CIS 5560

Cryptography
Lecture 13

Course website:
pratyushmishra.com/classes/cis-5560-s25/

Slides adapted from Dan Boneh and Vinod Vaikuntanathan

Announcements

- **Midterm coming up: 3/06 in class**
  - 70 minutes long, starts at 1:55PM
  - We will provide a cheat sheet with all the information (definitions, proof strategies, etc) you will need
  - 3/04 will be a review session in class.
  - 3/05 HW Party will be a review party

A.E. Theorems

Let (E,D) be CPA secure cipher and (S,V) secure MAC. Then:

1. **Encrypt-then-MAC**: always provides A.E.

2. **MAC-then-encrypt**: may be insecure against CCA attacks

   however: when (E,D) is rand-CTR mode or rand-CBC
   M-then-E provides A.E.

Number Theory Background

We will use a bit of number theory to construct:

- Key exchange protocols
- Digital signatures
- Public-key encryption

This module: crash course on relevant concepts

More info: read parts of Shoup's book referenced
at end of module

Notation

From here on:

- N denotes a positive integer.
- p denote a prime.

Notation: $\mathbb{Z}_N = \{0,1,\ldots,N-1\}$

Can do addition and multiplication modulo N

Greatest common divisor

<u>Def</u>: For all $x, y \in \mathbb{Z}$, $\gcd(x, y)$ is the <u>greatest common divisor</u> of x, y

Example: $\gcd(12,18) = 6$

<u>Fact</u>: for all $x, y \in \mathbb{Z}$, there exist $a, b \in \mathbb{Z}$ such that
$a \cdot x + b \cdot y = \gcd(x, y)$

a, b can be found efficiently using the extended Euclid algorithm

If $\gcd(x, y) = 1$, we say that x and y are **<u>relatively prime</u>**

Modular inversion

Over the rationals, inverse of 2 is ½ . What about $\mathbb{Z}_N$?

Def: The **inverse** of $x \in \mathbb{Z}_N$ is an element $y \in \mathbb{Z}_N$ s.t.

$$x \cdot y = 1 \mod N$$

y is denoted x^{-1}.

Example: let N be an odd integer. What is the inverse of $2 \mod N$?

Invertible elements

<u>Def:</u> $\mathbb{Z}_N^*$ = set of invertible elements in $\mathbb{Z}_N$

= $\{x \in \mathbb{Z}_N : \gcd(x, N) = 1\}$

Examples:

1. for prime p, $\mathbb{Z}_p^* := \{0, \ldots, p-1\}$
2. $\mathbb{Z}_{12}^* := \{1,5,7,11\}$

For $x \in \mathbb{Z}_N$, we can find x^{-1} using extended Euclid algorithm.

Today's Lecture

- More Number Theory
- Key Exchange
  - Merkle puzzles
  - Diffie—Hellman
    - Computational Diffie—Hellman Problem

Solving modular linear equations

Solve: $a \cdot x + b = 0$, where $a, x, b \in \mathbb{Z}_N$

Solution: $x = -b \cdot a^{-1} \mod N$

Find a^{-1} using extended Euclid algorithm.

Run time: $O(\log^2 N)$

Fermat's theorem (1640)

Thm: Let p be a prime. Then,

$$\forall x \in \mathbb{Z}_p^* : x^{p-1} = 1 \mod p$$

Example: p=5. $3^4 = 81 = 1$ in Z_5

How can we use this to compute inverses?

$$x \in \mathbb{Z}_p^* \Rightarrow x \cdot x^{p-2} = 1 \Rightarrow x^{-1} = x^{p-2}$$

(less efficient than Euclid)

The structure of $\mathbb{Z}_p^*$

<u>**Thm**</u> (Euler): $\mathbb{Z}_p^*$ is a **cyclic group**, that is

$$\exists g \in \mathbb{Z}_p^* \text{ such that } \{1, g, g^2, g^3, \ldots, g^{p-2}\} = \mathbb{Z}_p^*$$

g is called a <u>**generator**</u> of $\mathbb{Z}_p^*$

Example: $p = 7$. $\{1, 3, 3^2, 3^3, 3^4, 3^5\} = \{1, 3, 2, 6, 4, 5\} = \mathbb{Z}_7^*$

Not every elem. is a generator: $\{1, 2, 2^2, 2^3, 2^4, 2^5\} = \{1, 2, 4\}$

Order

For $g \in \mathbb{Z}_p^*$ the set $\{1, g, g^2, g^3, \ldots\}$ is called

the **group generated by g**, denoted $\langle g \rangle$

Def: the **order** of $g \in \mathbb{Z}_p^*$ is the size of $\langle g \rangle$

$\mathrm{ord}_p(g)$ = $|\langle g \rangle|$ = (smallest a > 0 s.t. $g^a = 1 \mod p$)

