Lecture 2
Applied Cryptography (Part 2)

Patrick P. C. Lee
Roadmap

- Number theory
- Public key cryptography
  - RSA
  - Diffie-Hellman
  - DSA
  - Certificates
Number Theory

- Number theory deals with the arithmetic operations of cryptographic algorithms.
- Not only to understand how an algorithm works, but also why it works.
- Most public key algorithms are based on modular arithmetic.
If $m$ and $n$ are two integers and $n > 0$, the **remainder** of $m$ divided by $n$ is the smallest non-negative integer that differs from $m$ by a multiple of $n$.

- For $3$, $13$ and $-7$ divided by $10$, remainder $= 3$

In modular arithmetic, two integers are **equivalent** if they differ by a multiple of $n$. 
Modular Arithmetic

- General rules:
  - \( a + b \mod n = [(a \mod n) + (b \mod n)] \mod n \)
  - \( a - b \mod n = [(a \mod n) - (b \mod n)] \mod n \)
  - \( a \times b \mod n = [(a \mod b) \times (b \mod n)] \mod n \)

- E.g., \( 12 \times 25 \mod 7 = (12 \mod 7) \times (25 \mod 7) \mod 7 = 5 \times 4 \mod 7 = 6. \)

- Note that
  - \( a^b \mod n = (a \mod n)^b \mod n \)
  - But \( a^b \mod n \neq a^{b \mod n} \mod n \)

- We sometimes drop “mod n” for brevity
A positive integer $p$ is prime iff it is evenly divisible by exactly two positive integers (itself and 1)

Question: is the set of primes infinite?

Many crypto algorithms work on very large primes, which can be found efficiently using randomized algorithms
Greatest Common Divisor (GCD)

- The GCD of two integers is the largest integer that evenly divides them
  - \( \text{gcd}(0, x) = x \)
- Two integers are relatively prime if their GCD = 1.
- For two integers \( a \) and \( b \), \( a \) is the multiplicative inverse mod \( n \) of \( b \) (and vice versa) if \( a \times b \mod n = 1 \) (or \( a^{-1} = b \mod n \))
- Both GCD and multiplicative inverse can be found efficiently via Euclid’s algorithm
GCD vs. Multiplicative Inverse

Facts:

• $a^{-1} = x \pmod{n}$ has a unique solution $x$ if $a$ and $n$ are relatively prime

• $a^{-1} \pmod{n}$ doesn’t exist if $a$ and $n$ are not relatively prime

• If $n$ is prime, all integers 1, 2, …, n-1 are relatively prime to $n$
Chinese Remainder Theorem

Let $n_1$, $n_2$, ..., $n_k$ be positive integers that are pairwise relatively prime. Then for any given integers $a_1$, $a_2$, ..., $a_k$, there exists an integer $x$, where $0 < x < n_1n_2...n_k$ such that $x = a_1 \pmod{n_1}$, $x = a_2 \pmod{n_2}$, ..., $x = a_k \pmod{n_k}$. 
$\mathbb{Z}_n^*$

- $\mathbb{Z}_n$ is the set of all integers modulo $n$.
- $\mathbb{Z}_n^*$ is the set of mod $n$ integers that are relatively prime to $n$.
  - E.g., If $n$ is prime, then $\mathbb{Z}_n^* = \{1, 2, \ldots, n-1\}$
  - Note that 0 is not in $\mathbb{Z}_n^*$
- $\mathbb{Z}_n^*$ is closed under multiplication (i.e., if $a$, $b$ are in $\mathbb{Z}_n^*$, then $ab$ is in $\mathbb{Z}_n^*$)
Euler Totient Function $\Phi(n)$

- Euler Totient function $\Phi(n)$ is defined as the number of elements in $\mathbb{Z}_n^*$.
  - E.g., if $n$ is prime, then $\Phi(n) = n-1$
  - If $n = pq$ for primes $p$ and $q$, what is $\Phi(n)$?

- Euler Theorem: For all $a$ in $\mathbb{Z}_n^*$, $a^{\Phi(n)} = 1 \mod n$.

- Fermat’s Little Theorem: If $n$ is prime and for any integer $a$ that is relative prime to $n$, $a^{n-1} = 1 \mod n$.
Roadmap

- Number theory
- Public key cryptography
  - RSA
  - Diffie-Hellman
  - DSA
  - Certificates
Public Key Cryptography

➢ Symmetric key cryptography
  • Assumption: both sender and receiver share the same secret key
  • But how to agree on the same key first?

