. **Select** $e$: $\gcd(e, 160) = 1$; **choose** $e = 7$
5. **Determine** $d$: $de = 1 \mod 160$
   and $d < 160$ **Value is** $d = 23$ since $23 \times 7 = 161 = 10 \times 160 + 1$

1. **Publish public key** $K_{pub} = \{7, 187\}$
2. **Keep secret private key** $K_{priv} = \{23, 187\}$
Another RSA Example - Encrypt/Decrypt

• **given message** \( M = 88 \) (nb. \( 88 < 187 \))

• **encryption:**
  
  \[ C = 88^7 \mod 187 = 11 \]

• **decryption:**
  
  \[ M = 11^{23} \mod 187 = 88 \]
RSA (complementary material)

- Practice (Optional Work-Assignment / Verification)
  See Optional WA-2


  Later (practical class):
  Programming Examples

- RSA Security foundations and math:
  later, in the annex of these slides (annex)
Outline

• **Asymmetric cryptography**
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    • Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange
  - Key-distribution with asymmetric cryptography

- Annex (complementary): RSA and Math foundations
Diffie-Hellman Key Exchange

• First public-key type scheme:
  - Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that Williamson (UK CESG) secretly proposed the concept in 1970

• Practical method for public exchange of a secret key \( k \) between A and B
  - Never exposing \( k \)
  - Without any previous shared secret between A and B
  - Use for shared secret key-establishment without any previous shared secret
  - Good for PFS and PBS warranties

• Used in many security standard protocols and in a number of commercial products
Diffie-Hellman Key Exchange

• D-H is a public-key scheme for use as a key (or secret) distribution scheme
  - cannot be used to exchange an arbitrary message (not an encryption method)
  - rather it can establish a common key, known only to the two participants
  - The common key can be used as a shared secret key or shared key-material/seed to generate a key, ...

• Value of key depends on and only the participants (and their private and public information)
  - D-H Private and public numbers

• Based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy to compute

• Security relies on the difficulty of computing discrete logarithms (similar to factoring) - hard to compute (or computationally impossible)
Diffie-Hellman: foundations (1)

- **Global parameters:**
  - \( q \): a large prime integer or polynomial
  - \( a \): a primitive root \( \mod q \)

- **In modular arithmetic, a primitive root \( \mod q \) is any number \( a \) that:**
  - Any number \( b \) relatively prime to \( q \) is congruent to a power \( a^i \mod q \)
  - \( a \) is called the generator of a multiplicative group of integers modulo \( q \)
  - \( a^i \mod p \), where \( 0 \leq i \leq (p-1) \) generates all the integers between 1 and \( q-1 \), in some permutation order
  - For any integer \( b < q \) with, there is a unique exponent integer \( i \) such that \( b = a^i \mod q \)

- Such \( i \) is called the index or the discrete logarithm of \( b \) for the base \( a \) \( \mod q \)

\[ i = \text{dlog}_{a,q}(b) \]
Diffie-Hellman: foundations (2)

- Considering \( i \) the discrete logarithm for which:
  \[ a^i \mod q = b, \text{ taking } a \text{ and } q \text{ (as initial known parameters)} \]
  - It is simple to calculate \( b \), knowing \( i \)
  - But is very hard to calculate \( i \) only knowing \( b, a \) and \( q \)
  - This implies the computation of the discrete logarithm:
    no efficient solution (computational impossibility)
    - Hard as polynomial complexity
    - Linear to \( a \), computational complexity equivalent to \( a^i \)

- We can verify that from modular arithmetic properties:
  \[ a^R \mod q = a^{R_1 \cdot R_2} \mod q \]
  \[ = (a^{R_1} \mod q) \cdot (a^{R_2} \mod q) = (a^{R_1} \mod q)^{R_2} \mod q \]
Diffie-Hellman Setup and agreement

• If A and B share the global parameters $a$ and $q$, being $a$ a primitive root modulo $q$
• Each user (eg. A, B) generates (private,public) pair
  - selects a random **private secret number**: $x < q$
  - Computes: $y = a^x \mod q$ and makes public $y$ as a **public number**
Diffie-Hellman Key Exchange

