Vladimir Shpilrain
The City College of New York
shpil@groups.sci.ccny.cuny.edu

February 28, 2012

# Why?

- Efficiency (smaller key size, less computation)
- Security (?)
- Trying to do something useful

# Why?

- Efficiency (smaller key size, less computation)
- Security (?)
- Trying to do something useful

# Why?

- Efficiency (smaller key size, less computation)
- Security (?)
- Trying to do something useful

One-way functions

$\longrightarrow$ easy $\longleftarrow$ hard

$$f(x) = x^n$$

**Trapdoor !**

# Public Key Cryptography

## One-way functions

$\longrightarrow$ easy $\qquad\qquad$ $\longleftarrow$ hard

$$f(x) = x^n$$

**Trapdoor !**

## One-way functions

$\longrightarrow$ easy $\longleftarrow$ hard

$$f(x) = x^n$$

**Trapdoor !**

# Public Key Cryptography

- Encryption
- Key agreement (a.k.a. key exchange, a.k.a. key establishment)
- Authentication

# Public Key Cryptography

- Encryption
- Key agreement (a.k.a. key exchange, a.k.a. key establishment)
- Authentication

# Encryption from key agreement

Let $K \in \{0,1\}^n$ be the parties' (Alice and Bob) shared secret key.

Bob encrypts his message $m \in \{0,1\}^n$ as

$$E(m) = m \oplus K,$$

where $\oplus$ is addition modulo 2.

Alice decrypts as

$$E(m) \oplus K = (m \oplus K) \oplus K = m \oplus (K \oplus K) = m.$$

If the adversary can somehow decrypt and recover $m$, then she can also recover $K = m \oplus E(m) = m \oplus (m \oplus K)$.

# Encryption from key agreement

Let $K \in \{0,1\}^n$ be the parties' (Alice and Bob) shared secret key.

Bob encrypts his message $m \in \{0,1\}^n$ as

$$E(m) = m \oplus K,$$

where $\oplus$ is addition modulo 2.

Alice decrypts as

$$E(m) \oplus K = (m \oplus K) \oplus K = m \oplus (K \oplus K) = m.$$

If the adversary can somehow decrypt and recover $m$, then she can also recover $K = m \oplus E(m) = m \oplus (m \oplus K)$.

# The Diffie-Hellman key establishment (1976)

1. Alice and Bob agree on a (finite) cyclic group $G$ and a generating element $g$ in $G$. We will write the group $G$ multiplicatively.
2. Alice picks a random natural number $a$ and sends $g^a$ to Bob.
3. Bob picks a random natural number $b$ and sends $g^b$ to Alice.
4. Alice computes $K_A = (g^b)^a = g^{ba}$.
5. Bob computes $K_B = (g^a)^b = g^{ab}$.

Since $ab = ba$ (because $\mathbb{Z}$ is commutative), both Alice and Bob are now in possession of the same group element $K = K_A = K_B$ which can serve as the shared secret key.

# Variations on Diffie-Hellman: why not just multiply them?

1. Alice and Bob agree on a (finite) cyclic group $G$ and a generating element $g$ in $G$. We will write the group $G$ multiplicatively.
2. Alice picks a random natural number $a$ and sends $g^a$ to Bob.
3. Bob picks a random natural number $b$ and sends $g^b$ to Alice.
4. Alice computes $K_A = (g^b) \cdot (g^a) = g^{b+a}$.
5. Bob computes $K_B = (g^a) \cdot (g^b) = g^{a+b}$.

Obviously, $K_A = K_B = K$, which can serve as the shared secret key.

Drawback: anybody can obtain $K$ the same way!

# Variations on Diffie-Hellman: why not just multiply them?

1. Alice and Bob agree on a (finite) cyclic group $G$ and a generating element $g$ in $G$. We will write the group $G$ multiplicatively.
2. Alice picks a random natural number $a$ and sends $g^a$ to Bob.
3. Bob picks a random natural number $b$ and sends $g^b$ to Alice.
4. Alice computes $K_A = (g^b) \cdot (g^a) = g^{b+a}$.
5. Bob computes $K_B = (g^a) \cdot (g^b) = g^{a+b}$.

Obviously, $K_A = K_B = K$, which can serve as the shared secret key.

Drawback: anybody can obtain $K$ the same way!

# Security assumptions 1

Computational Diffie-Hellman (CDH) assumption: no efficient algorithm exists to recover $g^{ab}$ from $(g, g^a, g^b)$.

More formally: A CDH algorithm $F$ for a family of groups $G$ is a probabilistic polynomial time (in $|G|$) algorithm satisfying, for some fixed $\alpha > 0$ and sufficiently large $n = \log |G|$,

$$\mathbb{P}[F(g, G, g^a, g^b) = g^{ab}] > \frac{1}{n^\alpha}.$$

The probability is over a uniformly random choice of $a$ and $b$. We say that a family of groups $G$ satisfies the CDH assumption if there is no CDH algorithm for that family.

