Department of Computer Science and Engineering
计算机科学与工程系

CSE5014 CRYPTOGRAPHY AND NETWORK SECURITY

Dr. Qi Wang

Department of Computer Science and Engineering
Office: Room413, CoE South Tower
Email: wangqi@sustech.edu.cn

The RSA problem

- Let $N = pq$ with p and q distinct, odd primes
- $\mathbb{Z}_N^* =$ *invertiable* elements under multiplication modulo N
  - The order of $\mathbb{Z}_N^*$ is $\phi(N) = (p-1) \cdot (q-1)$
  - $\phi(N)$ is *easy* to compute if p, q are known
  - $\phi(N)$ is *hard* to compute if p, q are not known
    - Equivalent (believed) to factoring N

The RSA problem

- Let $N = pq$ with p and q distinct, odd primes
- $\mathbb{Z}_N^* =$ *invertiable* elements under multiplication modulo N
  - The order of $\mathbb{Z}_N^*$ is $\phi(N) = (p-1)\cdot(q-1)$
  - $\phi(N)$ is *easy* to compute if p, q are known
  - $\phi(N)$ is *hard* to compute if p, q are not known
    - Equivalent (believed) to factoring N
- Fix e with $\gcd(e, \phi(N)) = 1$
  - Raising to the e-th power is a permutation of $\mathbb{Z}_N^*$

The RSA problem

- Let $N = pq$ with p and q distinct, odd primes
- $\mathbb{Z}_N^* =$ *invertiable* elements under multiplication modulo N
  - The order of $\mathbb{Z}_N^*$ is $\phi(N) = (p-1) \cdot (q-1)$
  - $\phi(N)$ is *easy* to compute if p, q are known
  - $\phi(N)$ is *hard* to compute if p, q are not known
    - Equivalent (believed) to factoring N
- Fix e with $\gcd(e, \phi(N)) = 1$
  - Raising to the e-th power is a permutation of $\mathbb{Z}_N^*$
- If $ed \equiv 1 \bmod \phi(N)$, raising to the d-th power is the *inverse* of raising to the e-th power
  - I.e., $(x^e)^d \equiv x \bmod N$
  - x^d is the e-th root of x modulo N

The RSA problem

- If p, q are known:
  $\Rightarrow \phi(N)$ can be computed
  $\Rightarrow d = e^{-1} \bmod \phi(N)$ can be computed
  $\Rightarrow$ possible to compute e-th roots modulo N

The RSA problem

- If p, q are known:
  $\Rightarrow \phi(N)$ can be computed
  $\Rightarrow d = e^{-1} \bmod \phi(N)$ can be computed
  $\Rightarrow$ possible to compute e-th roots modulo N

- If p, q are *not* known:
  $\Rightarrow$ computing $\phi(N)$ is as hard as factoring N
  $\Rightarrow$ computing d is as hard as factoring N

The RSA problem

- If p, q are known:
  $\Rightarrow \phi(N)$ can be computed
  $\Rightarrow d = e^{-1} \bmod \phi(N)$ can be computed
  $\Rightarrow$ possible to compute e-th roots modulo N

- If p, q are *not* known:
  $\Rightarrow$ computing $\phi(N)$ is as hard as factoring N
  $\Rightarrow$ computing d is as hard as factoring N

  $\mathcal{Q}$: Given d and e, can we factor N?

The RSA problem

- If p, q are known:
  $\Rightarrow \phi(N)$ can be computed
  $\Rightarrow d = e^{-1} \bmod \phi(N)$ can be computed
  $\Rightarrow$ possible to compute e-th roots modulo N

- If p, q are *not* known:
  $\Rightarrow$ computing $\phi(N)$ is as hard as factoring N
  $\Rightarrow$ computing d is as hard as factoring N
  $\mathcal{Q}$: Given d and e, can we factor N?

- Very useful for *public-key* cryptography

The RSA assumption (formal)

- GenRSA: on input 1^n, outputs (N, e, d) with $N = pq$ a product of two distinct n-bit primes, with $ed = 1 \bmod \phi(N)$
- Experiment RSA-inv$_{A,GenRSA}(n)$:
  - Compute $(N, e, d) \leftarrow GenRSA(1^n)$
  - Choose uniform $y \in \mathbb{Z}_N^*$
  - Run $A(N, e, y)$ to get x
  - Experiment evaluates to 1 if $x^e = y \bmod N$

The RSA assumption (formal)

- GenRSA: on input 1^n, outputs (N, e, d) with $N = pq$ a product of two distinct n-bit primes, with $ed = 1 \bmod \phi(N)$
- Experiment RSA-inv$_{A,GenRSA}(n)$:
  – Compute $(N, e, d) \leftarrow GenRSA(1^n)$
  – Choose uniform $y \in \mathbb{Z}_N^*$
  – Run $A(N, e, y)$ to get x
  – Experiment evaluates to 1 if $x^e = y \bmod N$

- The *RSA problem* is *hard* relative to GenRSA if for all PPT algorithms A,

$$\Pr[\text{RSA-inv}_{A,GenRSA}(n) = 1] < negl(n)$$

RSA and factoring

- If factoring moduli output by GenRSA is easy, then the *RSA problem* is easy relative to GenRSA
  - Factoring is easy $\Rightarrow$ RSA problem is easy

