CIS 5560

Cryptography
Lecture 12

Course website:

pratyushmishra.com/classes/cis-5560-s24/

Slides adapted from Dan Boneh and Vinod Vaikuntanathan

Announcements

- **Final Exam May 10, 2024, 9-11AM, DRLB A2**
- **HW6 out later today, due in 2 weeks (Tuesday 3/12)**

Recap of last lecture

Goals

An **authenticated encryption** system (Gen, Enc, Dec) is a cipher where

As usual: $\text{Enc} : \mathscr{K} \times \mathscr{M} \to \mathscr{C}$

but $\text{Dec} : \mathscr{K} \times \mathscr{C} \to \mathscr{M} \cup \{\bot\}$

ciphertext is rejected

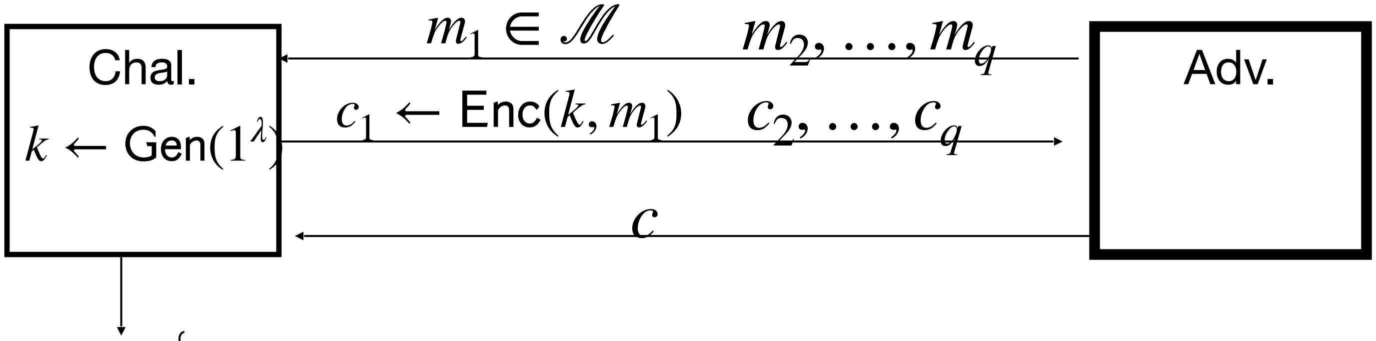
Security: the system must provide

- IND-CPA, and
- **ciphertext integrity:**

  attacker cannot create new ciphertexts that decrypt properly

Ciphertext integrity

Let (Gen, Enc, Dec) be a cipher with message space $\mathcal{M}$.



$$b \begin{cases} b = 1 & \text{if } \text{Dec}(k, c) \neq \bot \quad \text{and } c \notin \{c_1, \ldots, c_q\} \\ b = 0 & \text{otherwise} \end{cases}$$

Def: (Gen, Enc, Dec) has **ciphertext integrity** if for all PPT A:

$$\text{Adv}_{\text{CI}}[A] = \Pr[b = 1] = \text{negl}(\lambda)$$

Chosen ciphertext security

Adversary's power: both CPA and CCA

- Can obtain the encryption of arbitrary messages of his choice
- Can decrypt any ciphertext of his choice, other than challenge