Examples: $\mathrm{ord}_7(3) = 6$; $\mathrm{ord}_7(2) = 3$; $\mathrm{ord}_7(1) = 1$

Thm (Lagrange): $\forall g \in (Z_p)^*$: **$\mathrm{ord}_p(g)$** divides p - 1

The Multiplicative Group $\mathbb{Z}_p^*$

$\mathbb{Z}_p^*$: $(\{1,\ldots,p-1\},$ group operation: $\bullet$ mod p)

- Computing the group operation is easy.
- Computing inverses is easy: Extended Euclid.
- Exponentiation (given $g \in \mathbb{Z}_p^*$ and $x \in \mathbb{Z}_{p-1}$, find g^x mod p) is easy: **Repeated Squaring Algorithm.**
-
- The discrete logarithm problem (given a generator g and $h \in \mathbb{Z}_p^*$, find $x \in \mathbb{Z}_{p-1}$ s.t. $h = g^x$ mod p) is **hard**, to the best of our knowledge!

The Discrete Log Assumption

The discrete logarithm problem is: given a generator g and $h \in \mathbb{Z}_p^*$, find $x \in \mathbb{Z}_{p-1}$ s.t. h = g^x mod p.

Distributions…

1. Is the discrete log problem hard for a random p? Could it be easy for some p?

2. Given p: is the problem hard for all generators g?

3. Given p and g: is the problem hard for all x?

Random Self-Reducibility of DLOG

Theorem: If there is an p.p.t. algorithm A s.t.

$$\Pr\left[A\left(p, g, g^x \bmod p\right) = x\right] > 1/\text{poly}(\log p)$$

for some p, random generator g of $\mathbb{Z}_p^*$, and random x in $\mathbb{Z}_{p-1}$,

then there is a p.p.t. algorithm B s.t.

$B\left(p, g, g^x \bmod p\right) = x$

for all g and x.

Proof: On the board.

Random Self-Reducibility of DLOG

Theorem: If there is an p.p.t. algorithm A s.t.

$$\Pr\left[A\left(p, g, g^x \bmod p\right) = x\right] > 1/\mathrm{poly}(\log p)$$

for some p, random generator g of $\mathbb{Z}_p^*$, and random x in $\mathbb{Z}_{p-1}$, then there is a p.p.t. algorithm B s.t.

$$B\left(p, g, g^x \bmod p\right) = x$$

for all g and x.

2. Given p: is the problem hard for all generators g?

   … as hard for any generator is it for a random one.

3. Given p and g: is the problem hard for all x?

   … as hard for any x is it for a random one.

Algorithms for Discrete Log (for General Groups)

- Baby Step-Giant Step algorithm: time —and space— $O(\sqrt{p})$.

- Pohlig-Hellman algorithm: time $O(\sqrt{q})$ where q is the largest prime factor of the order of group (e.g. $p-1$ in the case of Z_p^*). That is, there are dlog-easy primes.

The Discrete Log (DLOG) Assumption

W.r.t. a random prime: for every p.p.t. algorithm $\underline{A}$, there is a negligible function $\underline{\mu}$ s.t.

$$\Pr\left[\begin{array}{l} p \leftarrow PRIMES_n; g \leftarrow GEN\left(\mathbb{Z}_p^*\right); \\ x \leftarrow \mathbb{Z}_{p-1}: A\left(p, g, g^x \bmod p\right) = x \end{array}\right] = \mu(n)$$

Sophie-Germain Primes and Safe Primes

- A prime q is called a **Sophie-Germain** prime if $p = 2q + 1$ is also prime. In this case, q is called a **safe prime**.

- Safe primes are maximally hard for the Pohlig-Hellman algorithm.

- It is unknown if there are infinitely many safe primes, let alone that they are sufficiently dense. Yet, heuristically, about C/n^2 of n-bit integers seem to be safe primes (for some constant C).

The Discrete Log (DLOG) Assumption

(the "safe prime" version)

<u>W.r.t. a random safe prime</u>: for every p.p.t. algorithm $\underline{A,}$ there is a negligible function $\underline{\mu}$ s.t.

$$\Pr\left[\begin{array}{l} p \leftarrow SAFEPRIMES_n; g \leftarrow GEN\left(\mathbb{Z}_p^*\right); \\ x \leftarrow \mathbb{Z}_{p-1}: A\left(p, g, g^x \bmod p\right) = x \end{array}\right] = \mu(n)$$

One-way Permutation (Family)

$$F\left(p, g, x\right) = (p, g, g^x \text{ mod p})$$

$\mathcal{F}_n = \{F_{n,p,g}\}$ where $F_{n,p,g}(x) = (p, g, g^x \text{ mod p})$

Theorem: Under the discrete log assumption, F is a one-way permutation (resp. $\mathcal{F}_n$ is a one-way permutation family).

