Special Topics in Cryptography

Mohammad Mahmoody

Last time

- Public-key encryption and key-agreement
- Diffie Hellman (key agreement protocol)

Today

- RSA public key encryption
- Digital signatures

Public Key Encryption

- Secure communication even without shared secret keys!

Recalling Public Key Encryption

- Key generation: $\mathrm{Gen}(1^n) \rightarrow (ek, dk)$ public secret
- Encryption: $\mathrm{Enc}(ek, m) \rightarrow c$
- Decryption: $\mathrm{Dec}(dk, c) = m'$

- Completeness: decrypting correction $m = m'$

- Security: same as CPA security for private-key, but the adversary does not need any encryption oracle: it has the encryption key itself!

Recalling Key Agreement

- **Interactive** protocol between **randomized** Alice and Bob
- The sequence of messages $T = (t_1, t_2, \dots t_m)$ is called "transcript"
- At the end, Alice and Bob output key k_A, k_B

- Completeness: getting same keys $k_A = k_b = key$
- Security: suppose $|key| = n$, then
  (T, key) is computationally indistinguishable from (T, U_n)
  Namely, even if T is known, key is indistinguishable from uniform U_n

Diffie Hellman Key Agreement

mod q

$\{g^1, g^2, \ldots g^{q-1}\} = \{1, 2, 3 \ldots q-1\}$

- Public parameters: large prime q a (multiplicative) generator g for $\boldsymbol{Z_q}$

1. Alice picks $x \leftarrow \{1, \ldots q-1\}$ and Bob picks $y \leftarrow \{1, \ldots q-1\}$ at random
2. Alice sends $a = g^x$ to Bob and Bob sends $b = g^y$ to Alice.
3. Alice takes $b^x = g^{xy} = k$ and Bob takes $a^y = g^{xy} = k$ as the key.

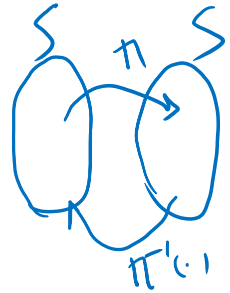
key

- To implement in **efficiently**, we use “fast exponentiation” algorithm that computes $a^y \bmod q$ in time **polynomial in lengths** of q, y, a

$(T = (g^x, g^y), \text{key} = g^{xy}) \approx_{ind} ((g^x, g^y), g^r) \quad r \in \{1, \ldots q-1\}$

RSA Public-Key Encryption

Main Idea of RSA: Trapdoor Permutations



S

- Intuition: Permutation $\pi : \{0, \dots N-1\} \to \{0, \dots N-1\}$ π(i) π is a bijection
  1. Easy to compute π publically
  2. Easy to "invert" π only if have the trapdoor.

π, σ both efficiently computable

- Key generation find: $ek = \pi$ and $dk = \sigma$ for permutations π, σ such that :
  1. for all $x \in \{0, \dots N-1\} : \sigma(\pi(x)) = x$ namely $\sigma(\cdot) = \pi^{-1}(\cdot)$
  2. Given $\pi(x)$ it is "hard" to invert it to find x. Formally, for all poly-time A

$$\Pr_{x \xleftarrow{\$} \{0,\dots N-1\}} [A(\overset{\pi(\cdot)}{\pi}(x)) = x] \le \text{negl}(n)$$ where $n \approx \log(N)$ is sec parameter

Comp(π/σ, x) ↓ π(x) σ(x)

- ~~What is useful intuitively?~~
- ~~Why cannot we use it naively~~?

③ implicit: if A also has description of σ then inverting π(x)=y back to x is easy just compute σ(y)=x

Number Theory 101- (continued)

gcd(N, M) = 1 : relatively prime

pq

pr

- $\gcd(N, M)$ = greatest common divisor of M, N

1, 2, ~~p~~ ~~2p~~ 3p, pq

- $\varphi(N) = |\{i \mid 1 \le i \le N, \gcd(N, i) = 1\}|$

pq − q − p + 1 = (p−1)(q−1)

- $\varphi(p)$ for prime p ? p−1

p = 2 q = 3

(p−1) × (q−1)

- $\varphi(pq)$ for primes $p \ne q$?