Example: $G = \mathbb{Z}_p^*$

Discrete log problem: recover $a$ from $g^a$ $mod$ $p$.

# Security assumptions 1

Computational Diffie-Hellman (CDH) assumption: no efficient algorithm exists to recover $g^{ab}$ from $(g, g^a, g^b)$.

More formally: A CDH algorithm $F$ for a family of groups $G$ is a probabilistic polynomial time (in $|G|$) algorithm satisfying, for some fixed $\alpha > 0$ and sufficiently large $n = \log|G|$,

$$\mathbb{P}[F(g, G, g^a, g^b) = g^{ab}] > \frac{1}{n^\alpha}.$$

The probability is over a uniformly random choice of $a$ and $b$. We say that a family of groups $G$ satisfies the CDH assumption if there is no CDH algorithm for that family.

Example: $G = \mathbb{Z}_p^*$

Discrete log problem: recover $a$ from $g^a$ $mod$ $p$.

# Security assumptions 1

Computational Diffie-Hellman (CDH) assumption: no efficient algorithm exists to recover $g^{ab}$ from $(g, g^a, g^b)$.

More formally: A CDH algorithm $F$ for a family of groups $G$ is a probabilistic polynomial time (in $|G|$) algorithm satisfying, for some fixed $\alpha > 0$ and sufficiently large $n = \log |G|$,

$$\mathbb{P}[F(g, G, g^a, g^b) = g^{ab}] > \frac{1}{n^\alpha}.$$

The probability is over a uniformly random choice of $a$ and $b$. We say that a family of groups $G$ satisfies the CDH assumption if there is no CDH algorithm for that family.

Example: $G = \mathbb{Z}_p^*$

Discrete log problem: recover $a$ from $g^a$ $mod$ $p$.

# Security assumptions 1

Computational Diffie-Hellman (CDH) assumption: no efficient algorithm exists to recover $g^{ab}$ from $(g, g^a, g^b)$.

More formally: A CDH algorithm $F$ for a family of groups $G$ is a probabilistic polynomial time (in $|G|$) algorithm satisfying, for some fixed $\alpha > 0$ and sufficiently large $n = \log|G|$,

$$\mathbb{P}[F(g, G, g^a, g^b) = g^{ab}] > \frac{1}{n^\alpha}.$$

The probability is over a uniformly random choice of $a$ and $b$. We say that a family of groups $G$ satisfies the CDH assumption if there is no CDH algorithm for that family.

Example: $G = \mathbb{Z}_p^*$

Discrete log problem: recover $a$ from $g^a$ *mod* $p$.

# Security assumptions 2

Decision Diffie-Hellman (DDH) assumption: no efficient algorithm exists that can distinguish between the two probability distributions $(g^a, g^b, g^{ab})$ and $(g^a, g^b, g^c)$, where $a, b$ and $c$ are chosen at random.

This is a stronger assumption, but at least it can be supported by statistical experiments.

# Security assumptions 2

Decision Diffie-Hellman (DDH) assumption: no efficient algorithm exists that can distinguish between the two probability distributions $(g^a, g^b, g^{ab})$ and $(g^a, g^b, g^c)$, where $a, b$ and $c$ are chosen at random.

This is a stronger assumption, but at least it can be supported by statistical experiments.

# Efficiency for legitimate parties

Exponentiation by "square-and-multiply":

$$g^{22} = (((g^2)^2)^2)^2 \cdot (g^2)^2 \cdot g^2$$

Complexity of computing $g^n$ is therefore $O(\log n)$, times complexity of reducing *mod p* (more generally, reducing to a "normal form").

# Efficiency for legitimate parties

Exponentiation by "square-and-multiply":

$$g^{22} = (((g^2)^2)^2)^2 \cdot (g^2)^2 \cdot g^2$$

Complexity of computing $g^n$ is therefore $O(\log n)$, times complexity of reducing *mod p* (more generally, reducing to a "normal form").

# Efficiency for legitimate parties

Exponentiation by "square-and-multiply":

$$g^{22} = (((g^2)^2)^2)^2 \cdot (g^2)^2 \cdot g^2$$

Complexity of computing $g^n$ is therefore $O(\log n)$, times complexity of reducing *mod p* (more generally, reducing to a "normal form").

# Authentication from Diffie-Hellman

Alice is the prover, and Bob the verifier. Alice's public key is $g^a$.

1. Bob picks a random natural number $b$ and sends a *challenge* $g^b$ to Alice.
2. Alice responds with a *proof* $P = (g^b)^a = g^{ba}$.
3. Bob verifies: $(g^a)^b = P$?