RSA and factoring

- If factoring moduli output by GenRSA is easy, then the *RSA problem* is easy relative to GenRSA
  - Factoring is easy $\Rightarrow$ RSA problem is easy

- Hardness of the *RSA problem* is not known to be implied by hardness of factoring
  - Possible factoring is hard but *RSA problem* is easy
  - Possible both are hard but *RSA problem* is “easier”
  - Currently, RSA is believed to be *as hard as factoring*

Trapdoor functions

- **Definition 10.1** (*Trapdoor functions*) A *trapdoor function collection* is a collection $\mathcal{F}$ of finite functions such that every $f \in \mathcal{F}$ is a one-to-one function from some set S_f to a set T_f. The following properties are requried.
  - Efficient generation, computation and inversion
    There is a PPT algorithm G that on input 1^n outputs a pair (f, f^{-1}), where these are two $poly(n)$ size strings that describe the functions f, f^{-1}
  - Efficient sampling There is a PPT algorithm that given f can output a random element of S_f
  - One-wayness The function f is hard to invert without knowing the *invertion key*. For all PPT A there is a negligible function ϵ s.t.

$$\Pr_{(f,f^{-1})\leftarrow_R G(1^n),\ x\leftarrow_R S_f}[A(1^n, f, f(x)) = x] < \epsilon(n)$$

RSA trapdoor function

- **Keys**: choose P, Q as random primes of length n, $N = P \cdot Q$. Choose e at random from $\{1, \ldots, \phi(N) - 1\}$ with $\gcd(e, \phi(N)) = 1$

  Forward **Key**: N, e

  Backward **Key**: d with $ed \equiv 1 \bmod \phi(N)$

  Function: $RSA_{N,e}(X) = X^e \pmod N$

  Inverse: If $Y = RSA_{N,e}(X) = X^e \bmod N$, then $Y^d \bmod N = X$.

RSA trapdoor function

- **Keys**: choose P, Q as random primes of length n, $N = P \cdot Q$. Choose e at random from $\{1, \ldots, \phi(N) - 1\}$ with $\gcd(e, \phi(N)) = 1$

  Forward **Key**: N, e

  Backward **Key**: d with $ed \equiv 1 \bmod \phi(N)$

  Function: $RSA_{N,e}(X) = X^e \pmod{N}$

  Inverse: If $Y = RSA_{N,e}(X) = X^e \bmod N$, then $Y^d \bmod N = X$.

- **RSA Assumption**: the RSA function is indeed a *trapdoor function*
  - This is stronger than the assumption that factoring is hard

Rabin's trapdoor function

- Assume that *factoring* random *Blum integers* is hard. A *Blum integer* is a number $n = pq$ where $p, q \equiv 3 \pmod 4$.

Rabin's trapdoor function

- Assume that *factoring* random *Blum integers* is hard. A *Blum integer* is a number $n = pq$ where $p, q \equiv 3 \pmod 4$.

- Define $\mathcal{B}_n := \{P \in [1 \ldots 2^n] : P \text{ prime and } P \equiv 3 \bmod 4\}$

  The Factoring Axiom For every PPT algorithm A there is a negligible function ϵ s.t.

$$\Pr_{P,Q \leftarrow_R \mathcal{B}_n}[A(P \cdot Q) = \{P, Q\}] < \epsilon(n)$$

Rabin's trapdoor function

- **Keys**: choose P, Q as random primes of length n with
  $P, Q \equiv 3 \bmod 4$, $N = P \cdot Q$.
  Forward **Key**: N
  Backward **Key**: P, Q
  Function: $Y = RABIN_N(X) = X^2 \bmod N$, which is a permutation on QR_N, where QR_N denotes the set of quadratic residues modulo N

Rabin's trapdoor function

- **Keys**: choose P, Q as random primes of length n with
  $P, Q \equiv 3 \bmod 4$, $N = P \cdot Q$.
  Forward **Key**: N
  Backward **Key**: P, Q
  Function: $Y = RABIN_N(X) = X^2 \bmod N$, which is a permutation on QR_N, where QR_N denotes the set of quadratic residues modulo N

  Inverse: Compute $A = Y \bmod P$ and $B = Y \bmod Q$. Since $P, Q \equiv 3 \bmod 4$, let $P = 4t + 3$ and $Q = 4t' + 3$.

Rabin's trapdoor function

- **Keys**: choose P, Q as random primes of length n with
  $P, Q \equiv 3 \bmod 4,\ N = P \cdot Q$.
  Forward **Key**: N
  Backward **Key**: P, Q
  Function: $Y = RABIN_N(X) = X^2 \bmod N$, which is a permutation on QR_N, where QR_N denotes the set of quadratic residues modulo N

  Inverse: Compute $A = Y \bmod P$ and $B = Y \bmod Q$. Since $P, Q \equiv 3 \bmod 4$, let $P = 4t + 3$ and $Q = 4t' + 3$.

  Compute $X_1 = A^{t+1} \bmod P$ and $X_2 = B^{t'+1} \bmod Q$. Using CRT, we find X.

