CSE 484 : Computer Security and Privacy

Asymmetric Cryptography

Spring 2021

Tadayoshi Kohno

yoshi@cs

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, David Kohlbrenner, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Administrivia

- Lab 1 due on Friday
- HW2 out today, due two weeks from Friday (May 14)

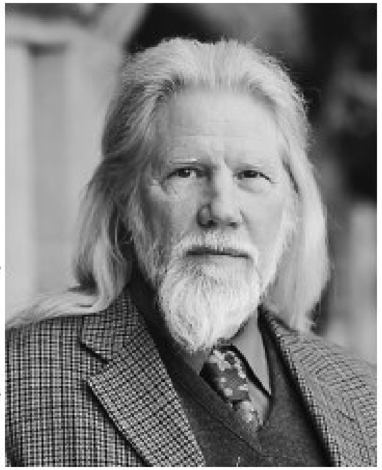
Session Key Establishment

Modular Arithmetic

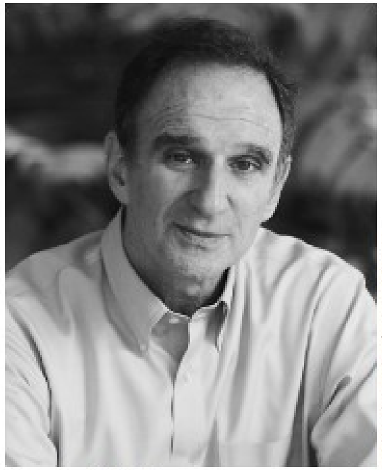
- Given g and prime p, compute: g^1 mod p, g^2 mod p, … g^{100} mod p
  - For p=11, g=10
    - 10^1 mod 11 = 10, 10^2 mod 11 = 1, 10^3 mod 11 = 10, …
    - Produces cyclic group {10, 1} (order=2)
  - For p=11, g=7
    - 7^1 mod 11 = 7, 7^2 mod 11 = 5, 7^3 mod 11 = 2, …
    - Produces cyclic group {7,5,2,3,10,4,6,9,8,1} (order = 10)
    - g=7 is a "generator" of Z_{11}^*

Diffie-Hellman Protocol (1976)

Diffie and Hellman Receive 2015 Turing Award



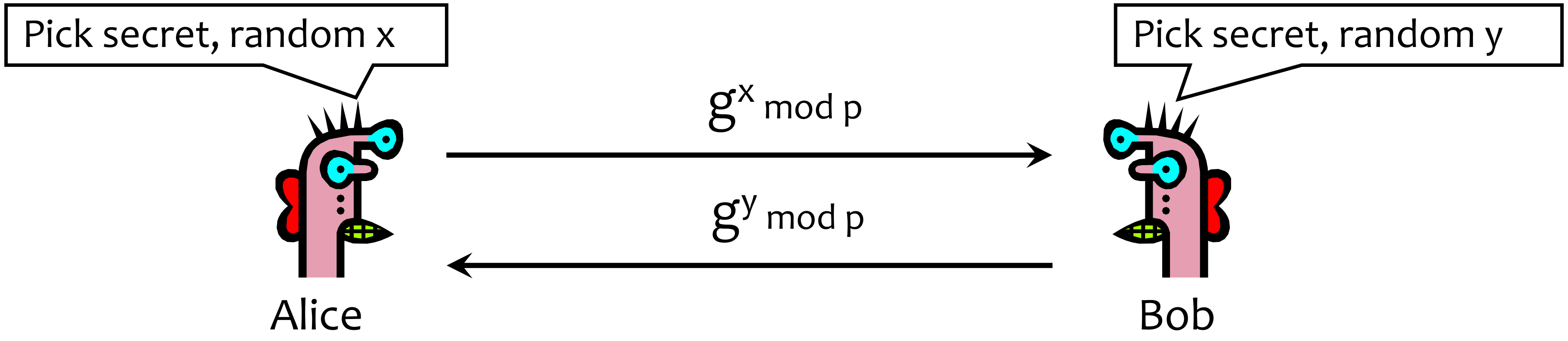
Whitfield Diffie



Martin E. Hellman

Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- <u>Public</u> info: p and g
  - p is a large prime, g is a **generator** of Z_p*
    - Z_p*={1, 2 … p-1}; a Z_p* i such that a=g^i mod p
    - <u>Modular arithmetic</u>: numbers "wrap around" after they reach p