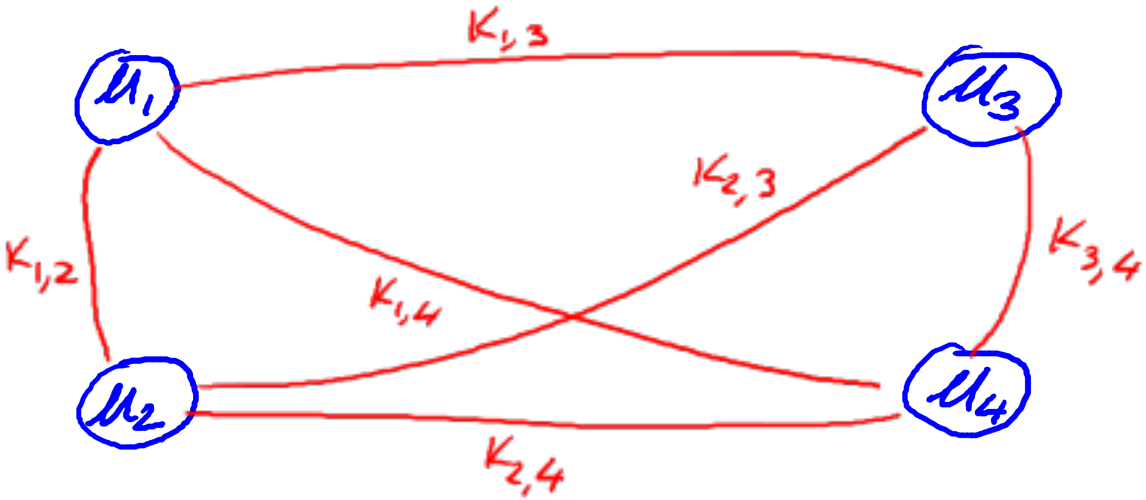
The Multiplicative Group $\mathbb{Z}_p^*$

$\mathbb{Z}_p^*$: $(\{1,\ldots,p-1\},$ group operation: $\bullet$ mod p)

- Computing the group operation is easy.
- Computing inverses is easy: Extended Euclid.
- Exponentiation (given $g \in \mathbb{Z}_p^*$ and $x \in \mathbb{Z}_{p-1}$, find g^x mod p) is easy: **Repeated Squaring Algorithm.**
-
- The discrete logarithm problem (given a generator g and $h \in \mathbb{Z}_p^*$, find $x \in \mathbb{Z}_{p-1}$ s.t. $h = g^x$ mod p) is **hard**, to the best of our knowledge!

Key management

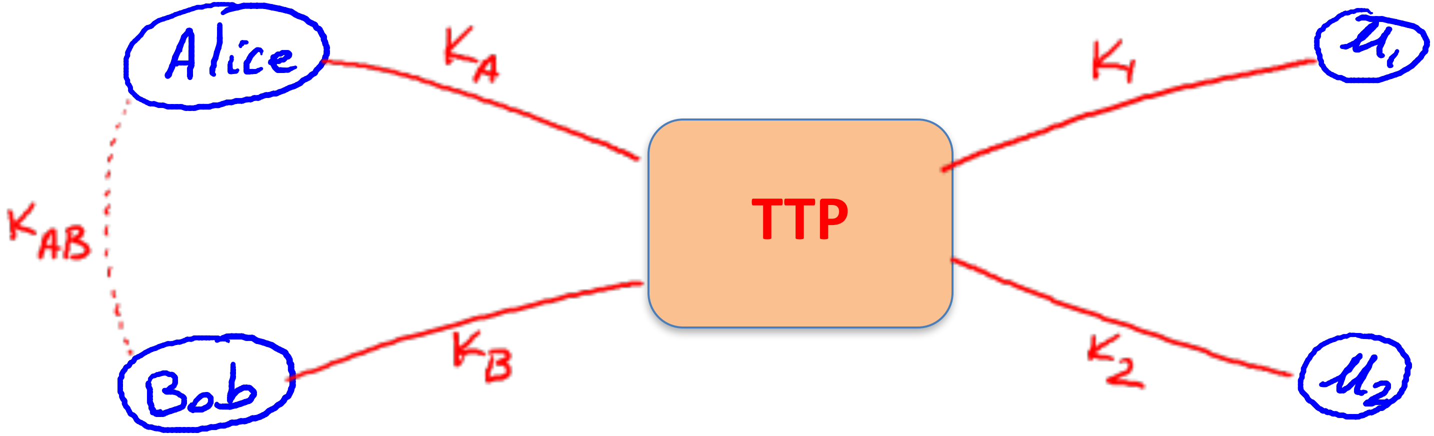
Problem: n users. Storing mutual secret keys is difficult



Total: O(n) keys per user

A better (?) solution

Online Trusted 3rd Party (TTP)



Every user only remembers one key.