• Shared session key for users A & B is $K_{AB}$:
  $$K_{AB} = a^{x_A.x_B} \mod q$$
  $$= y_A^{x_B} \mod q \text{ (which B can compute)}$$
  $$= y_B^{x_A} \mod q \text{ (which A can compute)}$$

• $K_{AB}$ is used as session key in secret-key sharing encryption scheme between Alice and Bob

• If Alice and Bob subsequently communicate, they will have the same key as before, unless they choose new public-numbers for new D-H agreement
  - Successive D-H agreements for rekeying of $K_{AB}$
  - PFS and PBS conditions warranted

• Note: It is possible to make generalized D-H agreements, extended to a group of N
Diffie-Hellman Example

- Users Alice & Bob who wish to swap keys:
  - Ex., agree on prime $q = 353$ and $a = 3$

- Select random secret numbers:
  - $A$ chooses $x_A = 97$, $B$ chooses $x_B = 233$

- Compute respective the public numbers:
  - $Y_A = 3^{97} \mod 353 = 40$ (Alice)
  - $Y_B = 3^{233} \mod 353 = 248$ (Bob)

- Compute shared session key as:
  - $K_{AB} = Y_B^{x_A} \mod 353 = 248^{97} = 160$ (Alice)
  - $K_{AB} = Y_A^{x_B} \mod 353 = 40^{233} = 160$ (Bob)

- PFS and PBS, without knowing the private numbers (never exposed) and without any previous shared secret or long-time duration secrets
Diffie-Hellman Key Exchange (example)

\[ \text{q} = 353, \ a = 3 \]

**Alice**

\[ Y_A = 3^{97} \mod 353 = 40 \]
\[ Y_B = 3^{233} \mod 353 = 248 \]

**Bob**

\[ Y_B = 3^{233} \mod 353 = 248 \]
\[ Y_A = 3^{97} \mod 353 = 40 \]

**Mallory**

\[ Y_A \cdot x_A \mod 353 = 248 \cdot 97 \mod 353 = 160 \]
\[ Y_B \cdot x_B \mod 353 = 40 \cdot 233 \mod 353 = 160 \]

\[ K_{ab} = 160 \]
\[ K_{ba} = 160 \]

Mallory can obtain \( K_{ab} \)?

Mallory can attack as a “man in the middle”?
**D-H with a MIM Attack**

**User A**
- Generate random $X_A < q$;
- Calculate $Y_A = \alpha^{X_A} \pmod{q}$
- Calculate $K = (Y_B)^{X_A} \pmod{q}$
- $K_A = Y_{D_2}^{X_A} \pmod{q}$

**User B**
- Generate random $X_B < q$;
- Calculate $Y_B = \alpha^{X_B} \pmod{q}$
- Calculate $K = (Y_A)^{X_B} \pmod{q}$
- $K_B = Y_{D_1}^{X_B} \pmod{q}$

**MIM**
- $X_{D_1} = a^{X_{D_1}} \pmod{q}$
- $Y_{D_1} = a^{X_{D_1}} \pmod{q}$
- $X_{D_2} = a^{X_{D_2}} \pmod{q}$
- $Y_{D_2} = a^{X_{D_2}} \pmod{q}$

- $K_B = Y_B^{X_{D_1}} \pmod{q}$
- $K_A = Y_A^{X_{D_2}} \pmod{q}$

**Encryption and Decryption**
- $C = \{ M \} K_A$
- $M = \{ C \} K_A$
- $C = \{ M \} K_B$
- $C = \{ M \} K_B$
Key Exchange Protocols and the Authentication Problem

- Users could create random private/public D-H keys each time they communicate
- Users could create a known private/public D-H key and publish in a directory, then consulted and used to securely communicate with them
  - Ephemeral D-H Agreement (EDH)
  - Fixed D-H Agreement (FDH)
- Both of these are vulnerable to a possible Meet-in-the-Middle (MIM) Attack
  - Why?
    - Anonymous D-H agreement (ADH)
- Authentication of the exchanged values is needed
  - So, you will need Authenticated D-H agreements
  - How?
Outline