➢ Public key cryptography
  • Use two keys – public and private keys
    • Public key for encryption – known to everyone
    • Private key for decryption – known to receiver only
  • Complement symmetric key crypto
Public Key Cryptography

plaintext message, $m$

$K_B^+(m)$

ciphertext

decryption algorithm

$m = K_B^-(K_B^+(m))$

Requirements:

1. $m = K_B^- K_B^+(m)$

2. Given $K_B^+$, computationally infeasible to tell $K_B^-$
**RSA**

- Developed by Rivest, Shamir & Adleman of MIT in 1977
- Best known & widely used public-key scheme
- Based on exponentiation in $\mathbb{Z}_n^*$ over integers
  - Exponentiation takes $O((\log n)^3)$ operations (easy)
- Uses large integers (e.g., 1024 bits)
- Security due to cost of factoring large numbers
  - Factorization takes $O(e^{\log n \log \log n})$ operations (hard)
RSA: Create Public/Private Key Pair

- Choose two large prime numbers $p$ and $q$ (e.g., 1024 bits each)
- Compute $n = pq$. Hence $\Phi(n) = (p-1)(q-1)$.
- Choose $e$ ($e < n$) relatively prime to $\Phi(n)$
- Compute $d$ s.t. $ed = 1 \mod \Phi(n)$
- Public key is $(n, e)$. Private key is $(n, d)$.
- Note that $p$ and $q$ must be kept secret.
RSA: Encryption/Decryption

Given message $m < n$,

- encryption: $c = m^e \mod n$
- decryption: $m = c^d \mod n$

Why it works?

- $c^d = m^{ed} \mod n$
- $= m^{\phi(n)k + 1} \mod n$ (for some $k$)
- $= m$ (by Euler’s theorem)
RSA Example: Key Setup

- Select primes: $p=17$ & $q=11$
- Calculate $n = pq = 17 \times 11 = 187$
- Calculate $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
- Select $e$: $\gcd(e, 160) = 1$; choose $e=7$
- Determine $d$: $de=1 \mod 160$ and $d < 160$
  Value is $d=23$ since $23 \times 7 = 161 = 10 \times 160 + 1$
- Publish public key $PU=\{7, 187\}$
- Keep secret private key $PR=\{23, 187\}$
RSA Example: En/Decryption

- Sample RSA encryption/decryption is:
- Given message \( M = 88 \) (88 < 187)

- Encryption:
  \[ C = 88^7 \mod 187 = 11 \]

- Decryption:
  \[ M = 11^{23} \mod 187 = 88 \]
Security of RSA

- Main assumption (fact): factoring a large number $n = pq$ is hard.
- Fact: The problem of computing $d$ from the public key $(n, e)$, and the problem of factoring $n$, are computationally equivalent.
Efficiency of RSA

- Fact: RSA is no less secure (by far) if $e$ is always chosen to be the same number.

- Typical values of $e$: 3 and 65537.

- If $e = 3$, some practical constraints need to be imposed:
  - If $m$ is small, then $c = m^3$ is small. Attackers can readily determine $m$ by computing the ordinary cube root.
Attacks on RSA

If $e = 3$, and the same message $m$ is sent encrypted to three different parties with different moduli (e.g., $c_i = m^3 \mod n_i$ for $i=1, 2, 3$), then an attacker can determine $m$ from $c_i$.

How?

- Attacker can efficiently find $x$ s.t. $x = c_i \mod n_1 n_2 n_3$.
- Since $m < n_i$ for each $i$, $m^3 < n_1 n_2 n_3$, by Chinese Remainder Theorem, $x = m^3$.
- Also, $m^3 \mod n_1 n_2 n_3$ will just be $m^3$. Then $m$ can be obtained by the cube root of $m^3$. 
Attacks on RSA

- Scenario: a central authority selects a single RSA modulus $n$, and distributes a distinct encryption/decryption exponent pair $(e_i, d_i)$ to each entity in a network.

- Common modulus attack:
  - if a single message $m$ were encrypted and sent to two or more entities in the network, then there is a technique by which an eavesdropper could recover $m$.

- Design rule: each $n$ must be associated with a unique $(e, d)$. 
Public-Key Cryptography Standard (PKCS)

- Useful to have some standard for encoding of information that is signed or encrypted via RSA, so that different implementations can interwork, and pitfalls are avoided.

- PKCS defines the encoding of RSA keys, signatures, encrypted messages.

