CSE 484 (Winter 2010)

# Asymmetric Cryptography

Tadayoshi Kohno

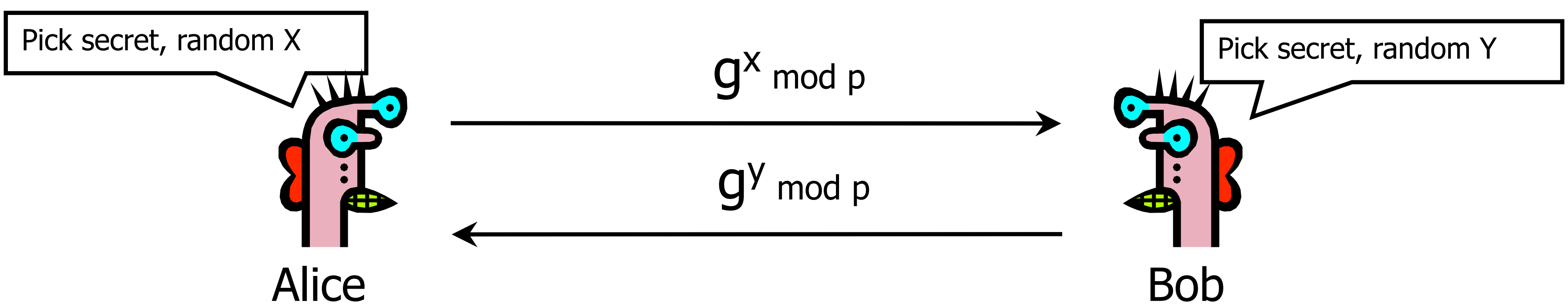
Thanks to Dan Boneh, Dieter Gollmann, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

# Goals for Today

- Asymmetric Cryptography

# Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: $p$ and $g$
  - $p$ is a large prime number, $g$ is a generator of $Z_p^*$
    - $Z_p^*=\{1, 2 \ldots p-1\}$; $\forall a \in Z_p^* \ \exists i$ such that $a=g^i \bmod p$
    - Modular arithmetic: numbers "wrap around" after they reach $p$




Compute $k=(g^y)^x=g^{xy} \bmod p$

Compute $k=(g^x)^y=g^{xy} \bmod p$

# Why Is Diffie-Hellman Secure?

- **Discrete Logarithm (DL) problem:**

  given $g^x$ mod p, it's hard to extract $x$

  - There is no known efficient algorithm for doing this
  - This is not enough for Diffie-Hellman to be secure!

- **Computational Diffie-Hellman (CDH) problem:**

  given $g^x$ and $g^y$, it's hard to compute $g^{xy}$ mod p

  - … unless you know x or y, in which case it's easy

- **Decisional Diffie-Hellman (DDH) problem:**

  given $g^x$ and $g^y$, it's hard to tell the difference between $g^{xy}$ mod p and $g^r$ mod p where r is random

# Properties of Diffie-Hellman

- Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against <u>passive</u> attackers
  - Eavesdropper can’t tell the difference between established key and a random value
  - Can use new key for symmetric cryptography
    - Approx. 1000 times faster than modular exponentiation
- Diffie-Hellman protocol (by itself) does not provide authentication

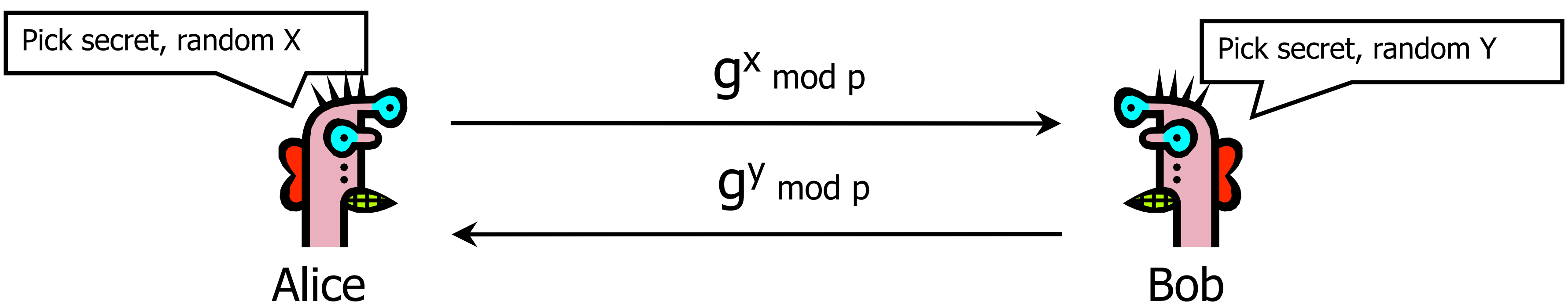
# Properties of Diffie-Hellman

- DDH: not true for integers mod p, but true for other groups
- DL problem in p can be broken down into DL problems for subgroups, if factorization of p-1 is known.
- Common recommendation:
  - Choose p = 2q+1 where q is also a large prime.
  - Pick a g that generates a subgroup of order q in $Z_p$*
  - (OK to not know all the details of why for this course.)