• Asymmetric cryptography
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    • Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange
  - Key-distribution with asymmetric cryptography

- Annex (complementary): RSA and Math foundations
Public-Key Distribution of Secret Keys

• As we know, we can use public-key methods for secrecy (confidentiality) or authentication
• But public-key algorithms are slow
• So usually want to use secret-key and symmetric encryption to protect message contents (confidentiality)
  - Hence need a session key
  - Have several alternatives for negotiating a suitable session key, using public-key methods
Simple Secret Key Distribution (1)

- Merkle, 1979
  - A generates a new temporary public key pair
  - A sends B the public key and their identity
  - B generates a session key K sends it to A encrypted using the supplied public key (public-key based session key envelope)
  - A decrypts the session key and both use

- Problems: the opponent can intercept and impersonate both halves of protocol. Solution?
Simple Secret Key Distribution (2)

- Problems: the opponent can intercept and impersonate both halves of protocol. Solutions?

1. \{ N1, IDA \}KpubB
2. \{ N1, N2 \}KpubA
3. \{ N2 \}KpubB
4. \{ N1 || Ks \}KpubA

Public Key Methods
- Symmetric Methods
Authentication with Public-Key Cryptography

- Fast mutual authentication and key-session distribution using a public-key cryptosystem.

- Hybrid approach:
  - Authentication using asymmetric cryptosystems
  - Confidentiality using symmetric cryptosystems

- Problems: impersonification and contributive key-generation or key-agreement mechanisms.

Solutions?

- Public Key Digital signatures for authentication proofs
- Public key protected envelopes for symmetric key distribution (or for the exchange of authenticated and confidential contributive master-secrets, key-material or key-seeds, used for independent key-generation)
- Symmetric message encryption with the generated session key, and cryptographic MACs with generated MAC-keys
Key Management with Public Key Methods

• Key Distribution
  - If we trust on Public Keys (authenticated), it is easy to generate and distribute session keys in “envelopes”, with authentication, confidentiality and integrity guarantees
    • No need of KDCs
    • Only need to obtain and manage public keys in a trust way
      - Authenticated or Certified Public Keys
      - Envelopes signed in the session key-distribution context
    • Good for PFS, PBS (minimization of TCB)

• Key Management
  - Trust model for (Pub.Key, Principal Id) pairs
    • Public Key trust distribution model
  - Principals must kept private keys in a secure way
Key distribution with Public Key Methods

- Key Distribution Schemes (example schemes)

Ex.1: A to B:
{ Ks, ... { H( Ks, ...) }KprivA } KpubB

Ks:
Shared Session Key (symmetric key)

Ex.2: A to B:
{ PMsA, ... { H( PMsA, ...) }KprivA } KpubB
B to A:
{ PMsB, ... { H( PMsB, ...) }KprivB } KpubA

Ks=
F_g(PMsA, PMsB)

Ex.2: A to B:
{ DHpA, a, q ... ||{H(DHpA, a, q, ...)}KprivA} KpubB
B to A:
{ DHpB, a, q ... ||{H(DHpB, a, q, ...)}KprivA} KpubB

Ks=
DH(DHpA, DHpB)
Key Management

• Public-key encryption helps address key distribution problems

• Have two aspects:
  - Trust distribution model of public keys
    • can be considered as using one of:
      - public announcement (ex., as a web of trust)
      - publicly available directory (well-known trust service)
      - public-key authority (ex., authority services)
      - public-key certificates and Certification Authorities (CAs)
        » Trust well known and authority services
  - use of public-key encryption to distribute secret keys
    • Use with symmetric methods
Public Announcement

• Users distribute public keys to recipients or broadcast to community at large
  - Web-of-Trust Distributed Model
  - eg. append PGP keys to email messages or post to newsgroups or email list