- You’ll know more when using OpenSSL.
Roadmap

➢ Number theory

➢ Public key cryptography
  • RSA
  • Diffie-Hellman
  • DSA
  • Certificates
Diffie-Hellman Key Exchange

- First public-key type scheme proposed
  - Earlier than RSA
- By 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
- A practical method for public exchange of a secret key
Diffie-Hellman Key Exchange

- A public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants
    - Can be extended to more than two parties
- value of key depends on the participants (and their private and public key information)
- based on exponentiation in $\mathbb{Z}_n^*$ (modulo a prime or a polynomial) - easy
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard
Diffie-Hellman Setup

➢ Two users A and B agree on global (public) parameters:
  • large prime integer $p$
  • $g$ being a generator (primitive root) mod $p$

➢ Each user (eg. A) generates their key
  • chooses a secret key (number): $x_A < p$
  • compute their public key: $y_A = g^{x_A} \mod p$

➢ each user makes public that key $y_A$
Diffie-Hellman Key Exchange

- Key exchange process:
  - A sends to B: $y_A = g^{x_A}$
  - B sends to A: $y_B = g^{x_B}$

- shared session key for users A & B is $K_{AB}$:
  $K_{AB} = g^{x_A \cdot x_B} \mod p$
  $= y_A^{x_B} \mod p$ (which B can compute)
  $= y_B^{x_A} \mod p$ (which A can compute)

- $K_{AB}$ is used as session key in private-key encryption scheme between Alice and Bob
- attacker needs an $x$, must solve discrete log
Diffie-Hellman Example

- users Alice & Bob who wish to swap keys:
- agree on prime $p=353$ and $g=3$
- select random secret keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- compute respective public keys:
  - $y_A = 3^{97} \mod 353 = 40$ (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)
Man-in-the-middle Attack on DH

Alice  \[ g^{x_A} \]  Trudy  \[ g^{x_T} \]  Bob

\[ g^{x_B} \]

K_{AT} = g^{x_A x_T}  
K_{BT} = g^{x_B x_T}

Trudy can intercept messages between Alice and Bob

Why MITM attack work?

- Alice and Bob can’t verify if the public keys are actually from Alice and Bob, respectively
Defenses Against MITM

- Authenticated Key Exchange

- Examples:
  - Encrypt public keys with pre-shared secret
  - Encrypt public keys with other side’s public keys
  - **Sign DH values with private keys**
    - will show how this approach works in a moment
Digital Signatures

- Digital signatures provide the ability to:
  - verify author, date & time of signature
  - authenticate message contents
  - be verified by third parties to resolve disputes
- Analogous to hand-written signatures
- Goal is similar to Message Authentication Code (MAC), but using public key crypto.
Roadmap

- Number theory
- Public key cryptography
  - RSA
  - Diffie-Hellman
  - DSA
  - Certificates
Digital Signatures: Idea

- **Encryption/decryption**
  - Sender encrypts with public key
  - Receiver decrypts with private key

- **Signature/verification** – two main operations
  - Sender signs with private key
  - Receiver verifies with public key

- Can be directly based on RSA, but we will show more efficient approaches.
Digital Signatures: Naive Approach

Naive approach: sign message $m$ directly

Bob’s message, $m$

Dear Alice
Oh, how I have missed you. I think of you all the time! …(blah blah blah)
Bob

Bob’s private key

Public key encryption algorithm

$K_B^-(m)$

Bob’s message, $m$, signed (encrypted) with his private key
Digital Signatures: Signed Message Digest

Bob sends digitally signed message:

- **large message** \( m \)
- **H**: Hash function
- **H(m)**
- **digital signature (encrypt)** \( K_B^{-1}(H(m)) \)
- **encrypted msg digest**

Alice verifies signature and integrity of digitally signed message:

- **large message** \( m \)
- **H**: Hash function
- **H(m)**
- **Bob’s public key** \( K_B^+ \)
- **digital signature (decrypt)**
- **encrypted msg digest** \( K_B^{-1}(H(m)) \)
- **equal?**
Digital Signatures with Encryption

Signing before encrypting:

- Alice signs the message with her private key
  - $S_A(M)$
- Alice encrypts the message and signature with Bob’s public key
  - $E_B(S_A(M))$
- Bob decrypts message with his private key
  - $D_B(E_B(S_A(M))) = S_A(M)$
- Bob verifies with Alice’s public key
  - $V_A(S_A(M)) = M$

Why “sign before encrypt”, but not “sign after encrypt”? 