Rabin's trapdoor function

- **Keys**: choose P, Q as random primes of length n with
  $$P, Q \equiv 3 \bmod 4, \; N = P \cdot Q.$$
  Forward **Key**: N
  Backward **Key**: P, Q
  Function: $Y = RABIN_N(X) = X^2 \bmod N$, which is a permutation on QR_N, where QR_N denotes the set of quadratic residues modulo N

  Inverse: Compute $A = Y \bmod P$ and $B = Y \bmod Q$. Since $P, Q \equiv 3 \bmod 4$, let $P = 4t + 3$ and $Q = 4t' + 3$.

  Compute $X_1 = A^{t+1} \bmod P$ and $X_2 = B^{t'+1} \bmod Q$. Using CRT, we find X.

  We know that $X = S^2 \bmod P$, then
  $$X_1 = (X^2)^{t+1} = S^{4(t+1)} = S^{P-1+2} = S^2 = X \bmod P.$$

  Similarly, $X_2 = S^2 = X \bmod Q$.

Rabin's trapdoor function

- **Lemma 10.2** Let X, Y be such that $X \not\equiv \pm Y \pmod{N}$ but $X^2 \equiv Y^2 \pmod{N}$. Then $\gcd(X - Y, N) \notin \{1, N\}$.

  Proof. easy.

Rabin's trapdoor function

- **Lemma 10.2** Let X, Y be such that $X \not\equiv \pm Y \pmod N$ but $X^2 \equiv Y^2 \pmod N$. Then $\gcd(X - Y, N) \notin \{1, N\}$.

  Proof. easy.

  Theorem 10.3 (*One-wayness of Rabin's function*)
  Rabin's function is a *trapdoor function* under the factoring axiom.

- **Lemma 10.2** Let X, Y be such that $X \not\equiv \pm Y \pmod N$ but $X^2 \equiv Y^2 \pmod N$. Then $\gcd(X - Y, N) \notin \{1, N\}$.

  Proof. easy.

  Theorem 10.3 (*One-wayness of Rabin's function*)
  Rabin's function is a *trapdoor function* under the factoring axiom.

  Proof. By contradiction. (see blackboard)

Cyclic groups

- Let G be a finite group of order m (written multiplicatively)

- Let g be some element of G

- Consider the set $\langle g \rangle = \{g^0, g^1, \ldots\}$

Cyclic groups

- Let G be a finite group of order m (written multiplicatively)
- Let g be some element of G
- Consider the set $\langle g \rangle = \{g^0, g^1, \ldots\}$
  - We know $g^m = 1 = g^0$, so the set has $\leq m$ elements
  - If the set has m elements, then it is all of G!

Cyclic groups

- Let G be a finite group of order m (written multiplicatively)

- Let g be some element of G

- Consider the set $\langle g \rangle = \{g^0, g^1, \ldots\}$
  - We know $g^m = 1 = g^0$, so the set has $\leq m$ elements
  - If the set has m elements, then it is all of G!
    - In this case, we say g is a *generator* of G
    - If G has a generator, we say G is *cyclic*

Examples

- $\mathbb{Z}_N$
  - Cyclic (for any N); 1 is always a generator: $\{0, 1, 2, \ldots, N-1\}$

Examples

- $\mathbb{Z}_N$
  - Cyclic (for any N); 1 is always a generator:
    $\{0, 1, 2, \ldots, N-1\}$

- $\mathbb{Z}_8$
  - Is 3 a *generator*?
    $\{0, 3, 6, 1, 4, 7, 2, 5\}$ - Yes!

Examples

- $\mathbb{Z}_N$
  - Cyclic (for any N); 1 is always a generator: $\{0, 1, 2, \dots, N-1\}$
- $\mathbb{Z}_8$
  - Is 3 a *generator*? $\{0, 3, 6, 1, 4, 7, 2, 5\}$ - Yes!
  - Is 2 a *generator*? $\{0, 2, 4, 6\}$ - No!

Example

- $\mathbb{Z}_{11}^*$
  - Is 3 a *generator*?
    $\{1, 3, 9, 5, 4\}$ - No!

Example

- $\mathbb{Z}_{11}^*$
  - Is 3 a *generator*?
    $\{1, 3, 9, 5, 4\}$ - No!
  - Is 2 a *generator*?
    $\{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$ - Yes!

Example

- $\mathbb{Z}_{11}^{*}$
  - Is 3 a *generator*?
    $\{1, 3, 9, 5, 4\}$ - No!
  - Is 2 a *generator*?
    $\{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$ - Yes!
  - Is 8 a *generator*?
    $\{1, 8, 9, 6, 4, 10, 3, 2, 5, 7\}$ - Yes!

Important examples

- **Theorem 11.1** Any group of prime order is cyclic, and every non-identity element is a generator.

- **Theorem 11.2** If p is prime, then $\mathbb{Z}_p^*$ is cyclic
  - **Note**: the order is $p-1$, which is not prime for $p > 3$

Uniform sampling

- Given cyclic group G of order q along with generator g, easy to sample a uniform $h \in G$
  - Choose uniform $x \in \{0, \ldots, q-1\}$: set $h := g^x$

Uniform sampling

- Given cyclic group G of order q along with generator g, easy to sample a uniform $h \in G$
  - Choose uniform $x \in \{0, \ldots, q-1\}$: set $h := g^x$

- Fix *cyclic* group G of order q, and *generator* g

Uniform sampling

- Given cyclic group G of order q along with generator g, easy to sample a uniform $h \in G$
  - Choose uniform $x \in \{0, \ldots, q-1\}$: set $h := g^x$

- Fix *cyclic* group G of order q, and *generator* g

- We know that $\{g^0, g^1, \ldots, g^{q-1}\} = G$
  - For every $h \in G$, there is a unique $x \in \mathbb{Z}_q$, s.t. $g^x = h$
  - Define $\log_g h$ to be this x – the *discrete logarithm* of h with respect to g (in the group G)

Examples

- In $\mathbb{Z}_{11}^*$
  - What is $\log_2 9$?