# RSA

1. Alice's *private key* is a pair of large primes $p, q$, and her *public key* consists of: (1) the product $n = pq$; (2) an integer $e$ such that $1 < e < \varphi(n)$, and $e$ and $\varphi(n)$ are relatively prime. Here $\varphi(n) = (p-1)(q-1)$, the Euler function of $n$.
2. To encrypt his message $m$, which is an integer, $0 < m < n$, Bob computes $c \equiv m^e \pmod{n}$ and sends $c$ to Alice.
3. To decrypt, Alice first finds an integer $d$ such that $de \equiv 1 \pmod{\varphi(n)}$. Then she computes:

$$c^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}.$$

Now, since $ed = 1 + k\varphi(n)$, one has

$$m^{ed} \equiv m^{1+k\varphi(n)} \equiv m(m^k)^{\varphi(n)} \equiv m \pmod{n}.$$

The last congruence follows directly from Euler's generalization of Fermat's little theorem if $m$ is relatively prime to $n$. By using the Chinese remainder theorem it can be shown that this congruence holds for all $m$.

# Security assumption

No efficient (i.e., polynomial time in $\log n$) algorithm exists for factoring $n = pq$.

This is necessary, but is not known to be sufficient.

# Security assumption

No efficient (i.e., polynomial time in $\log n$) algorithm exists for factoring $n = pq$.

This is necessary, but is not known to be sufficient.

# Rabin's cryptosystem

1. Alice's private key is a pair of large primes $p, q$, where $p \equiv q \equiv 3 \pmod 4$, and her public key is the product $n = pq$.
2. If Bob wants to encrypt his message $m$, which is an integer, $0 < m < n$, he computes $c \equiv m^2 \pmod n$ and sends $c$ to Alice.
3. Alice computes square roots of $c$ modulo $p$ and modulo $q$:

   $$m_p = c^{\frac{(p+1)}{4}} \bmod p$$

   and

   $$m_q = c^{\frac{(q+1)}{4}} \bmod q.$$

   Then, by using the Chinese remainder theorem, she computes the four square roots of $c \pmod n$:

   $$\pm r = (y_p \cdot p \cdot m_q + y_q \cdot q \cdot m_p) \bmod n$$
   $$\pm s = (y_p \cdot p \cdot m_q - y_q \cdot q \cdot m_p) \bmod n.$$

   Here $y_p$ and $y_q$, such that $y_p \cdot p + y_q \cdot q = 1$, can be found by using Euclidean algorithm.

Major disadvantage: only one out of four square roots is the actual message $m$.

Major advantage: finding all four square roots of a given $c$ is polynomial-time equivalent to factoring $n = pq$.

If $n = pq$, then, given a square $x^2 \pmod n$, there are four different square roots, call them $\pm x$ and $\pm y$. If we know $x$ and $y$, then

$$(x - y)(x + y) = x^2 - y^2 = 0 \pmod n.$$

Therefore, $n = pq$ divides $(x - y)(x + y)$, so either $p$ divides $(x + y)$ and $q$ divides $(x - y)$ or vice versa. In either case we can easily find one of the prime factors of $n$ by computing $g.c.d.(x + y, n)$ using Euclidean algorithm.

# Rabin's cryptosystem (cont.)

Major disadvantage: only one out of four square roots is the actual message $m$.

Major advantage: finding all four square roots of a given $c$ is polynomial-time equivalent to factoring $n = pq$.

If $n = pq$, then, given a square $x^2 \pmod{n}$, there are four different square roots, call them $\pm x$ and $\pm y$. If we know $x$ and $y$, then

$$(x - y)(x + y) = x^2 - y^2 = 0 \pmod{n}.$$

Therefore, $n = pq$ divides $(x - y)(x + y)$, so either $p$ divides $(x + y)$ and $q$ divides $(x - y)$ or vice versa. In either case we can easily find one of the prime factors of $n$ by computing $g.c.d.(x + y, n)$ using Euclidean algorithm.

# Rabin's cryptosystem (cont.)

Major disadvantage: only one out of four square roots is the actual message $m$.

Major advantage: finding all four square roots of a given $c$ is polynomial-time equivalent to factoring $n = pq$.

If $n = pq$, then, given a square $x^2 \pmod{n}$, there are four different square roots, call them $\pm x$ and $\pm y$. If we know $x$ and $y$, then

$$(x - y)(x + y) = x^2 - y^2 = 0 \pmod{n}.$$

Therefore, $n = pq$ divides $(x - y)(x + y)$, so either $p$ divides $(x + y)$ and $q$ divides $(x - y)$ or vice versa. In either case we can easily find one of the prime factors of $n$ by computing $g.c.d.(x + y, n)$ using Euclidean algorithm.

# "Mock RSA"

There is a public group $G$ and a public automorphism $\varphi$ of $G$. Alice's private key is $\varphi^{-1}$.

1. Encryption: Bob sends $\varphi(w)$ to Alice, where $w \in G$ is his secret message.
2. Alice decrypts: $w = \varphi^{-1}(\varphi(w))$.

This encryption is *homomorphic* !

# “Mock RSA”

There is a public group $G$ and a public automorphism $\varphi$ of $G$. Alice’s private key is $\varphi^{-1}$.

1. Encryption: Bob sends $\varphi(w)$ to Alice, where $w \in G$ is his secret message.
2. Alice decrypts: $w = \varphi^{-1}(\varphi(w))$.

This encryption is *homomorphic* !

# Thank you