Generating keys: a toy protocol

Alice wants a shared key with Bob. Eavesdropping security only.

(E,D) a CPA-secure cipher

Generating keys: a toy protocol

Alice wants a shared key with Bob. Eavesdropping security only.

Eavesdropper sees: $E(k_A,$ "A, B" $\|$ $k_{AB})$; $E(k_B,$ "A, B" $\|$ $k_{AB})$

(E,D) is CPA-secure $\Rightarrow$

eavesdropper learns nothing about k_{AB}

Note: TTP needed for every key exchange, knows all session keys.
(basis of Kerberos system)

Toy protocol: insecure against active attacks

Example: insecure against replay attacks

Attacker records session between Alice and merchant Bob

– For example a book order

Attacker replays session to Bob

– Bob thinks Alice is ordering another copy of book

Key question

Can we generate shared keys without an **online** trusted 3rd party?

Answer: yes!

Starting point of public-key cryptography:

- Merkle (1974), Diffie-Hellman (1976), RSA (1977)

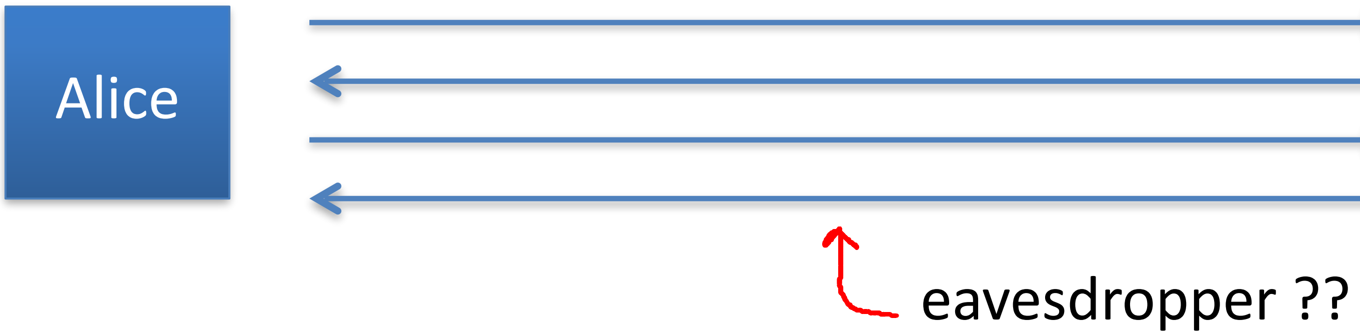
- More recently: ID-based enc. (BF 2001), Functional enc. (BSW 2011)

Basic key exchange:
Merkle Puzzles

Key exchange without an online TTP?

Goal: Alice and Bob want shared key, unknown to eavesdropper

- For now: security against eavesdropping only (no tampering)



Can this be done using generic symmetric crypto?

Merkle Puzzles (1974)

Answer: yes, but very inefficient

Main tool: puzzles

- Problems that can be solved with some effort
- Example: E(k,m) a symmetric cipher with $k \in \{0,1\}^{128}$
  - **puzzle(P) = E(P, "message")** where $P = 0^{96} \,||\, b_1 \ldots b_{32}$
  - Goal: find P by trying all 2^{32} possibilities

Merkle puzzles

Alice: prepare 2^{32} puzzles

- For $i=1, \ldots, 2^{32}$ choose random $\mathbf{P_i} \in \mathbf{\{0,1\}^{32}}$ and $\mathbf{x_i, k_i} \in \mathbf{\{0,1\}^{128}}$

  set puzzle$_i$ $\leftarrow$ E(0^{96} ll $\mathbf{P_i}$, **"Puzzle # x_i" ll k_i**)