Examples

- In $\mathbb{Z}_{11}^*$
  - What is $\log_2 9$?
    - $\langle 2 \rangle = \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$, so $\log_2 9 = 6$

Examples

- In $\mathbb{Z}_{11}^*$
  - What is $\log_2 9$?
    - $\langle 2 \rangle = \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$, so $\log_2 9 = 6$
  - What is $\log_8 9$?

Examples

- In $\mathbb{Z}_{11}^*$
  - What is $\log_2 9$?
  - $\langle 2 \rangle = \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$, so $\log_2 9 = 6$
  - What is $\log_8 9$?
  - $\langle 8 \rangle = \{1, 8, 9, 6, 4, 10, 3, 2, 5, 7\}$, so $\log_8 9 = 2$

Examples

- In $\mathbb{Z}_{11}^*$
  - What is $\log_2 9$?
    - $\langle 2 \rangle = \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$, so $\log_2 9 = 6$
  - What is $\log_8 9$?
    - $\langle 8 \rangle = \{1, 8, 9, 6, 4, 10, 3, 2, 5, 7\}$, so $\log_8 9 = 2$

- In $\mathbb{Z}_{13}^*$
  - What is $\log_2 9$?

Examples

- In $\mathbb{Z}_{11}^*$
  - What is $\log_2 9$?
    - $\langle 2 \rangle = \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$, so $\log_2 9 = 6$
  - What is $\log_8 9$?
    - $\langle 8 \rangle = \{1, 8, 9, 6, 4, 10, 3, 2, 5, 7\}$, so $\log_8 9 = 2$
- In $\mathbb{Z}_{13}^*$
  - What is $\log_2 9$?
    - $\langle 2 \rangle = \{1, 2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7\}$, so $\log_2 9 = 8$

Discrete-logarithm problem (informal)

- DLog problem in G:

  Given *generator* g and element h, compute $\log_g h$

- DLog assumption in G:

  Solving the discrete log problem in G is hard

Discrete-logarithm problem (informal)

- DLog problem in G:
  Given *generator* g and element h, compute $\log_g h$

- DLog assumption in G:
  Solving the discrete log problem in G is hard

- In $\mathbb{Z}^*_{3092091139}$
  - What is $\log_2 1656755742$?

Discrete-logarithm problem

- Let $\mathcal{G}$ be a group-generation algorithm
  - On input 1^n, outputs a cyclic group G, its order q (with $\|q\| = n$), and a generator g

Discrete-logarithm problem

- Let $\mathcal{G}$ be a group-generation algorithm
  - On input 1^n, outputs a cyclic group G, its order q (with $\|q\| = n$), and a generator g
- For algorithm A, define *exp't* $Dlog_{A,\mathcal{G}}(n)$:
  - Compute $(G, q, g) \leftarrow \mathcal{G}(1^n)$
  - Choose uniform $h \in G$
  - Run $A(G, q, g, h)$ to get x
  - Experiment evaluates to 1 if $g^x = h$

Discrete-logarithm problem

- Let $\mathcal{G}$ be a group-generation algorithm
  - On input 1^n, outputs a cyclic group G, its order q (with $\|q\| = n$), and a generator g
- For algorithm A, define *exp't* $Dlog_{A,\mathcal{G}}(n)$:
  - Compute $(G, q, g) \leftarrow \mathcal{G}(1^n)$
  - Choose uniform $h \in G$
  - Run $A(G, q, g, h)$ to get x
  - Experiment evaluates to 1 if $g^x = h$
- **Definition 11.3** The *discrete-logarithm problem* is hard relative to $\mathcal{G}$ if for all PPT algorithms A,

$$\Pr[Dlog_{A,\mathcal{G}}(n) = 1] \leq negl(n)$$

Diffie-Hellman problems

- Fix cyclic group G and *generator* g
- Define $DH_g(h_1, h_2) = DH_g(g^x, g^y) = g^{xy}$

Diffie-Hellman problems

- Fix cyclic group G and *generator* g
- Define $DH_g(h_1, h_2) = DH_g(g^x, g^y) = g^{xy}$
- In $\mathbb{Z}_{11}^*$
  - $\langle 2 \rangle = \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$
  - So $DH_2(7, 5) = ?$

Diffie-Hellman problems

- Fix cyclic group G and *generator* g
- Define $DH_g(h_1, h_2) = DH_g(g^x, g^y) = g^{xy}$
- In $\mathbb{Z}_{11}^*$
  - $\langle 2 \rangle = \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$
  - So $DH_2(7, 5) = ?$
- In $\mathbb{Z}_{3092091139}^*$
  - What is $DH_2(1656755742, 938640663) = ?$
  - Is 1994993011 the answer, or is it just a random element of $\mathbb{Z}_{3092091139}^*$?

