MIT 6.875 & Berkeley CS276

Foundations of Cryptography

Lecture 10
Today:
Constructions of Public-Key Encryption

1: Trapdoor Permutations (RSA) \( \text{composite } N/\text{factoring} \)

2: Quadratic Residuosity/Goldwasser-Micali \( \text{composite } N/\text{factoring} \)

3: Diffie-Hellman/El Gamal \( \text{prime } p/\text{discrete log} \)

4: Learning with Errors/Regev \( \text{small numbers, large dimensions} \)
Trapdoor One-Way Functions

\[ F \]

- Easy to compute
- Hard to invert
- Easy to invert given a trapdoor

Domain = Range
Review: Number Theory

Let’s review some number theory from L7-8.

Let $N = pq$ be a product of two large primes.

Fact: $\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$ is a group.
  
  • group operation is multiplication mod $N$.
  • inverses exist and are easy to compute (how so?)
  • the order of the group is $\phi(N) = (p - 1)(q - 1)$

Lecture 8: The map $F(x) = x^2 \mod N$ is a 4-to-1 trapdoor function, as hard to invert as factoring $N$. 
The RSA Trapdoor Permutation

Today: Let \( e \) be an integer with \( \gcd(e, N) = 1 \). Then, the map \( F_{N,e}(x) = x^e \mod N \) is a trapdoor permutation.

**Key Fact:** Given \( d \) such that \( ed = 1 \mod \phi(N) \), it is easy to compute \( x \) given \( x^e \).

**Proof:** \((x^e)^d\)

This gives us the RSA trapdoor permutation collection.

\[ \{F_{N,e} : \gcd(e, N) = 1\} \]

Trapdoor for inversion: \( d = e^{-1} \mod \phi(N) \).
The RSA Trapdoor Permutation

Today: Let $e$ be an integer with $\gcd(e, N) = 1$. Then, the map $F_{N,e}(x) = x^e \mod N$ is a trapdoor permutation.

Hardness of inversion without trapdoor = RSA assumption

given $N, e$ (as above) and $x^e \mod N$, hard to compute $x$.

We know that if factoring is easy, RSA is broken (and that’s the only known way to break RSA)

Major Open Problem: Are factoring and RSA equivalent?
The RSA Trapdoor Permutation

Today: Let $e$ be an integer with $\gcd(e, N) = 1$. Then, the map $F_{N,e}(x) = x^e \mod N$ is a trapdoor permutation.

Hardcore bits (galore) for the RSA trapdoor one-way perm:

- The Goldreich-Levin bit $GL(r; r') = \langle r, r' \rangle \mod 2$
- The least significant bit $\text{LSB}(r)$
- The “most significant bit” $\text{HALF}_N(r) = 1$ iff $r < N/2$
- In fact, any single bit of $r$ is hardcore.
RSA Encryption

- $Gen(1^n)$: Let $N = pq$ and $(e, d)$ be such that $ed = 1 \mod \phi(N)$.

  Let $pk = (N, e)$ and let $sk = d$.

- $Enc(pk, b)$ where $b$ is a bit: Generate random $r \in Z_N^*$ and output $r^e \mod N$ and $\text{LSB}(r) \oplus m$.

- $Dec(sk, c)$: Recover $r$ via RSA inversion.

IND-secure under the RSA assumption: given $N, e$ (as above) and $r^e \mod N$, hard to compute $r$. 
Today:
Constructions of Public-Key Encryption

1: Trapdoor Permutations (RSA)
2: Quadratic Residuosity/Goldwasser-Micali
3: Diffie-Hellman/El Gamal
4: Learning with Errors/Regev
Quadratic Residuosity

Let’s review some more number theory from L7-8.

Let $N = pq$ be a product of two large primes.

$Z_N^*$

$\begin{array}{c}
\{x: \left(\frac{x}{N}\right) = -1\} \\
\{x: \left(\frac{x}{N}\right) = +1\}
\end{array}$

Jacobi symbol $\left(\frac{x}{N}\right) = \left(\frac{x}{p}\right) \left(\frac{x}{q}\right)$ is +1 if $x$ is a square mod both $p$ and $q$ or a non-square mod both $p$ and $q$. 
Quadratic Residuosity

Let’s review some more number theory from L7-8.

