Purdue CS555: Cryptography
Lecture 8 Scribe Notes

Instructor: Hanshen Xiao
Teaching Assistant: Justin He

Fall 2025

Recap from Previous Lecture

- GGM Construction of Pseudorandom Functions
- Applications of PRFs:
  - Identity Identification
  - Message Authentication Codes (MACs)
  - Privacy vs. Integrity

Topics Covered in This Lecture

1. Review of MACs and Encrypt-then-MAC
2. Learning Theory Applications
3. Introduction to Public-Key Encryption
4. The Multiplicative Group $\mathbb{Z}_N^*$
5. Discrete Logarithm Problem
6. Diffie-Hellman Key Exchange
7. El Gamal Encryption

1 Review: Message Authentication Codes

1.1 MAC Definition

Definition 1 (Message Authentication Code)**.** *A MAC consists of three algorithms* (Gen, MAC, Ver)*:*

- $Gen(1^n)$*: Produces key* $k \leftarrow \mathcal{K}$
- $MAC(k, m)$*: Outputs tag* t *(may be deterministic)*
- $Ver(k, m, t)$*: Outputs Accept or Reject*

Correctness: $\Pr[\mathrm{Ver}(k, m, \mathrm{MAC}(k, m)) = \mathrm{Accept}] = 1$
Security: Hard to forge. Intuitively, it should be hard to produce a new valid pair (m', t').

1.2 Adversarial Model

Power of Adversary (Chosen Message Attack):

- Can observe many pairs $(m_i, \text{MAC}(k, m_i))$
- Has oracle access to $\text{MAC}(k, \cdot)$
- Can obtain tags for messages of choice

Security Levels:

- **Total break:** Adversary recovers key k
- **Universal break:** Adversary can generate valid tag for every message
- **Existential break:** Adversary can generate valid tag for *some* new message

Standard: Secure against existential forgery.

Definition 2 (EUF-CMA Security)**.** *A MAC is* ***Existentially Unforgeable under Chosen Message Attack*** *if for all p.p.t. A:*

$$\Pr[k \leftarrow \mathcal{K}; (m,t) \leftarrow A^{MAC(k,\cdot)}(1^n) : \mathit{Ver}(k,m,t) = 1 \wedge (m,t) \notin Q] = \mathit{negl}(n)$$

where Q is the set of query-response pairs obtained by A.

1.3 Why Encryption Isn't Enough

Problem: Encryption schemes are typically *malleable*.

Example 1 (One-Time Pad Malleability)**.** Given ciphertext $c = m \oplus k$, adversary can create $c' = m' \oplus k$ by computing:

$$c' = c \oplus m \oplus m' = (m \oplus k) \oplus m \oplus m' = m' \oplus k$$

Example 2 (PRF-Based Encryption Malleability)**.** Given ciphertext $c = (r, f_k(r) \oplus m)$, adversary can create:

$$c' = (r, f_k(r) \oplus m') = (r, (f_k(r) \oplus m) \oplus m \oplus m')$$

Key Lesson: *Privacy and integrity are very different goals.*

1.4 PRF-Based MAC Construction

Construction:

- $\text{Gen}(1^n)$: Generate PRF key k
- $\text{MAC}(k, m)$: Output $f_k(m)$
- $\text{Ver}(k, m, t)$: Accept if $f_k(m) = t$, reject otherwise

Security: Follows from unpredictability lemma for PRFs.

1.5 Replay Attacks

Issue: Adversary can resend old valid (m, tag) pairs.
Note: Our EUF-CMA definition doesn't rule this out.
Solutions:

1. **Timestamps:** $(m, T, \text{MAC}(k, m\|T))$ where T = current time
2. **Sequence numbers:** $(m, \text{seq}, \text{MAC}(k, m\|\text{seq}))$ (stateful)

1.6 Encrypt-then-MAC

Solution for Privacy + Integrity: Use two independent keys k, k'.