Digital Signature Standard (DSS)

- **DSS**
  - a US Govt approved signature standard designed by NIST & NSA in early 90's
  - based on the SHA hash algorithm

- **DSS is the standard, DSA is the algorithm**

- **DSA**
  - digital signature only
  - unlike RSA, cannot be used for encryption and key exchange
  - It is a public-key technique
DSA vs. RSA

(a) RSA Approach

(b) DSS Approach
Digital Signature Algorithm (DSA)

- Creates a 320 bit signature with 512-1024 bit security
- Smaller and faster than RSA
- A digital signature scheme only, not for encryption or key exchange
- Security depends on difficulty of computing discrete logarithms
- Details not covered here.
Digital Signature with Key Exchange

- Let’s revisit Diffie-Hellman Key Exchange
- A sends to B: $g^{x_A}$, $\text{sign}_A(g^{x_A})$
- B sends to A: $g^{x_B}$, $\text{sign}_B(g^{x_B})$
- A and B verify with B’s and A’s public keys, resp.
- This defends against MITM attack (why?)
- But, how A and B obtain the public keys for signatures in the first place?
Roadmap

- Number theory
- Public key cryptography
  - RSA
  - Diffie-Hellman
  - DSA
  - Certificates
Public Key Infrastructure (PKI)

- We need some trusted public key authority to authenticate keys.
- Public key infrastructure (PKI) consists of the components necessary to securely distribute public/private keys.
Public Key Certificates

- **Certificates** allow key exchange without real-time access to public-key authority.

- A certificate binds **identity** to **public key**
  - usually with other info such as period of validity, rights of 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
  - usually pre-installed in applications
Operations on Certificate

- **Certification authority (CA):** binds public key to particular entity (e.g., Bob).
- Bob registers his public key with CA.
  - Bob provides “proof of identity” to CA.
  - CA creates certificate binding Bob to his public key.
  - certificate containing Bob’s public key digitally signed by CA – CA says “this is Bob’s public key”

![Diagram showing the process of certifying a public key with a certification authority. The diagram includes symbols for Bob’s public key, CA’s private key, and a certificate for Bob’s public key, signed by CA.]
When Alice wants Bob’s public key:

- gets Bob’s certificate (Bob or elsewhere).
- apply CA’s public key to Bob’s certificate, get Bob’s public key

\[ K_B^+ \]

\[ K_{CA}^+ \]

\[ \text{digital signature (decrypt)} \]

\[ K_B^+ \text{ Bob's public key} \]
Long-term/Short-term Keys

- Certificates aim to bind keys. But how are they different with the key exchange algorithms (e.g., Diffie-Hellman)?
  - Certificates deal with long-term public/private keys, which remain the same for a very long time
  - Key exchange algorithms deal with short-term public/private keys, which change in every session
X.509 Authentication Service

- X.509 defines framework for authentication services
  - directory may store public-key certificates
  - with public key of user signed by certification authority
- It also defines authentication protocols
- It uses public-key crypto & digital signatures
  - algorithms not standardised, but RSA recommended
- X.509 certificates are widely used
  - have 3 versions
X.509 Certificate

(a) X.509 Certificate

(b) Certificate Revocation List

CRL discussed later
Obtaining a Certificate

- Any user with access to CA can get any certificate from it
- Only the CA can modify a certificate
- Because certificates cannot be forged, they can be placed in a public directory
CA Hierarchy

- If both users share a common CA then they are assumed to know its public key.
- Otherwise CA's must form a hierarchy.
- Use certificates linking members of hierarchy to validate other CA's:
  - each CA has certificates for clients (forward) and parent (backward).
Denote
CA<<A>> certificate for A signed by CA

X and W can mutually verify each other
Cert of X signed by W
Cert of W signed by X
Cert of Z signed by X
CA Hierarchy Use

- Chains of certificates:
  - A acquires B certificate using chain:
    X<<W>>W<<V>>V<<Y>>Y<<Z>>Z<<B>>>
  - B acquires A certificate using chain:
    Z<<Y>>Y<<V>>V<<W>>W<<X>>X<<A>>>

- Each user only needs to contact one CA (A contacts X, B contacts Z), and unwraps the certificates signed by other CAs through the chain of trusts
Certificate Revocation

- certificates have a period of validity
- may need to revoke before expiry, eg:
  1. user's private key is compromised
  2. user is no longer certified by this CA
  3. CA's certificate is compromised
- CA’s maintain list of revoked certificates
  - the Certificate Revocation List (CRL)
- users should check certificates with CA’s CRL
References

- Kaufman et al., Ch. 5, 6, 7
- Stallings, Ch. 8 – 14
- Kurose & Ross, Ch. 8