Let $N = pq$ be a product of two large primes.

Surprising fact: Jacobi symbol $(\frac{x}{N}) = (\frac{x}{p})(\frac{x}{q})$ is computable in poly time without knowing $p$ and $q$. 
Let’s review some *more* number theory from L7-8.

Let $N = pq$ be a product of two large primes.

So: $QR_N = \{x: \left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = +1\}$

$QNR_N = \{x: \left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = -1\}$

$QR_N$ is the set of squares mod $N$ and $QNR_N$ is the set of non-squares mod $N$ with Jacobi symbol $+1$. 

$Jac_{+1}$
Quadratic Residuosity

Let’s review some *more* number theory from L7-8.

Let $N = pq$ be a product of two large primes.

**Quadratic Residuosity Assumption (QRA)**

Let $N = pq$ be a product of two large primes. No PPT algorithm can distinguish between a random element of $QR_N$ from a random element of $QNR_N$ given only $N$. 
Goldwasser-Micali (GM) Encryption

\textit{Gen}(1^n): Generate random \( n \)-bit primes \( p \) and \( q \) and let \( N = pq \). Let \( y \in \mathbb{QNR}_N \) be some quadratic non-residue with Jacobi symbol +1.

Let \( pk = (N, y) \) and let \( sk = (p, q) \).

\textit{Enc}(pk, b) \text{ where } b \text{ is a bit: }
Generate random \( r \in \mathbb{Z}_N^* \) and output \( r^2 \mod N \) if \( b = 0 \) and \( r^2 y \mod N \) if \( b = 1 \).

\textit{Dec}(sk, c): \text{ Check if } c \in \mathbb{Z}_N^* \text{ is a quadratic residue using } p \text{ and } q. \text{ If yes, output 0 else 1.}
Goldwasser-Micali (GM) Encryption

$Enc(pk, b)$ where $b$ is a bit:
Generate random $r \in \mathbb{Z}_N^*$ and output $r^2 \mod N$ if $b = 0$ and $r^2 y \mod N$ if $b = 1$.

IND-security follows directly from the quadratic residuosity assumption.
GM is a Homomorphic Encryption

Given a GM-ciphertext of $b$ and a GM-ciphertext of $b'$, I can compute a GM-ciphertext of $b + b' \mod 2$. without knowing anything about $b$ or $b'$!

$Enc(pk, b)$ where $b$ is a bit:
Generate random $r \in \mathbb{Z}_N^*$ and output $r^2 y^b \mod N$.

Claim: $Enc(pk, b) \cdot Enc(pk, b')$ is an encryption of $b \oplus b' = b + b' \mod 2$. 
Today:
Constructions of Public-Key Encryption

1: Trapdoor Permutations (RSA)

2: Quadratic Residuosity/Goldwasser-Micali

3: Diffie-Hellman/El Gamal

4: Learning with Errors/Regev
**Diffie-Hellman Key Exchange**

Commutativity in the exponent: \((g^x)^y = (g^y)^x\)  
(where \(g\) is an element of some group)

So, you can compute \(g^{xy}\) given either \(g^x\) and \(y\), or \(g^y\) and \(x\).

**Diffie-Hellman Assumption (DHA):**

Hard to compute \(g^{xy}\) given only \(g\), \(g^x\) and \(g^y\)
Diffie-Hellman Assumption (DHA):
Hard to compute it given only $g$, $g^x$ and $g^y$

We know that if discrete log is easy, DHA is false.

Major Open Problem:
Are discrete log and DHA equivalent?
**Diffie-Hellman Key Exchange**

\[ p, g: \text{Generator of our group } Z_p^* \]

\[ g^x \mod p \]

\[ g^y \mod p \]

**Pick a random number** \( x \in Z_p^* \)

**Pick a random number** \( y \in Z_p^* \)

**Shared key** \( K = g^{xy} \mod p \)

\[ K = (g^y)^x \mod p \]

\[ K = (g^x)^y \mod p \]
Diffie-Hellman/El Gamal Encryption

• $Gen(1^n)$: Generate an $n$-bit prime $p$ and a generator $g$ of $\mathbb{Z}_p^*$. Choose a random number $x \in \mathbb{Z}_{p-1}$

Let $pk = (p, g, g^x)$ and let $sk = x$.