• Major weakness is forgery
  - Anyone can create a key claiming to be someone else and broadcast it
  - Until forgery is discovered can masquerade as claimed user
  - Require some kind of “trust or reputation-based” evaluation scheme to manage disseminated public-keys
    • Trust metrics and reputation models can be implemented by autonomous trust management or in a collaborative distributed trust or reputation models
    • Ex., PGP management of Public Key Rings and Private Key Rings (autonomously computed trust metrics)
Publicly Available Directory

- Can obtain greater security by registering keys with a public directory, used as a TCB service

- Directory must be trusted (TCB) with properties:
  - contains \( \{ \text{name, public-key} \} \) entries
  - or ... \( \{ \text{name, public-key, TTL validation, policy validation,} \} \) entries

- Assumptions:
  - participants register securely with directory
  - participants can replace key at any time
  - directory is periodically published (timeliness conditions)
  - directory can be accessed electronically

- Still vulnerable to tampering or forgery
  - How?
Public-Key Authority

- Improve security by tightening control over distribution of keys from a certified directory
  - Public Keys in Registration entries signed by the Directory (available via Public Key Centers or PKCs)
  - Registration entries for Principal IDs and Public Keys (signed) by the Directory Service Provider

- Has similar properties of directory

- Requires users to know (and trust) public key for the directory

- Then users interact with directory to obtain any desired public key securely
  - Require real-time access to directory when keys are needed
  - How to address asynchronous use?
  - What about possible public-key revocation?
Authentication and integrity issues as well, non-replaying protection, Time-synchronization, public key time-to-live control, availability, performance or DoS are some other complementary issues that must be also addressed in complete and specific protocols.
Public-Key Certificates

- Certificates allow key exchange without real-time access to public-key authority
- A certificate binds **identity** to **public key** for a well-known validity time for the association
- Time is all we need?
  - usually we need other info such as, issuer identification, issuing delegation control, issuing verification conditions, rights (or certification policies) as attributes for the correct certified use, etc
- With all contents **signed** by a trusted Public-Key or Certificate Authority (CA)
- Can be verified by anyone who knows the public-key authorities public-key
  - Verification must be complete according with all the issuing terms of use
Principle of Public-Key Certificate Use

Cert = [A, KpubA, TTL, Policy attributes, CA-Issuer, Issuing Date, …]

Authenticated Cert (CA signed)

{ H(Cert) } KprivCA
Public-Key Certificates

Certificate (Pub Key) Sign Request
(CA enrolment and registration)

Certificates
Issued and
signed by
the CA

Use of
certificates

Certificate Authority

A, KpubA, …

CA= { A, KpubA, Time Verif., …}KprivCA

B, KpubB, …

CB{ A, KpubA, Time Verif, …}KprivCA

CA ||…. 

CB ||…. 
Readings

- William Stallings, Network Security Essentials, Chap.2 (summarized vision)


Outline (annex)

• Asymmetric cryptography
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    • Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange
  - Key-distribution with asymmetric cryptography

- Annex (complementary): RSA and Math foundations
Security of Public Key Schemes

- like secret key schemes brute force exhaustive search attack is always theoretically possible
- but keys used are now too large (~1024 bits)

- security relies on a large enough difference in difficulty between
  - easy (encrypt/decrypt) from known keys
  - hard (cryptanalysis) problems, not knowing the keys

- more generally the hard problem is known, but is made hard enough to be impractical to break
- requires the use of very large numbers
- hence is slow compared to secret key schemes and symmetric methods
RSA Use

- to encrypt a message $M$ the sender:
  - obtains public key of recipient $PU = \{e, n\}$
  - computes: $C = M^e \mod n$, where $0 \leq M < n$

- to decrypt the ciphertext $C$ the owner:
  - uses their private key $PR = \{d, n\}$
  - computes: $M = C^d \mod n$

- note that the message $M$ must be smaller than the modulus $n$ (block if needed)
Why RSA Works

- because of Euler's Theorem:
  - \(a^{\phi(n)} \mod n = 1\) where \(\gcd(a, n) = 1\)

- in RSA have:
  - \(n = p \cdot q\)
  - \(\phi(n) = (p-1)(q-1)\)
  - carefully chose \(e\) & \(d\) to be inverses \(\mod \phi(n)\)
  - hence \(e \cdot d = 1 + k \cdot \phi(n)\) for some \(k\)