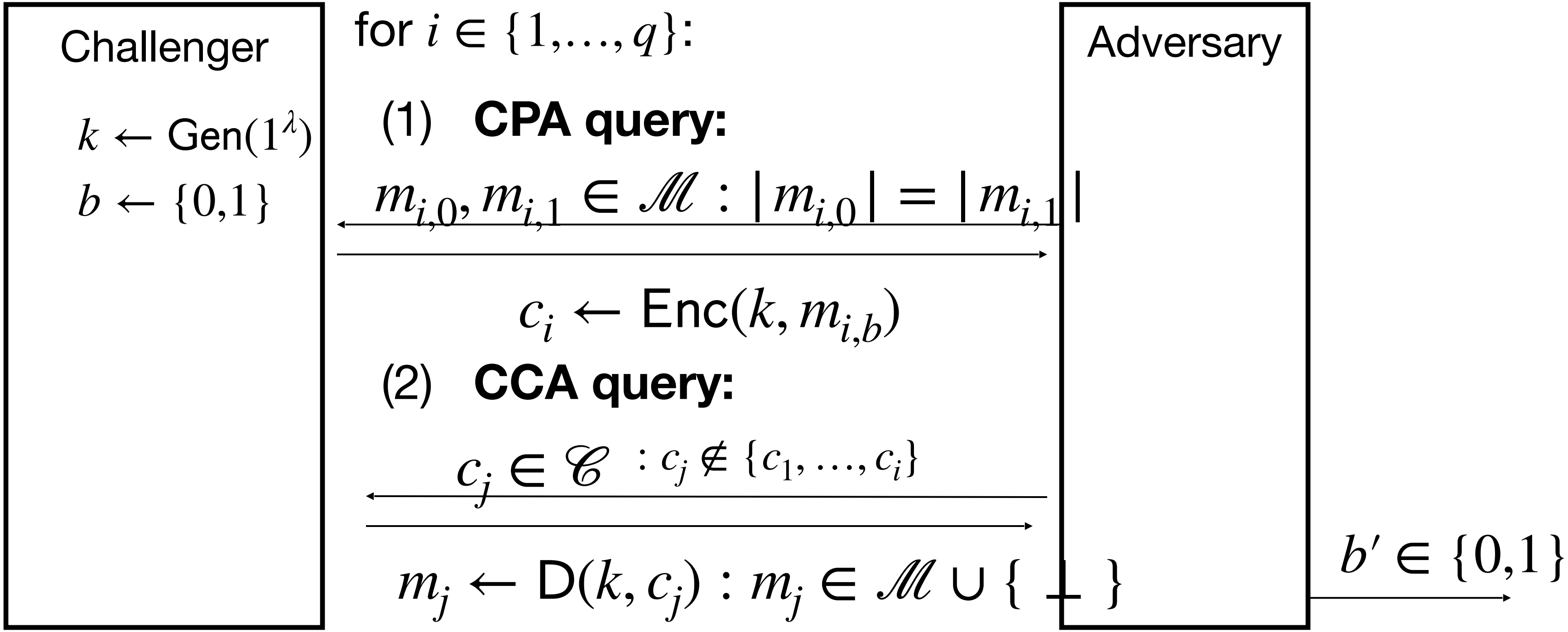
(conservative modeling of real life)

Adversary's goal:

Learn partial information about challenge plaintext

Chosen ciphertext security: definition

Let (Gen, Enc, Dec) be a cipher with message space $\mathscr{M}$



Authenticated enc. ⇒ CCA security

<u>Thm</u>: Let (E,D) be a cipher that provides AE.

Then (E,D) is CCA secure !

In particular, for any q-query eff. A there exist eff. B_1, B_2 s.t.

$$Adv_{CCA}[A,E] \leq 2q \cdot Adv_{CI}[B_1,E] + Adv_{CPA}[B_2,E]$$

Combining MAC and ENC (CCA)

Encryption key k_E. MAC key = k_M

Option 1: (SSL) msg m ⇨ msg m | tag t ⇨ $\text{Enc}(k_E, m || t)$

$\text{MAC}(k_M, m)$

always correct

Option 2: (IPsec) msg m ⇨ $\text{Enc}(k_E, m)$ ⇨ | tag t

$\text{MAC}(k_M, c)$

Option 3: (SSH) msg m ⇨ $\text{Enc}(k_E, m)$ ⇨ | tag t

$\text{MAC}(k_M, m)$

Security of Encrypt-then-MAC

Today's Lecture

- Number Theory refresher
  - Arithmetic modulo primes
  - Fermat's Little Theorem
  - Quadratic residuosity
  - Discrete Logarithms

  - Arithmetic modulo composites
  - Euler's Theorem
  - Factoring

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

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

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

Fact: 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 **relatively prime**

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 $2 \mod N$?

Modular inversion

Which elements have an inverse in $\mathbb{Z}_N$?