  - Hash output of DH key exchange to get the key

# Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: p and g
  - p, q are large prime numbers, p=2q+1, g a generator for the subgroup of order q
    - Modular arithmetic: numbers "wrap around" after they reach p




Compute $k=H((g^y)^x)=H(g^{xy}) \bmod p$ Compute $k=H((g^x)^y)=H(g^{xy}) \bmod p$

# Requirements for Public-Key Encryption

- Key generation: computationally easy to generate a pair (public key PK, private key SK)
  - Computationally infeasible to determine private key SK given only public key PK
- Encryption: given plaintext M and public key PK, easy to compute ciphertext $C=E_{PK}(M)$
- Decryption: given ciphertext $C=E_{PK}(M)$ and private key SK, easy to compute plaintext M
  - Infeasible to compute M from C without SK
  - Even infeasible to learn partial information about M
  - Trapdoor function: Decrypt(SK,Encrypt(PK,M))=M

# Some Number Theory Facts

- Euler totient function $\varphi(n)$ where $n \geq 1$ is the number of integers in the [1,n] interval that are relatively prime to n
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
- Euler's theorem:

  if $a \in Z_n^*$, then $a^{\varphi(n)} = 1 \bmod n$
- Special case: Fermat's Little Theorem

  if p is prime and $\gcd(a,p)=1$, then $a^{p-1} = 1 \bmod p$

# RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate large primes p, q
    - Say, 1024 bits each (need primality testing, too)
  - Compute $n=pq$ and $\varphi(n)=(p-1)(q-1)$
  - Choose small e, relatively prime to $\varphi(n)$
    - Typically, $e=3$ or $e=2^{16}+1=65537$ (why?)
  - Compute unique d such that $ed = 1 \bmod \varphi(n)$
  - Public key = (e,n); private key = (d,n)
- Encryption of m: $c = m^e \bmod n$
  - Modular exponentiation by repeated squaring
- Decryption of c: $c^d \bmod n = (m^e)^d \bmod n = m$

# Why RSA Decryption Works

- $e \cdot d = 1 \bmod \varphi(n)$
- Thus $e \cdot d = 1 + k \cdot \varphi(n) = 1 + k(p-1)(q-1)$ for some k

- Let m be any integer in $Z_n$
- If $\gcd(m,p)=1$, then $m^{ed} = m \bmod p$
  - By Fermat's Little Theorem, $m^{p-1} = 1 \bmod p$
  - Raise both sides to the power $k(q-1)$ and multiply by m
  - $m^{1+k(p-1)(q-1)} = m \bmod p$, thus $m^{ed} = m \bmod p$
  - By the same argument, $m^{ed} = m \bmod q$
- Since p and q are distinct primes and $p \cdot q = n$,
  $m^{ed} = m \bmod n$

# Why Is RSA Secure?

- RSA problem: given $n=pq$, $e$ such that $\gcd(e,(p-1)(q-1))=1$ and $c$, find $m$ such that $m^e=c \bmod n$
  - i.e., recover m from ciphertext c and public key (n,e) by taking $e^{th}$ root of c
  - There is no known efficient algorithm for doing this
- Factoring problem: given positive integer n, find primes $p_1, \ldots, p_k$ such that $n=p_1^{e_1}p_2^{e_2}\ldots p_k^{e_k}$
- If factoring is easy, then RSA problem is easy, but there is no known reduction from factoring to RSA
  - It may be possible to break RSA without factoring n

# Caveats

- e =3 is a common exponent
  - If $m < n^{1/3}$, then $c = m^3 < n$ and can just take the cube root of c to recover m
    - Even problems if "pad" m in some ways [Hastad]
  - Let $c_i = m^3 \bmod n_i$ - same message is encrypted to three people
    - Adversary can compute $m^3 \bmod n_1 n_2 n_3$ (using CRT)
    - Then take ordinary cube root to recover m
- Don't use RSA directly for privacy!

# Integrity in RSA Encryption

- Plain RSA does not provide integrity
  - Given encryptions of $m_1$ and $m_2$, attacker can create encryption of $m_1 \cdot m_2$
    - $(m_1^e) \cdot (m_2^e) \bmod n = (m_1 \cdot m_2)^e \bmod n$
  - Attacker can convert m into $m^k$ without decrypting
    - $(m_1^e)^k \bmod n = (m^k)^e \bmod n$
- In practice, OAEP is used: instead of encrypting M, encrypt $M \oplus G(r)\ ;\ r \oplus H(M \oplus G(r))$
  - r is random and fresh, G and H are hash functions
  - Resulting encryption is plaintext-aware: infeasible to compute a valid encryption without knowing plaintext
    - … if hash functions are "good" and RSA problem is hard

# OAEP (image from PKCS #1 v2.1)