(p−1)(q−1) = 1×2 = 2

$\varphi(6) = |\{1, 5\}| = 2$

- Euler's theorem: if $\gcd(a, N) = 1 \rightarrow a^{\varphi(N)} = 1 \pmod N$

RSA Trapdoor Permutation

$\{0, \_\_\_\_ N-1\}$

$\gcd(a, N) = 1$

- Euler's theorem: if $\gcd(a, N) = 1 \rightarrow a^{\varphi(N)} = 1 \pmod N$
- $\underset{a^x}{a^{\varphi(N)}} \cdot a^{\varphi(N)} \cdot a^{\varphi(N)} \ldots = \underset{a^x}{1} \quad \rightarrow \quad a^{\underbrace{1+k\cdot\varphi(N)}} = a \pmod N$
- If $e \cdot d = 1 + k \cdot \varphi(N)$ which is the same as $e \cdot d = 1 \left(\bmod \varphi(n)\right)$ then $a^{ed} = (a^e)^d = a \,(\bmod \varphi(N))$

  which means $\pi(x) = x^e$ is inverse of $\sigma(y) = y^d$ for $\gcd(x, N) = 1$
- Interestingly $\sigma\big(\pi(x)\big) = x$ even for $\gcd(x, N) \neq 1$

RSA: $N = p \times q$ Prime p, q

How to use ~~RSA~~ any Trapdoor Permutation for public-key encryption?

- Intuition: Permutation $\pi : \{0, \dots N-1\} \to \{0, \dots N-1\}$
  1. Easy to compute π publically
  2. Easy to "invert" π only if have the trapdoor.

- What is useful intuitively?

Public key: π Private/decryption key: σ

encryption key

$\mathrm{Enc}(x, \underbrace{ek}_{\pi}) = \pi(x) \leftarrow c$

$x \in \{1, \dots N-1\}$

$\mathrm{Dec}(c, \underbrace{dk}_{\sigma}) = \underbrace{\sigma(c)}$

$\sigma(\pi(x)) = x$

- Why cannot we use it naively?

b/c it is not randomized

$\to$ it cannot be CPA secure

One "correct" way to use RSA trapdoor permutation for public key encryption

$r = (r_1, — r_n)$
$s = (s_1, — s_n)$

- Actual Randomized (CPA secure) Encryption of a bi~~g~~t b:
  Pick $r, s \in \{0, \ldots, N-1\}$ at random and output $\underbrace{[\pi(r), s, \langle r, s\rangle \oplus b]}_{\text{Ciphertext}}$

$\langle r, s\rangle = \bigoplus_{i=1}^{n} r_i \cdot s_i$

Goldreich-Levin Thm: :

if $(\pi, \cancel{\ })$ is a one way permutation $\rightarrow$ all poly time A.

$$\Pr_{r,s}\left[A(s, \pi(r)) = \langle r, s\rangle\right] \leq \frac{1}{2} + \text{neg}$$

Bellare Rogaway.

More efficient way, using an "ideal" hash function

π $\quad \bar{g}' : \bar{\pi}' : \sigma$

- Let $h: \{0,1\}^n \to \{0,1\}^n$ be an "idea" hash function
  - Key gen: generate a pair $ek = g(\cdot)$, $dk = g^{-1}(\cdot)$
  - Encryption of $m \in \{0,1\}^n$: pick $r \leftarrow \{0,1\}^n$, output $c = g(r), h(r) \oplus m$ ~~$, h(m, r))$~~
  - Decryption of $c = (y, z)$: output $m = h(g^{-1}(y)) \oplus z$

- Even possible to do it efficiently in a CCA secure way using ideal hashing

$(g, \bar{g}')$ are a secure TDP.

Claim: if we you a "random function" $h(\cdot)$

→ the above scheme is provably CPA secure

We need prime numbers for RSA and DH !

- We need large prime numbers!

- How many prime numbers are there? roughly $\frac{1}{1000}$ fraction of 1000-bit number are prime

- How can we find one? Test Primality test $\begin{cases} \text{randomized} \\ \text{deterministic } \underline{2003, 2004} \end{cases}$

Public Key Authentication:
Digital Signatures

- Secure authentication without shared secret keys!