Lemma: $x \in \mathbb{Z}_N$ has an inverse if and only if $\gcd(x, N) = 1$

Proof:

$$\gcd(x, N) = 1 \implies \exists a, b : a \cdot x + b \cdot N = 1$$

$$\implies a \cdot x = 1 \mod N$$

$\gcd(x, N) \neq 1$ ⇒ ∀a: gcd(a·x, N) > 1 ⇒ a·x ≠ 1 in

Invertible elements

Def: $\mathbb{Z}_N^*$ = 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.

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)

Application: generating random primes

Suppose we want to generate a large random prime

say, prime p of length 1024 bits (i.e. p ≈ 2^{1024})

Step 1: sample $p \in [2^{1024}, 2^{1025} - 1]$

Step 2: test if $2^{p-1} = 1 \mod p$

If so, output p and stop. If not, goto step 1 .

Simple algorithm (not the best).

$\Pr[p \notin \text{PRIMES} \mid \text{test passes}] < 2^{-60}$

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 $\langle g \rangle$

$ord_p(g)$ = $|\langle g \rangle|$ = (smallest a > 0 s.t. $g^a = 1 \mod p$)

Examples: $ord_7(3) = 6$; $ord_7(2) = 3$; $ord_7(1) = 1$

Thm (Lagrange): $\forall g \in (Z_p)^*$: **$ord_p(g)$** divides p - 1

How to come up with a generator g

(1) **There are lots of generators:** $\approx 1/\log n$ fraction of $\mathbb{Z}_p^*$ are generators (where p is an n-bit prime).

(2) **Testing if g is a generator**:

<u>Theorem</u>: let $\underline{q_1, \ldots, q_k}$ be the prime factors of $\underline{p-1}$.
Then, g is a generator of $\underline{\mathbb{Z}_p^*}$ if and only if
$\underline{g^{(p-1)/q_i} \neq 1 \pmod{p}}$ for all i.

OPEN: Can you test if g is a generator without knowing the prime factorization of p-1?

OPEN: Deterministically come up with a generator?

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

$\mathbb{Z}_p^*$: $(\{1,\ldots,\mathrm{p}-1\},$ group operation: • 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.

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

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.

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)

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

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

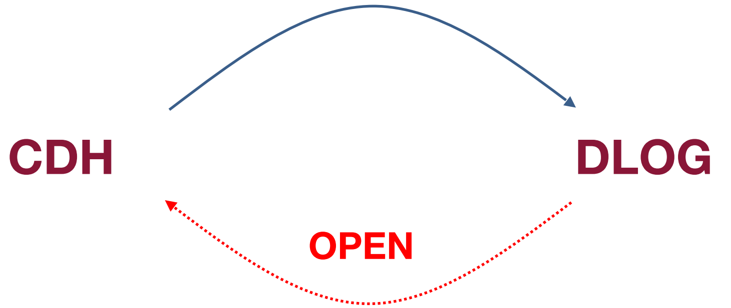
$$\mathscr{F}_n = \{F_{n,p,g}\} \text{ where } F_{n,p,g}(x) = (p, g, g^x \text{ mod p})$$

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

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.

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



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

An application: collision resistance

Choose a group G where Dlog is hard (e.g. $(Z_p)^*$ for large p)

Let $q = |G|$ be a prime. Choose generators g, h of G

For $x,y \in \{1,\ldots,q\}$ define $H(x,y) = g^x \cdot h^y$ in G

Lemma: finding collision for H(.,.) is as hard as computing $Dlog_g(h)$

Proof: Suppose we are given a collision $H(x_0,y_0) = H(x_1,y_1)$

then $g^{x_0} \cdot h^{y_0} = g^{x_1} \cdot h^{y_1} \Rightarrow g^{x_0-x_1} = h^{y_1-y_0} \Rightarrow h = g^{x_0-x_1/y_1-y_0}$

$\neq 0$

Further reading

- A Computational Introduction to Number Theory and Algebra,
  V. Shoup, 2008 (V2), Chapter 1-4, 11, 12

  Available at **//shoup.net/ntb/ntb-v2.pdf**