- hence:
  \[
  C^d = M^{e \cdot d} = M^{1 + k \cdot \phi(n)} = M^1 \cdot (M^{\phi(n)})^k
  = M^1 \cdot (1)^k = M^1 = M \mod n
  \]
Operations with big integers (large numbers)

- **Optimization strategy**
  - Elementary addition and subtraction
    - Multiple-precision addition or subtraction of large numbers are $O(n)$
      - $n$ the number of bits of operands
  - Modular addition and subtraction is $O(n)$
    - $(x + y) \mod N$
      - $= x + y$, if $x + y < N$
      - $= x + y - m$, if $x + y \geq N$

- Large number multiplication
  - Pencil and paper method algorithm: $O(n^2)$
  - Russian Peasant Multiplication Method
    - Good for binary representations
    - Multiplication as a series of additions and shifts
    - Variable complexity: from $O(n)$ to $O(n^2)$
Modular multiplication

- Requires the computation of $x \cdot y \mod N$
  - First must compute $x \cdot y$
  - Followed by a reduction of the result modulo $M$
    - Relates with a division and computation of large dimension intermediate products

- Possible optimized reductions:
  - Barret modular reduction

- Montgomery’s multiplication
- Multiplication by squaring
Multiplication by squaring

\[ a.b \mod n = \left\{ \left[ (a+b)^2 - a^2 - b^2 \right] / 2 \right\} \mod n \]

\[ a.b \mod n = \left\{ \left[ (a+b)^2 - (a - b)^2 \right] / 4 \right\} \mod n \]

Advantage:

*Can also benefit from fast calculations performed on a cryptographic coprocessor*
Exponentiation can be simple

- Can use the Square and Multiply Algorithm
- A fast, efficient algorithm for exponentiation
- Conceptually is based on repeatedly squaring base and multiplying in the ones that are needed to compute the result
- Look at binary representation of exponent
- Only takes $O(\log_2 n)$ multiples for number $n$
  - eg. $7^5 = 7^4.7^1 = 3.7 = 10 \mod 11$
  - eg. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \mod 11$
Exponentiation algorithm

c = 0; f = 1
for i = k downto 0
  do c = 2 x c
      f = (f x f) mod n
  if b_i == 1 then
    c = c + 1
    f = (f x a) mod n
return f
Efficient Encryption

- Encryption uses exponentiation to power e
- Hence if e small, this will be faster
  - often choose e=65537 (2^{16}-1)
  - also see choices of e=3 or e=17
- But if e too small (eg e=3) can attack
  - using CRT (Chinese Remainder Theorem) & 3 messages with different modulii
- If e fixed must ensure \( \gcd(e, \phi(n)) = 1 \)
  - ie reject any p or q not relatively prime to e
  - Need to generate those two primes with these property
    - First generate primes (randomly + primality test)
    - Check if \( \gcd(e, \phi(n)) = 1 \).
    - If yes, p and q are ok, if not, generate other pair (p,q)
Efficient Decryption

• Decryption uses exponentiation to power d
  - this is likely large, insecure if not

• Can use the Chinese Remainder Theorem (CRT) to compute mod p & q separately. then combine to get desired answer
  - approx 4 times faster than doing directly

• The key idea: only owner of private key who knows values of p & q can use this technique
CRT - Chinese Reminder Theorem (1)

- Can decrease the processing time involving private keys by a factor of ~4.