Diffie-Hellman assumptions

- *Computational* Diffie-Hellman (CDH) problem:
  - Given g, h_1, h_2, compute $DH_g(h_1, h_2)$

Diffie-Hellman assumptions

- *Computational* Diffie-Hellman (CDH) problem:
  – Given g, h_1, h_2, compute $DH_g(h_1, h_2)$

- *Decisional* Diffie-Hellman (DDH) problem:
  – Given g, h_1, h_2, distinguish $DH_g(h_1, h_2)$ from a uniform element of G

DDH problem

- Let $\mathcal{G}$ be a group-generation algorithm
  - On input 1^n, outputs a cyclic group G, its order q (with $\|q\| = n$), and a generator g

DDH problem

- Let $\mathcal{G}$ be a group-generation algorithm
  - On input 1^n, outputs a cyclic group G, its order q (with $\|q\| = n$), and a generator g

- The DDH problem is hard relative to $\mathcal{G}$ if for all PPT algorithm A:
  $|\Pr[A(G, q, g, g^x, g^y, g^z) = 1] - \Pr[A(G, q, g, g^x, g^y, g^{xy}) = 1]| \leq \epsilon(n)$

Relating the Diffie-Hellman problems

- Relative to $\mathcal{G}$
  - If the discrete-logarithm problem is easy, so is the CDH problem
  - If the CDH problem is easy, so is the DDH problem

Relating the Diffie-Hellman problems

- Relative to $\mathcal{G}$
  - If the discrete-logarithm problem is easy, so is the CDH problem
  - If the CDH problem is easy, so is the DDH problem

  - I.e., the DDH assumption is *stronger* than the CDH assumption
  - I.e., the CDH assumption is *stronger* than the dlog assumption

Group selection

- The *discrete logarithm* is not hard in all groups!

Group selection

- The *discrete logarithm* is not hard in all groups!
- Nevertheless, there are certain groups where the problem is believed to be hard

Group selection

- The *discrete logarithm* is not hard in all groups!
- Nevertheless, there are certain groups where the problem is believed to be hard
- For cryptographic applications, best to use *prime-order* groups
  - The *dlog* problem becomes easier if the order of the group has small prime factors
  - Prime-order groups have several nice features: e.g., every element except identity is a generator

Group selection

- The *discrete logarithm* is not hard in all groups!
- Nevertheless, there are certain groups where the problem is believed to be hard
- For cryptographic applications, best to use *prime-order* groups
  - The *dlog* problem becomes easier if the order of the group has small prime factors
  - Prime-order groups have several nice features: e.g., every element except identity is a generator
- Two common choices of groups

Group selection: choice 1

- *Prime-order* subgroup of $\mathbb{Z}_p^*$, p prime
  - E.g., $p = tq + 1$ for q prime
  - Take the subgroup of t^{th} powers, i.e.,
    $G = \{[x^t \bmod p] | x \in \mathbb{Z}_p^*\}$
    - This is a group
    - It has order $(p-1)/t = q$
    - Since q is prime, the group must be *cyclic*

Group selection: choice 1

- *Prime-order* subgroup of $\mathbb{Z}_p^*$, p prime
  - E.g., $p = tq + 1$ for q prime
  - Take the subgroup of t^{th} powers, i.e., $G = \{[x^t \bmod p] | x \in \mathbb{Z}_p^*\}$
    - This is a group
    - It has order $(p-1)/t = q$
    - Since q is prime, the group must be *cyclic*

  - Generalizations based on finite fields are also used

Group selection: choice 2

- *Prime-order* subgroup of an *elliptic curve* group
  - See book for details

Group selection: choice 2

- *Prime-order* subgroup of an *elliptic curve* group
  - See book for details

- We will describe algorithm in "abstract" groups
  - Can ignore details of the underlying group in the analysis
  - Can instantiate with any (appropriate) group for an implementation

Concrete parameters

- We have discussed two classes of cryptographic assumptions
  - *Factoring-based* (factoring, RSA assumptions)
  - *DLog-based* (DLog, CDH, and DDH assumptions)

Concrete parameters

- We have discussed two classes of cryptographic assumptions
  - *Factoring-based* (factoring, RSA assumptions)
  - *DLog-based* (DLog, CDH, and DDH assumptions)

- All these problems are (believed to be) "hard", i.e., to have no polynomial-time algorithms
  - But how hard are they, concretely?

Concrete parameters

- We have discussed two classes of cryptographic assumptions
  - *Factoring-based* (factoring, RSA assumptions)
  - *DLog-based* (DLog, CDH, and DDH assumptions)
- All these problems are (believed to be) "hard", i.e., to have no polynomial-time algorithms
  - But how hard are they, concretely?
- The goal here is to give an idea as to how parameters are calculated, and what relevant parameters are

Security

- Recall: For symmetric-key algorithms
  - Block cipher with n-bit key $\approx$ security against 2^n-time attacks
  - Hash functions with n-bit output $\approx$ security against $2^{n/2}$-time attacks

Security

- Recall: For symmetric-key algorithms
  - Block cipher with n-bit key $\approx$ security against 2^n-time attacks
  - Hash functions with n-bit output $\approx$ security against $2^{n/2}$-time attacks
- Factoring of a modulus of size 2^n (i.e., length n) using exhaustive search takes $2^{n/2}$ time

Security

- Recall: For symmetric-key algorithms
  - Block cipher with n-bit key $\approx$ security against 2^n-time attacks
  - Hash functions with n-bit output $\approx$ security against $2^{n/2}$-time attacks
- Factoring of a modulus of size 2^n (i.e., length n) using exhaustive search takes $2^{n/2}$ time
- Computing discrete logarithms in a group of order 2^n takes 2^n time

Security

- Recall: For symmetric-key algorithms
  - Block cipher with n-bit key $\approx$ security against 2^n-time attacks
  - Hash functions with n-bit output $\approx$ security against $2^{n/2}$-time attacks
- Factoring of a modulus of size 2^n (i.e., length n) using exhaustive search takes $2^{n/2}$ time
- Computing discrete logarithms in a group of order 2^n takes 2^n time
  - Are these the best algorithms possible?