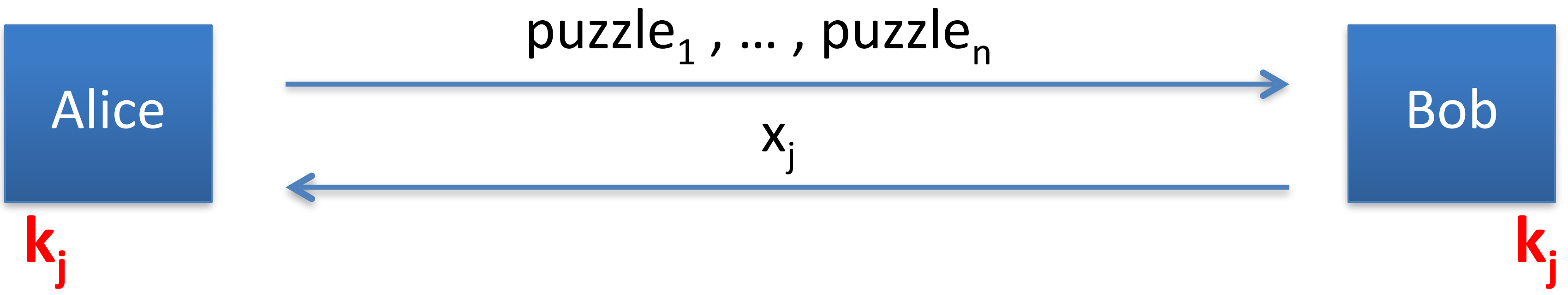
- Send puzzle$_1$, … , puzzle$_{2^{32}}$ to Bob

Bob: choose a random puzzle$_j$ and solve it. Obtain (x_j, k_j) .

- Send x_j to Alice

Alice: lookup puzzle with number x_j . Use k_j as shared secret

In a figure



Alice's work: O(n) (prepare n puzzles)

Bob's work: O(n) (solve one puzzle)

Eavesdropper's work: $O(n^2)$ (e.g. 2^{64} time)

Impossibility Result

Can we achieve a better gap using a general symmetric cipher?

Answer: unknown

But: roughly speaking,

quadratic gap is best possible if we treat cipher as

a black box oracle [IR’89, BM’09]

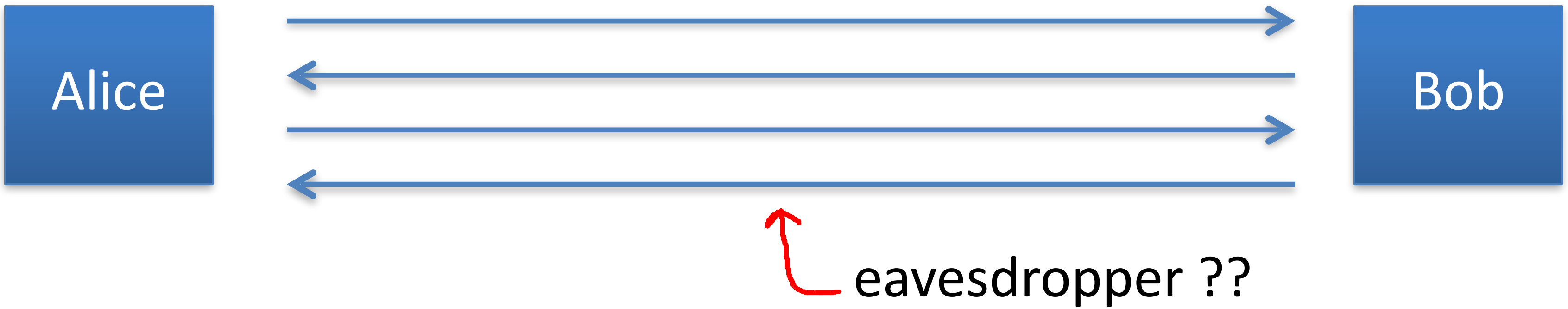
Better key exchange:

Diffie—Hellman

Key exchange without an online TTP?

Goal: Alice and Bob want shared secret, unknown to eavesdropper

- For now: security against eavesdropping only (no tampering)



Can this be done with an exponential gap?

The Diffie-Hellman protocol (informally)

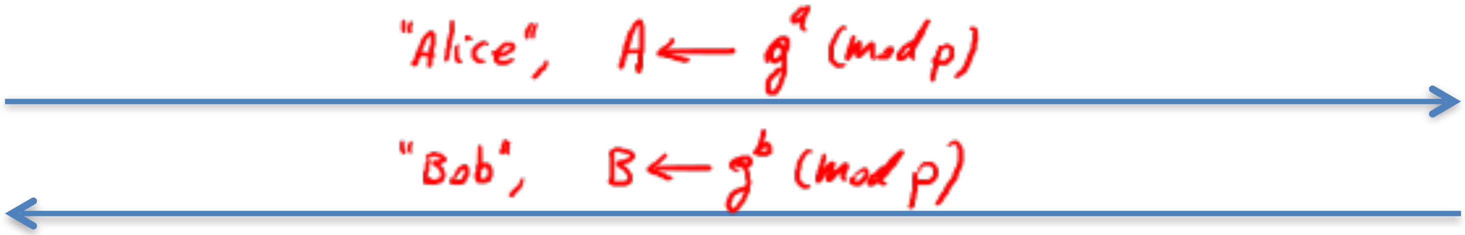
Fix a large prime p (e.g. 600 digits)