• $Enc(pk, m)$ where $m \in \mathbb{Z}_p^*$: Generate random $y \in \mathbb{Z}_{p-1}$ and output $(g^y, g^{xy} \cdot m)$

• $Dec(sk = x, c)$: Compute $g^{xy}$ using $g^y$ and $x$ and divide the second component to retrieve $m$.

Is this Secure?
The Problem

Claim: Given $p$, $g$, $g^x \mod p$ and $g^y \mod p$, adversary can compute some information about $g^{xy} \mod p$.

Corollary: Therefore, additionally given $g^{xy} \cdot m \mod p$, the adversary can determine whether $m$ is a square mod $p$, violating “IND-security”.
The Problem

Claim: Given $p, g, g^x \mod p$ and $g^y \mod p$, adversary can determine if $g^{xy} \mod p$ is a square mod $p$.

$g^{xy} \mod p$ is a square $\iff xy \pmod{p-1}$ is even

$\iff xy$ is even

$\iff x$ is even or $y$ is even

$\iff x \pmod{p-1}$ is even or $y \pmod{p-1}$ is even

$\iff g^x \mod p$ or $g^y \mod p$ is a square

This can be checked in poly time!
Diffie-Hellman Encryption

Claim: Given $p, g, g^x \mod p$ and $g^y \mod p$, adversary can determine if $g^{xy} \mod p$ is a square mod $p$.

More generally, dangerous to work with groups that have non-trivial subgroups (in our case, the subgroup of all squares mod $p$)

**Lesson:** Best to work over a group of prime order. Such groups have no subgroups.

**An Example:** Let $p = 2q + 1$ where $q$ is a prime itself.
Then, the group of squares mod $p$ has order $\frac{(p-1)}{2} = q$. 
Diffie-Hellman/El Gamal Encryption

- **Gen($1^n$)**: Generate an $n$-bit “safe” prime $p = 2q + 1$ and a generator $g$ of $Z_p^*$ and let $h = g^2 \mod p$ be a generator of $QR_p$. Choose a random number $x \in Z_q$.

  Let $pk = (p, h, h^x)$ and let $sk = x$.

- **Enc($pk, m$) where $m \in QR_p$** : Generate random $y \in Z_q$ and output $(g^y, g^{xy} \cdot m)$

- **Dec($sk = x, c$)**: Compute $g^{xy}$ using $g^y$ and $x$ and divide the second component to retrieve $m$. 
Decisional Diffie-Hellman Assumption

\textit{Decisional} Diffie-Hellman Assumption (DDHA):

Hard to distinguish between \( g^{xy} \) and a uniformly random group element, given \( g, g^x \) and \( g^y \)

That is, the following two distributions are computationally indistinguishable:

\[
(g, g^x, g^y, g^{xy}) \approx (g, g^x, g^y, u)
\]

DH/El Gamal is IND-secure under the DDH assumption.
Today: Constructions of Public-Key Encryption

1: Trapdoor Permutations (RSA)

2: Quadratic Residuosity/Goldwasser-Micali

3: Diffie-Hellman/El Gamal

4: Learning with Errors/Regev
   (post-quantum secure, as far as we know)
Solving Linear Equations

How about:

\[(s_1 | s_2) \begin{bmatrix} 5 & 1 & 3 \\ 6 & 2 & 1 \end{bmatrix} + [e_1 \ e_2 \ e_3] = [11 \ 3 \ 9] \]

(e_1,e_2,e_3) are “small” numbers

Very hard!

Find \( \hat{s} \)

in large dimensions
Learning with Errors (LWE)

[Regev05, following BFKL93, Ale03]

LWE:

\[(A, sA + e)\]

\(A \in \mathbb{Z}_q^{n \times m}\)
\(s \in \mathbb{Z}_q^n\) random "small" secret vector
\(e \in \mathbb{Z}_q^n\) random "small" error vector

Find \(s\)

very hard!