Algorithms for factoring

- There exist algorithms factoring an integer N that run in much less than $2^{\|N\|/2}$ time

- Best known algorithm (asymptotically): *general number field sieve*
  - Running time (heuristic): $2^{O(\|N\|^{1/3} \log^{2/3} \|N\|)}$
  - Makes a huge difference in practice
  - Exact constant term also important!

Algorithms for dlog

- Two classes of algorithms:
  - Ones that work for *arbitrary* ("generic") groups
  - Ones that target specific groups
    - Recall that in some groups the problem is not even hard

- Best "generic" algorithms:
  - Time $2^{n/2}$ in a group of order $\approx 2^n$
  - This is known to be optimal for generic algorithms

Algorithms for dlog

- Best known algorithm for (subgroups of) $\mathbb{Z}_p^*$: *number field sieve*
  - Running time (heuristic): $2^{O(\|p\|^{1/3} \log^{2/3} \|p\|)}$

- For (appropriately chosen) elliptic-curve groups, nothing better than generic algorithms is known!
  - This is why elliptic-curve groups can allow for more-efficient cryptography

Choosing parameters

- As recommended by NIST (112-bit security):
  - *Factoring*: 2048-bit modulus
  - *Dlog*: order-q subgroup of $\mathbb{Z}_p^*$: $\|q\| = 224$, $\|p\| = 2024$
  - *Dlog*, elliptic-curve group of order q: $\|q\| = 224$

Choosing parameters

- As recommended by NIST (112-bit security):
  - *Factoring*: 2048-bit modulus
  - *Dlog*: order-q subgroup of $\mathbb{Z}_p^*$: $\|q\| = 224$, $\|p\| = 2024$
  - *Dlog*, elliptic-curve group of order q: $\|q\| = 224$

- Much longer than for symmetric-key algorithms!
  - Explains in part why public-key crypto is less efficient than symmetric-key crypto

Private-key cryptography

- *Private-key cryptography* allows two uses who *share a secret key* to establish a "secure channel"

Private-key cryptography

- *Private-key cryptography* allows two uses who *share a secret key* to establish a “secure channel”
- The need to share a *secret key* has several drawbacks

Private-key cryptography

- *Private-key cryptography* allows two uses who *share a secret key* to establish a "secure channel"
- The need to share a *secret key* has several drawbacks
- How do users share a key in the first place?
  - Need to share the key using a secure channel

Private-key cryptography

- *Private-key cryptography* allows two uses who *share a secret key* to establish a "secure channel"
- The need to share a *secret key* has several drawbacks
- How do users share a key in the first place?
  - Need to share the key using a secure channel
- This problem can be solved in some settings
  - E.g., physical proximity, trusted courier, ...
  - **Note**: this does not make *private-key cryptography* useless!

Private-key cryptography

- *Private-key cryptography* allows two uses who *share a secret key* to establish a "secure channel"
- The need to share a *secret key* has several drawbacks
- How do users share a key in the first place?
  - Need to share the key using a secure channel
- This problem can be solved in some settings
  - E.g., physical proximity, trusted courier, ...
  - **Note**: this does not make *private-key cryptography* useless!
- Can be difficult or expensive to solve in other settings

The key-management problem

- Imagine an organization with N employees, where each pair of employees might need to communicate securely

The key-management problem

- Imagine an organization with N employees, where each pair of employees might need to communicate securely
- Solution using *private-key cryptography*
  - Each user shares a key with all other users

  $\Rightarrow$ Each user must store/manage $N - 1$ secret keys!

  $\Rightarrow$ $O(N^2)$ keys overall!

Lack of support for "open systems"

- Say two users *who have no prior relationship* want to communicate securely
  - When would they ever have shared a key?

Lack of support for "open systems"

- Say two users *who have no prior relationship* want to communicate securely
  - When would they ever have shared a key?

- This happens all the time!
  - Customer sending credit-card data to merchant
  - Contacting a friend-of-a-friend on social media
  - Emailing a colleague
  - ...