Fix generator g of $\mathbb{Z}_p^*$

Alice | **Bob**

choose random **a** in {1,…,p-1}

choose random **b** in {1,…,p-1}



$$\mathbf{B}^{\mathbf{a}} \pmod p = (g^b)^a = \mathbf{k}_{\mathbf{AB}} = \mathbf{g}^{\mathbf{ab}} \pmod p = (g^a)^b = \mathbf{A}^{\mathbf{b}} \pmod p$$

Security (much more on this later)

Eavesdropper sees: p, g, $A=g^a \pmod p$, and $B=g^b \pmod p$

Can she compute $g^{ab} \pmod p$??

More generally: define $DH_g(g^a, g^b) = g^{ab} \pmod p$

How hard is the DH function mod p?

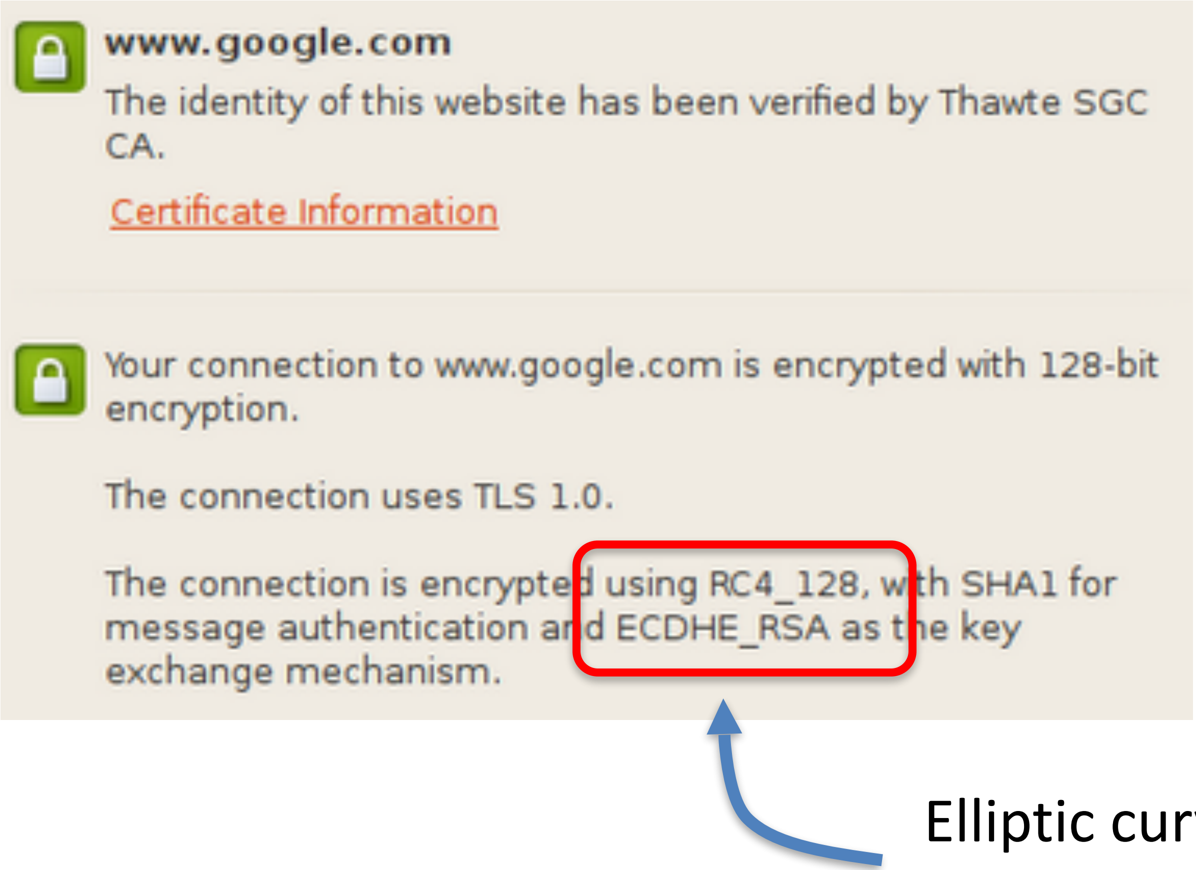
How hard is the DH function mod p?