Decisional LWE:

\[(A, sA + e) \approx (A, b)\]

\(b\) uniformly random

“Decisional LWE is as hard as LWE”.
Basic (Secret-key) Encryption

Secret key \( sk = \) Uniformly random vector \( s \in \mathbb{Z}_q \)

Encryption \( Enc_s(m) \):  \( m \in \{0,1\} \)

- Sample uniformly random \( a \in \mathbb{Z}_q \), “short” noise \( e \in \mathbb{Z} \)
- The ciphertext \( c = (a, b = \langle a, s \rangle + e + m) \)

Decryption \( Dec_{sk}(c) \): Output \( \langle b - \langle a, s \rangle \mod q \rangle \)

// correctness as long as \(|e| < q/4\)

\( n \) = security parameter, \( q \) = “small” prime

[Regev05]
Basic (Secret-key) Encryption

[Regev05]

This is an incredibly cool scheme. In particular, additively homomorphic.

\[ c = (a, b = \langle a, s \rangle + e + m \ [q/2]) \quad + \]

\[ c' = (a', b' = \langle a', s \rangle + e' + m' \ [q/2]) \]

\[ c + c' = (a+a', b+ b' = \langle a +a', s \rangle + (e+e') + (m+m') \ [q/2]) \]

In words: \( c + c' \) is an encryption of \( m+m' \) (mod 2)
Public-key Encryption

[Regev05]

Here is a crazy idea. Public key has an encryption of 0 (call it $c_0$) and an encryption of 1 (call it $c_1$). If you want to encrypt 0, output $c_0$ and if you want to encrypt 1, output $c_1$.

Well, turns out to be a crazy bad idea.

If only we could produce fresh encryptions of 0 or 1 given just the pk...
Public-key Encryption
[Regev05]

Here is another crazy idea. Public key has *many* encryptions of 0 and an encryption of 1 (call it $c_1$).

If you want to encrypt 0, output a random linear combination of the 0-encryptions.

If you want to encrypt 1, output a random linear combination of the 0-encryptions plus $c_1$.

This one turns out to be a crazy *good* idea.
Public-key Encryption
[Regev05]

- Secret key $sk =$ Uniformly random vector $s \in Z_q^n$
- Public key $pk$: for $i$ from 1 to $k = \text{poly}(n)$
  
  $\left( c_0 = (a_0, \langle a_0, s \rangle + e_0 + \lfloor \frac{q}{2} \rfloor), c_i = (a_i, \langle a_i, s \rangle + e_i) \right)$

- Encrypting a bit $m$: pick $k$ random bits $r_1, ..., r_k$
  
  $\sum_{i=1}^{k} r_i c_i + m \cdot c_0$

Correctness: additive homomorphism

Security: decisional LWE + “Leftover Hash Lemma”
We saw:
Constructions of Public-Key Encryption

1: Trapdoor Permutations (RSA)

2: Quadratic Residuosity/Goldwasser-Micali

3: Diffie-Hellman/El Gamal

4: Learning with Errors/Regev
I want to encrypt to Bob. How do I know his public key?

Public-key Infrastructure: a directory of identities together with their public keys.

Needs to be “authenticated”: otherwise Eve could replace Bob’s pk with her own.
Practical Considerations

Public-key encryption is orders of magnitude slower than secret-key encryption.

1. We just showed how to encrypt bit-by-bit! Super-duper inefficient.
2. Exponentiation takes $O(n^2)$ time as opposed to typically linear time for secret key encryption (AES).
3. The $n$ itself is large for PKE (RSA: $n \geq 2048$) compared to SKE (AES: $n = 128$).

Can solve problem 1 and minimize problems 2&3 using hybrid encryption.
Hybrid Encryption

To encrypt a long message $m$ (think 1 GB):

Pick a random key $K$ (think 128 bits) for a secret-key encryption

Encrypt $K$ with the PKE: $PKE.\ Enc(pk, K)$

Encrypt $m$ with the SKE: $SKE.\ Enc(K, m)$

To decrypt: recover $K$ using $sk$. Then using $K$, recover $m$
Next Lecture: 

Digital Signatures