“Classical” cryptography offers no solution

644 IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-22, NO. 6, NOVEMBER 1976

New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE

***Abstract*—Two kinds of contemporary developments in cryptography are examined. Widening applications of teleprocessing have given rise to a need for new types of cryptographic systems, which minimize the need for secure key distribution channels and supply the equivalent of a written signature. This paper suggests ways to solve these currently open problems. It also discusses how the theories of communication and computation are beginning to provide the tools to solve cryptographic problems of long standing.**

I. Introduction

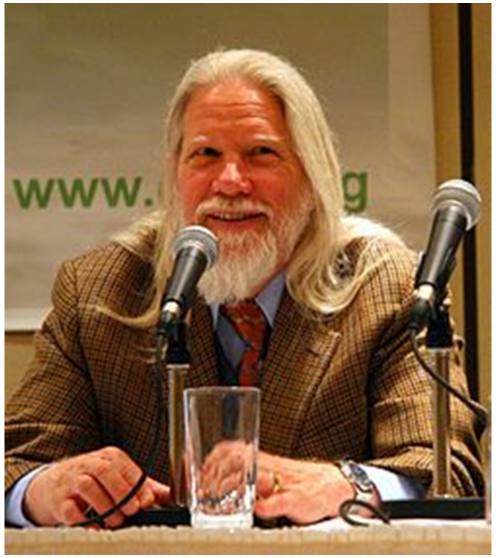
WE STAND TODAY on the brink of a revolution in cryptography. The development of cheap digital hardware has freed it from the design limitations of mechanical computing and brought the cost of high grade cryptographic devices down to where they can be used in such commercial applications as remote cash dispensers and computer terminals. In turn, such applications create a need for new types of cryptographic systems which minimize the necessity of secure key distribution channels and supply the equivalent of a written signature. At the same time, theoretical developments in information theory and computer science show promise of providing provably secure cryptosystems, changing this ancient art into a science.

The best known cryptographic problem is that of privacy: preventing the unauthorized extraction of information from communications over an insecure channel. In order to use cryptography to insure privacy, however, it is currently necessary for the communicating parties to share a key which is known to no one else. This is done by sending the key in advance over some secure channel such as private courier or registered mail. A private conversation between two people with no prior acquaintance is a common occurrence in business, however, and it is unrealistic to expect initial business contacts to be postponed long enough for keys to be transmitted by some physical means. The cost and delay imposed by this key distribution problem is a major barrier to the transfer of business communications to large teleprocessing networks.

Section III proposes two approaches to transmitting keying information over public (i.e., insecure) channels without compromising the security of the system. In a *public key cryptosystem* enciphering and deciphering are governed by distinct keys, E and D, such that computing D from E is computationally infeasible (e.g., requiring 10^{100} instructions). The enciphering key E can thus be publicly disclosed without compromising the deciphering key D. Each user of the network can, therefore, place his enciphering key in a public directory. This enables any user of the system to send a message to any other user enci-

- History (from 1976)
  - ◇ W. Diffie, M. Hellman, "New direction in cryptography", *IEEE Transactions on Information Theory, vol. 22, pp. 644-654, 1976.*

"We stand today on the brink of a revolution in cryptography."



Bailey W. Diffie

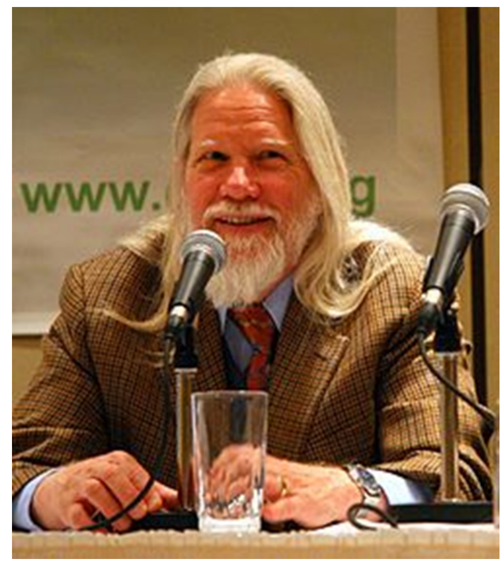
Martin E. Hellman

Cryptography History

- History (from 1976)
  - ◇ W. Diffie, M. Hellman, "New direction in cryptography", *IEEE Transactions on Information Theory, vol. 22, pp. 644-654, 1976.*

"We stand today on the brink of a revolution in cryptography."