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

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

Example Diffie Hellman Computation

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

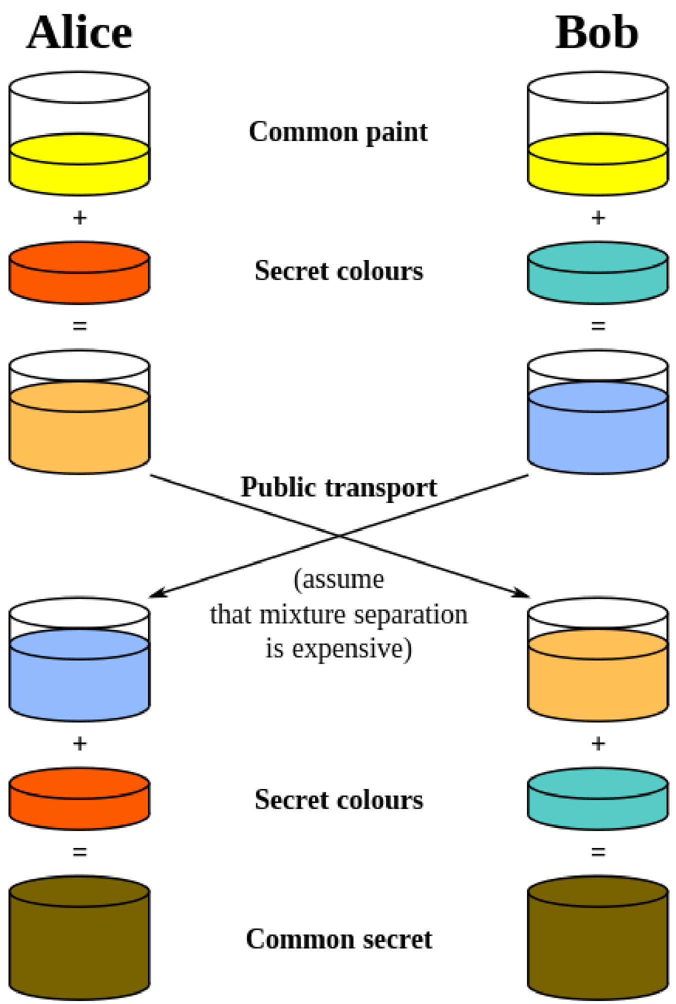
- Decisional Diffie-Hellman (DDH) problem:

  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

More on Diffie-Hellman Key Exchange

- Important Note:
  - We have discussed discrete logs modulo integers
  - Significant advantages in using elliptic curve groups
    - Groups with some similar mathematical properties (i.e., are “groups”) but have better security and performance (size) properties

Diffie-Hellman: Conceptually



Common paint: p and g

Secret colors: x and y

Send over public transport:
$g^x \bmod p$
$g^y \bmod p$

Common secret: $g^{xy} \bmod p$

[from Wikipedia]

Diffie-Hellman Caveats

- Assuming DDH problem is hard (depends on choice of parameters!), Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  - Common recommendation:
    - Choose p=2q+1, where q is also a large prime
    - Choose g that generates a subgroup of order q in Z_p*
    - DDH is hard in this group
  - Eavesdropper can't tell the difference between the established key and a random value
  - In practice, often hash $g^{xy} \bmod p$, and use the hash as the key
  - Can use the new key for symmetric cryptography
- Diffie-Hellman protocol (by itself) does not provide authentication (against active attackers)
  - Person in the middle attack (also called "man in the middle attack")

Example from Earlier

- Given g and prime p, compute: $g^1 \bmod p$, $g^2 \bmod p$, … $g^{100} \bmod p$
  - For p=11, g=10
    - $10^1 \bmod 11 = 10$, $10^2 \bmod 11 = 1$, $10^3 \bmod 11 = 10$, …
    - Produces cyclic group {10, 1} (order=2)
  - For p=11, g=7
    - $7^1 \bmod 11 = 7$, $7^2 \bmod 11 = 5$, $7^3 \bmod 11 = 2$, …
    - Produces cyclic group {7,5,2,3,10,4,6,9,8,1} (order = 10)
    - g=7 is a "generator" of Z_{11}^*
  - For p=11, g=3
    - $3^1 \bmod 11 = 3$, $3^2 \bmod 11 = 9$, $3^3 \bmod 11 = 5$, …
    - Produces cyclic group {3,9,5,4,1} (order = 5) (5 is a prime)
    - g=3 generates a group of prime order

Stepping Back: Asymmetric Crypto

- We've just seen session key establishment
  - Can then use shared key for symmetric crypto
- Next: public key encryption
  - For confidentiality
- Then: digital signatures
  - For authenticity

Requirements for Public Key Encryption

- Key generation: computationally easy to generate a pair (public key PK, private key SK)
- 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 learn anything about M from C without SK
  - Trapdoor function: Decrypt(SK,Encrypt(PK,M))=M

Some Number Theory Facts

- Euler totient function $\varphi(n)$ $(n \geq 1)$ 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
  - Easy to compute for primes: $\varphi(p) = p-1$
  - Note that $\varphi(ab) = \varphi(a)\,\varphi(b)$ if a & b are relatively prime

RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate large primes p, q
    - Say, 2048 bits each (need primality testing, too)
  - Compute **n**=pq and **φ(n)**=(p-1)(q-1)
  - Choose small **e**, relatively prime to $\varphi(n)$
    - Typically, **e=3** or **$e=2^{16}+1=65537$**
  - Compute unique **d** such that $ed \equiv 1 \bmod \varphi(n)$
    - Modular inverse: $d \equiv e^{-1} \bmod \varphi(n)$
  - Public key = (e,n); private key = (d,n)
- Encryption of m: $c = m^e \bmod n$
- Decryption of c: $c^d \bmod n = (m^e)^d \bmod n = m$

How to compute?

Why is RSA Secure?

- RSA problem: given c, n=pq, and e such that $\gcd(e, \varphi(n))=1$, find m such that $m^e = c \bmod n$
  - In other words, recover m from ciphertext c and public key (n,e) by taking e^{th} root of c modulo n
  - There is no known efficient algorithm for doing this *without* knowing p and q
- 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 (knowing factors means you can compute d = inverse of e mod (p-1)(q-1))
  - It may be possible to break RSA without factoring n -- but if it is, we don't know how

RSA Encryption Caveats

- Encrypted message needs to be interpreted as an integer less than n
- Don't use RSA **directly** for privacy – output is deterministic! Need to pre-process input somehow
- Plain RSA also does not provide integrity
  - Can tamper with encrypted messages

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

RSA OAEP

$M \oplus G(r) \,||\, r \oplus H(M \oplus G(r))$