Suppose prime p is n bits long.

Best known algorithm (GNFS): run time exp($\tilde{O}(\sqrt[3]{n})$)

cipher key size	modulus size	Elliptic Curve size
80 bits	1024 bits	160 bits
128 bits	3072 bits	256 bits
256 bits (AES)	**15360** bits	512 bits

As a result: slow transition away from (mod p) to elliptic curves

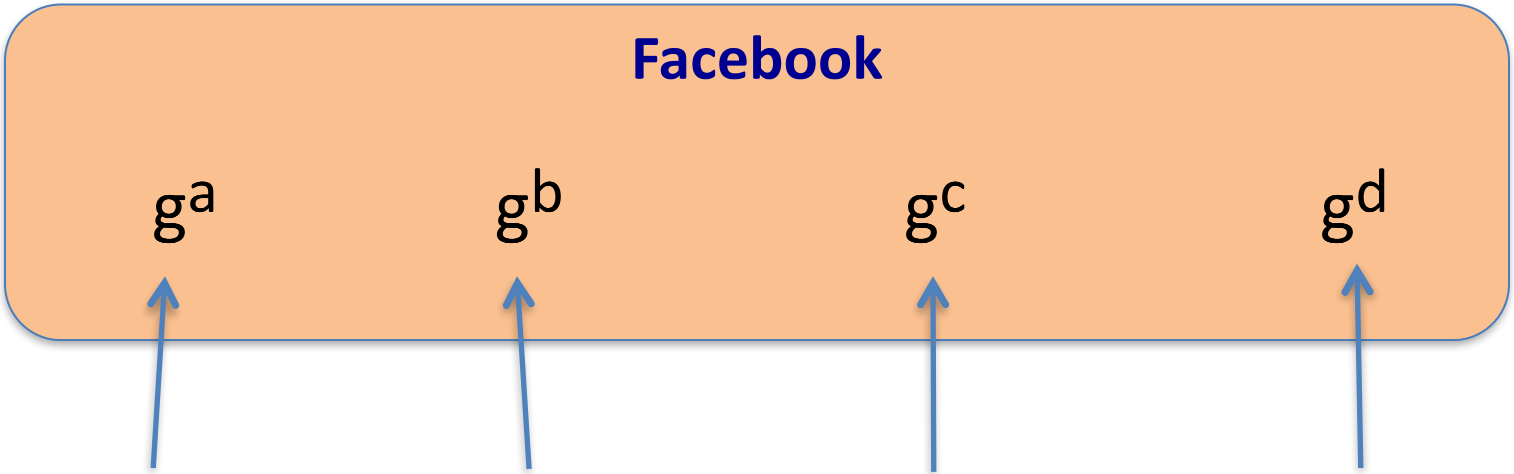


Elliptic curve Diffie-Hellman

Security against man-in-the-middle?