- If the integers \( n_1, n_2, n_3, \ldots, n_k \) are pairwise relatively prime, then the system of simultaneous congruences

\[
\begin{align*}
x &\equiv a_1 \mod n_1 \\
x &\equiv a_2 \mod n_2 \\
\vdots \\
x &\equiv a_k \mod n_k
\end{align*}
\]

Tem uma única solução \( x \), tal que \( 0 \leq x \leq n \)

Com \( n = n_1 n_2 n_3 \ldots n_k \)
CRT – Chinese Reminder Theorem (2)

A solução para \( x \) pode ser calculada na forma:

\[
x \equiv \sum_{i=1}^{k} a_i N_i \cdot N'_i
\]

\[
n = n_1 \cdot n_2 \cdot n_3 \ldots \cdot n_k
\]

\[
N_i = \frac{n}{n_i} \quad N'_i = N_i^{-1} \pmod{n_i}
\]

\[
i = 1, 2, \ldots, k
\]
**CRT (practical example - 1)**

\[ M = c^d \mod n \]

\( p = 7, \; q = 11, \; e = 19, \; d = e^{-1} \mod (p-1)(q-1) = 19 \)

Precomputing:

\( d_P = e^{-1} \mod (p-1) = d \mod (p-1) = 19 \mod 6 = 1 \)

\( d_Q = e^{-1} \mod (q-1) = d \mod (q-1) = 19 \mod 10 = 9 \)

\( q_{\text{Inv}} = q^{-1} \mod p = 11^{-1} \mod 7 = 2 \)

Then, storing the quintuple \((p, q, d_P, d_Q, q_{\text{Inv}})\) (as a representation of the private key)

---

*Note* we can also try \( n = p \times q \times r \times t \)
CRT (practical example - 2)

Then, to compute

\[ s = m^d \mod n = m^d \mod (p.q) \]

We can compute (Garner’s Algorithm)

\[ s_1 = m^{d_p} \mod p = 50^1 \mod 7 = 1 \]
\[ s_2 = m^{d_Q} \mod q = 50^9 \mod 11 = 2 \]

\[ h = q \text{Inv} \ (s_1 - s_2) \mod p = 2(1-2) \mod 7 = 5 \]

\[ s = s_2 + hq = 2 + 5(11) = 57 \]

Similar to:

\[ s = m^d \mod pq = 50^{19} \mod 77 = 57 \]
RSA Key Generation

- Users of RSA must:
  - determine two primes at random - $p$, $q$
  - Primality test
  - select either $e$ or $d$ and compute the other
- Primes $p, q$ must not be easily derived from modulus $n = p \cdot q$
  - means must be sufficiently large
  - Difficult for factorization
  - typically guess and use probabilistic test
    - (ex., Probabilistic Rabin-Miller)
- exponents $e, d$ are inverses, so use
  Inverse algorithm to compute the other
  - Euclid’s Inverse Algorithm
RSA Security

• Possible approaches to attacking RSA are:
  - Brute force key search (infeasible given size of numbers)
  - Mathematical attacks (based on difficulty of computing $\varphi(n)$, by factoring modulus $n$)
  - Timing attacks (on running of decryption)
  - Chosen ciphertext attacks (given properties of RSA)
Factoring Problem

- mathematical approach takes 3 forms:
  - factor $n = p \cdot q$, hence compute $\varphi(n)$ and then $d$
  - determine $\varphi(n)$ directly and compute $d$
  - find $d$ directly

- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of May-05 best is 200 decimal digits (663 bit) with LS (LATTICE SIEVE)
  - biggest improvement comes from improved algorithm
    - cf QS to GNFS to LS
  - currently assume 1024-2048 bit RSA is secure
    - ensure $p, q$ of similar size and matching other constraints
    - But observations and studies evolve, considering also that computers will continue to get faster
Timing Attacks

• Developed by Paul Kocher in mid-1990’s
  - Applicable to any public-key crypto system
  - Ciphertext only attack

• Exploit timing variations in operations
  - eg. multiplying by small vs large number
  - or IF’s varying which instructions executed

• Infer operand size based on “time taken”

• RSA exploits time taken in exponentiation

• Countermeasures
  - Use constant exponentiation time
  - Add random delays
  - Blind values used in calculations
  - Secure message padding can also help
Chosen Ciphertext Attacks

RSA is vulnerable to a Chosen Ciphertext Attack (CCA)

Attackers chooses ciphertexts & gets decrypted plaintext back

Choose ciphertext to exploit properties of RSA to provide info to help cryptanalysis

Can counter with random pad of plaintext or use Optimal Asymmetric Encryption Padding (OASP)