Encryption:

1. Encrypt: $c = (x, f_k(x) \oplus m)$
2. Tag: $\text{tag} = f_{k'}(c)$
3. Send: (c, tag)

Provides both IND-CPA security and EUF-CMA security.

2 Applications to Learning Theory

Theorem 1 (Kearns and Valiant 1994)**.** *Assuming PRFs exist, there are hypothesis classes that cannot be learned by polynomial-time algorithms.*

Intuition:

- **Learning Theory/ML:** Given labeled examples $(x_i, f(x_i))$ for unknown f, learn hypothesis $h \approx f$
- **Cryptography (PRFs):** Construct function families $\{f_k\}$ where it's hard to predict f_k on new inputs even with query access

Connection: If you can efficiently learn a PRF family, you can break its security. Therefore, PRFs give computationally hard-to-learn function classes.

2.1 More Negative Results:

- Intersections of halfspaces
- Agnostic learning of halfspaces
- Various concept classes in computational learning theory

3 Public-Key Encryption

Up to now, we've studied *symmetric* (secret-key) cryptography where both parties share a secret key. We now turn to *asymmetric* (public-key) cryptography.

3.1 Agenda

1. Key Agreement and Public-Key Encryption: Definition and Properties
2. Constructions:
   (a) Diffie-Hellman/El Gamal
   (b) Trapdoor Permutations (RSA)
   (c) Quadratic Residuosity/Goldwasser-Micali
   (d) Learning with Errors/Regev

4 The Multiplicative Group $\mathbb{Z}_N^*$

Definition 3. *The multiplicative group modulo N is:*

$$\mathbb{Z}_N^* = \{1 \leq x < N : \gcd(x, N) = 1\}$$

Theorem 2. *$\mathbb{Z}_N^*$ is a group under multiplication modulo N.*

Proof. **Inverses exist:** Since $\gcd(x, N) = 1$, by Bézout's identity there exist integers a and b such that:

$$ax + bN = 1$$

Therefore, $ax = 1 \pmod{N}$, so $a = x^{-1} \pmod{N}$. □

4.1 Euler's Totient Function

Definition 4. *The* ***order*** *of $\mathbb{Z}_N^*$ is given by Euler's totient function $\varphi(N)$:*

$$\varphi(P) = P - 1 \quad \textit{if } P \textit{ is prime}$$
$$\varphi(N) = (P-1)(Q-1) \quad \textit{if } N = PQ,\ P \neq Q \textit{ primes}$$
$$\varphi(N) = \prod P_i^{\alpha_i - 1}(P_i - 1) \quad \textit{if } N = \prod P_i^{\alpha_i}$$

Theorem 3 (Lagrange, Euler)**.** *For every $a \in \mathbb{Z}_N^*$:*

$$a^{\varphi(N)} = 1 \pmod{N}$$

4.2 Examples

- $\mathbb{Z}_2^* = \{1\}$
- $\mathbb{Z}_3^* = \{1, 2\}$
- $\mathbb{Z}_4^* = \{1, 3\}$
- $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$
- $\mathbb{Z}_6^* = \{1, 5\}$
- $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$

5 The Group $\mathbb{Z}_p^*$ for Prime p

$\mathbb{Z}_p^* = (\{1, 2, \ldots, p-1\}, \text{multiplication mod } p)$

5.1 Computational Properties

- **Group operation:** Easy to compute
- **Inverses:** Easy to compute via Extended Euclidean Algorithm
- **Exponentiation:** Given $g \in \mathbb{Z}_p^*$ and $x \in \mathbb{Z}_{p-1}$, computing $g^x \bmod p$ is easy via Repeated Squaring Algorithm
- **Discrete Logarithm:** Given generator g and $h \in \mathbb{Z}_p^*$, finding $x \in \mathbb{Z}_{p-1}$ such that $h = g^x \bmod p$ is *hard* (to the best of our knowledge)

6 The Discrete Logarithm Problem

Definition 5 (Discrete Logarithm Problem (DLOG))**.** *Given a generator g of $\mathbb{Z}_p^*$ and $h \in \mathbb{Z}_p^*$, find $x \in \mathbb{Z}_{p-1}$ such that:*

$$h = g^x \bmod p$$

6.1 Questions About Hardness

1. Is DLOG 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?

6.2 Random Self-Reducibility

Theorem 4 (Random Self-Reducibility of DLOG)**.** *If there exists a p.p.t. algorithm A such that:*

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

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

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

for all *g and x.*

Implication: DLOG is as hard for any generator as it is for a random one, and as hard for any x as it is for a random one.