As described, the protocol is insecure against **active** attacks

Another look at DH



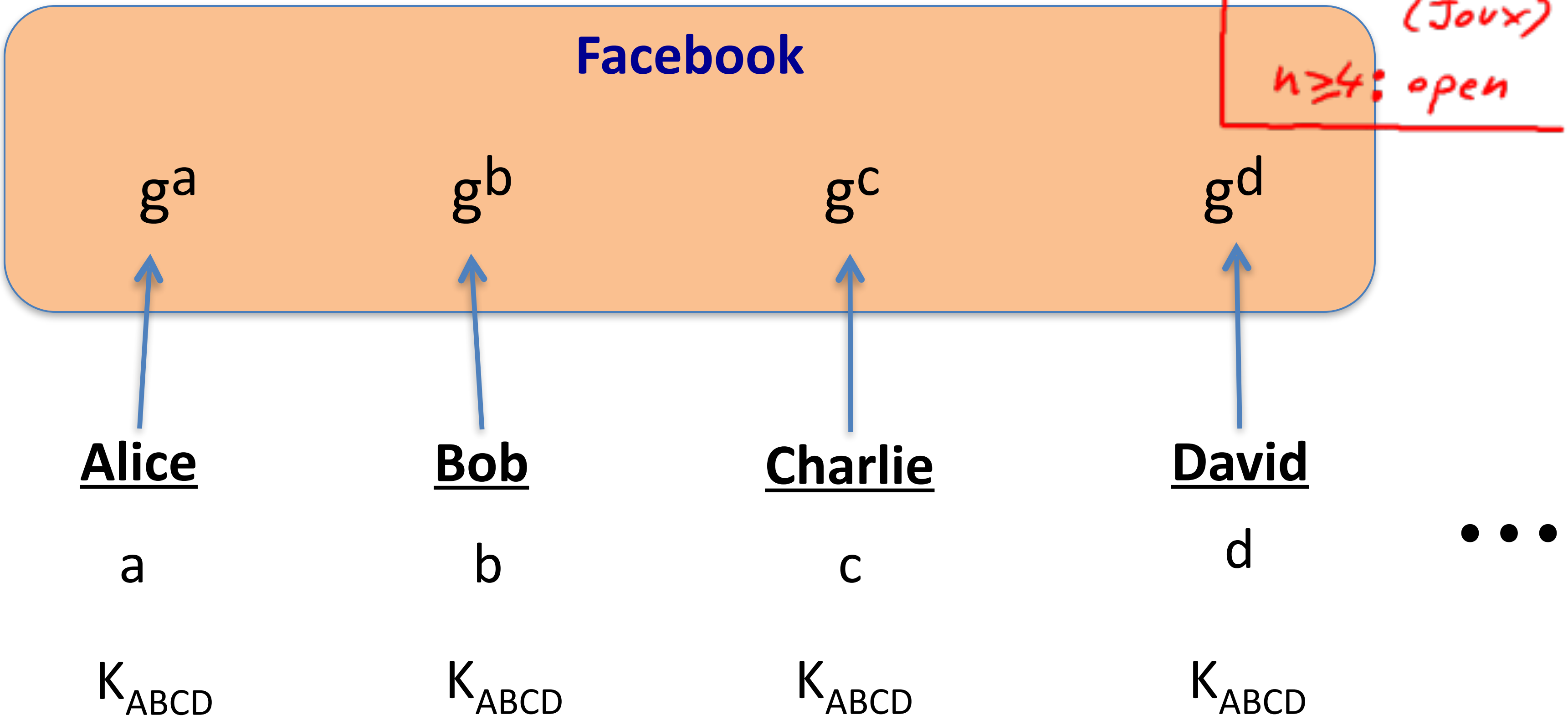
Alice **Bob** **Charlie** **David**

a b c d …

$K_{AC}=g^{ac}$ $K_{AC}=g^{ac}$

An open problem

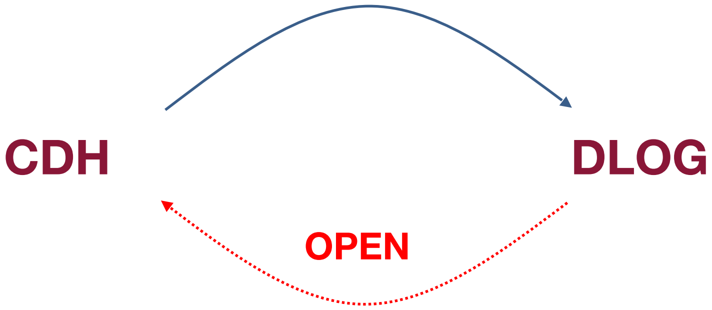
n=2 : DH
n=3 : Known (Joux)
n≥4 : open



Computational Diffie-Hellman (CDH) Assumption

W.r.t. a random prime: for every p.p.t. algorithm A, there is a negligible function μ s.t.

$$\Pr\left[\begin{array}{l} p \leftarrow PRIMES_n; g \leftarrow GEN\left(\mathbb{Z}_p^*\right); \\ x, y \leftarrow \mathbb{Z}_{p-1}:\ A\left(p, g, g^x, g^y\right) = g^{xy} \end{array}\right] = \mu(n)$$



Further readings

- Merkle Puzzles are Optimal,
  B. Barak, M. Mahmoody-Ghidary, Crypto ’09

- On formal models of key exchange (sections 7-9)
  V. Shoup, 1999

DLOG: more generally

Let $\mathbb{G}$ be a finite cyclic group and g a generator of $\mathbb{G}$

$$\mathbb{G} = \{ 1 , g , g^2 , g^3 , \ \ldots \ , g^{q-1} \}$$ (q is called the order of G)

<u>Def</u>: We say that **DLOG is hard in G** if for all efficient alg. A:

$$\Pr_{g \leftarrow G,\, x \leftarrow Z_q} \left[\, A(G, q, g, g^x) = x \, \right] < \text{negligible}$$

Example candidates:

(1) $(Z_p)^*$ for large p, (2) Elliptic curve groups mod p

Computing Dlog in $(Z_p)^*$ (n-bit prime p)

Best known algorithm (GNFS): run time exp($\tilde{O}(\sqrt[3]{n})$)

cipher key size	modulus size	Elliptic Curve group size
80 bits	1024 bits	160 bits
128 bits	3072 bits	256 bits
256 bits (AES)	**15360** bits	512 bits

As a result: slow transition away from (mod p) to elliptic curves