Making MACs public key (just like how we moved to public key encryption)

(public) verif-key

Public

A

B

Secret: signing key

(m, t)

used

Can make sure that m is really from Alice.

Enc

π $\sigma = \pi'$

Enc dec

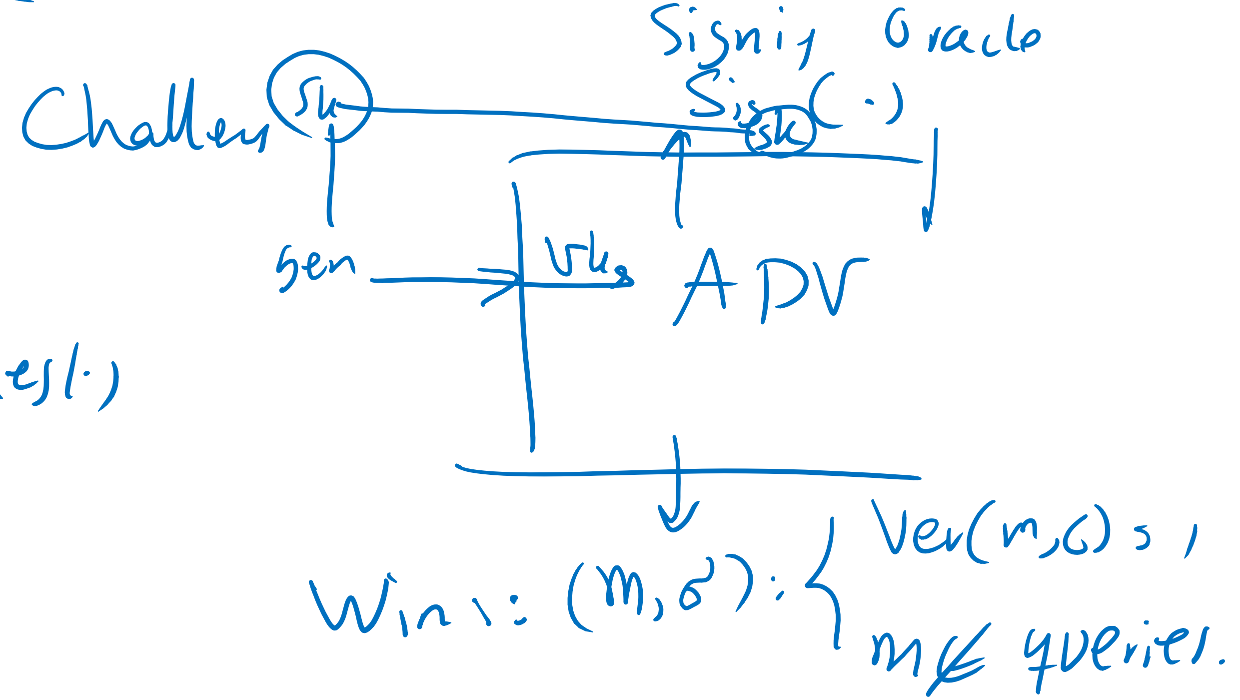
pub sec

Defining Digital Signatures

- Alice has a signing key sk and a verification key vk
- Using sk Alice can sign m with $\sigma = \text{Sign}_{sk}(m)$
- If Bob verifies $\text{Verif}_{vk}(m,\sigma) = 1$ he can be sure Alice signed m

- Security: Game

Def. $\Pr[\text{ADV wins}] < \text{negl}(\cdot)$



One possible idea based on TDPs (e.g. RSA)

- Signing key: “private key” (or the trapdoor)
- Verification key: “public key” (or the description of the permutation)

- To sign m publish $\sigma(m) = t$

- To verify (m, t) accept if and only if: $\pi(t) = m$

- Is it secure signature?

$\sigma(\pi(m)) = m$

$\pi(\sigma(m)) = m$

public π v/k

$\sigma = \pi^{-1}$ sk private

“Hash and sign” using ideal hash function