2015 **Turing Award**



Bailey W. Diffie

Martin E. Hellman

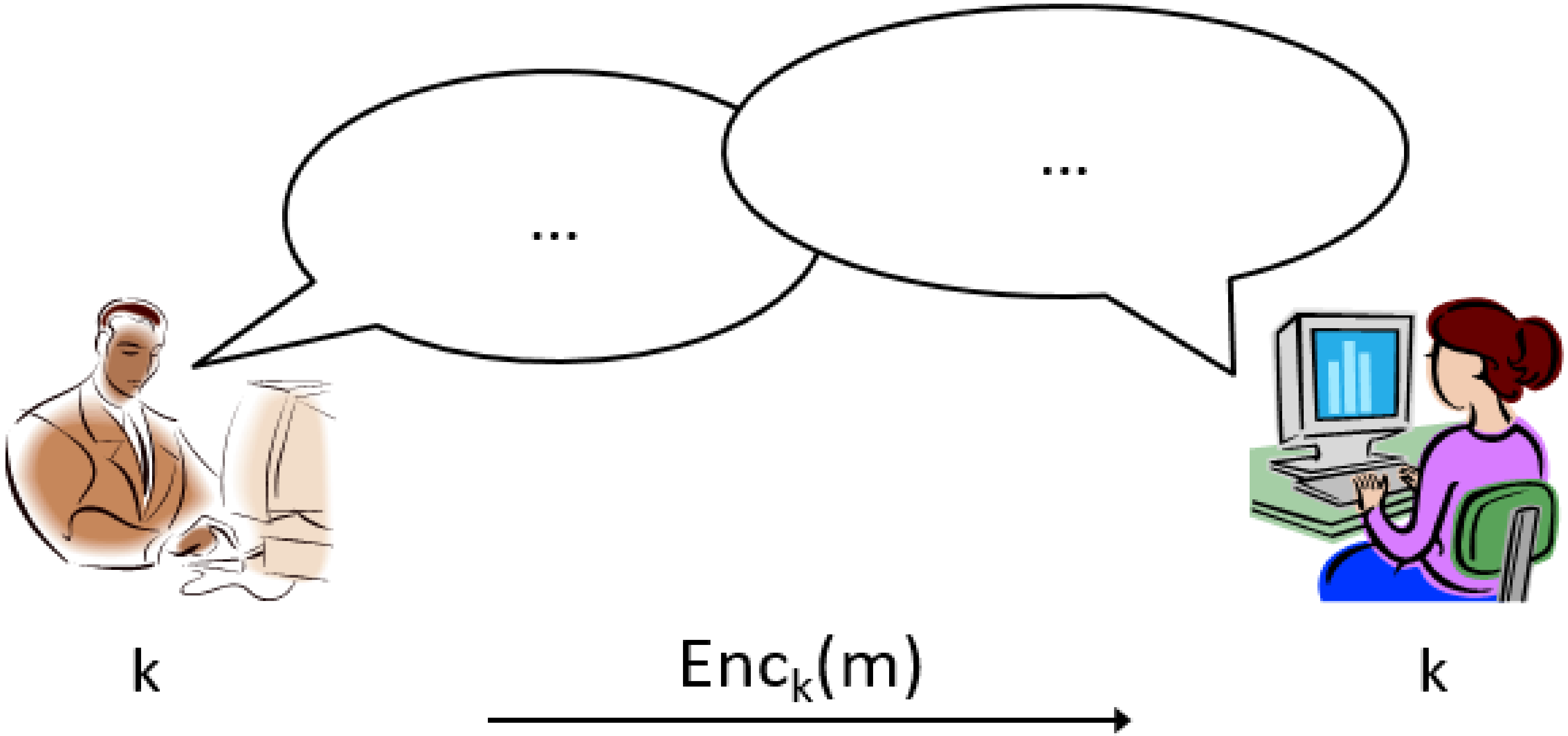
2015	Martin E. Hellman Whitfield Diffie	For fundamental contributions to **modern cryptography**. Diffie and Hellman's groundbreaking 1976 paper, "New Directions in Cryptography,"[39] introduced the ideas of public-key cryptography and digital signatures, which are the foundation for most regularly-used security protocols on the internet today.[40]

New directions

- Main ideas:
  - – Some problems exhibit *asymmetry* - easy to compute, but hard to invert (*factoring*, *RSA*, *group exponentiation*, ...)
  - – Use this *asymmetry* to enable two parties to agree on a shared secret key via public discussion
    - – *Key exchange*

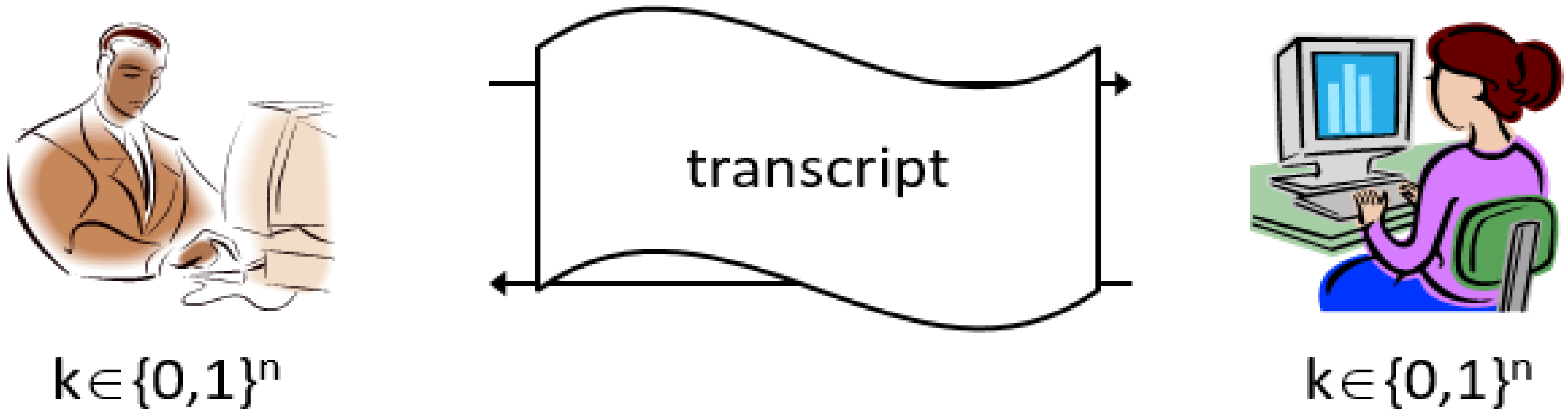
Key exchange

- *Secure* against an eavesdropper who sees everything!



More formally ...

- *Security goal*: even after observing the transcript, the shared key k should be *indistinguishable* from a uniform key



Next Lecture

- digital signature ...