6.3 Algorithms for Discrete Log

For General Groups:

- **Pohlig-Hellman algorithm:** Time $O(\sqrt{q})$ where q is the largest prime factor of the group order (e.g., $p-1$ for $\mathbb{Z}_p^*$)

  Implication: There are "DLOG-easy" primes (those with small prime factors)

- **Baby Step-Giant Step algorithm:** Time and space $O(\sqrt{p})$

6.4 Safe Primes and Sophie Germain Primes

Definition 6. *A prime q is called a* ***Sophie Germain prime*** *if* $p = 2q + 1$ *is also prime. In this case, p is called a* ***safe prime****.*

Properties:

- Safe primes are maximally hard for the Pohlig-Hellman algorithm
- Unknown if there are infinitely many safe primes
- Heuristically, about $\frac{C}{n^2}$ of n-bit integers are safe primes (for some constant C)

6.5 The Discrete Log Assumption

Definition 7 (DLOG Assumption w.r.t. Random Prime)**.** *For every p.p.t. algorithm A, there exists negligible* μ *such that:*

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

Definition 8 (DLOG Assumption w.r.t. Safe Primes)**.** *For every p.p.t. algorithm A, there exists negligible* μ *such that:*

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

6.6 One-Way Permutation from DLOG

Theorem 5. *Under the discrete log assumption,* $F(p, g, x) = (p, g, g^x \bmod p)$ *is a one-way permutation. More formally,* $\mathcal{F}_n = \{F_{n,p,g}\}$ *where* $F_{n,p,g}(x) = (p, g, g^x \bmod p)$ *is a one-way permutation family.*

7 Computational Diffie-Hellman Assumption

Definition 9 (CDH Assumption)**.** *For every p.p.t. algorithm A, there exists negligible* μ *such that:*

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

Relationship: CDH $\Leftarrow$ DLOG, but the converse is unknown (OPEN problem).

8 Diffie-Hellman Key Exchange

Setup: Public parameters are prime p and generator g of $\mathbb{Z}_p^*$.

Protocol:

Alice		**Bob**
Pick random $x \in \mathbb{Z}_{p-1}$		Pick random $y \in \mathbb{Z}_{p-1}$
	$\xrightarrow{g^x \bmod p}$	
	$\xleftarrow{g^y \bmod p}$	
Compute $K = (g^y)^x \bmod p$		Compute $K = (g^x)^y \bmod p$

Shared Secret: Both parties compute $K = g^{xy} \bmod p$.

9 El Gamal Encryption

9.1 Initial Construction

Key Generation: $\text{Gen}(1^n)$

1. Generate n-bit prime p and generator g of $\mathbb{Z}_p^*$
2. Choose random $x \in \mathbb{Z}_{p-1}$
3. Set $pk = (p, g, g^x)$ and $sk = x$

Encryption: $\text{Enc}(pk, m)$ where $m \in \mathbb{Z}_p^*$

1. Generate random $y \in \mathbb{Z}_{p-1}$
2. Output $(g^y, g^{xy} \cdot m)$

Decryption: $\text{Dec}(sk = x, c = (c_1, c_2))$

1. Compute $g^{xy} = c_1^x$
2. Compute $m = c_2 / g^{xy} = c_2 \cdot (c_1^x)^{-1}$

9.2 Security Issues

Problem: This construction is *not* IND-CPA secure.

Claim 1. *Given $p, g, g^x \bmod p$ and $g^y \bmod p$, an adversary can compute information about $g^{xy} \bmod p$. Specifically, the adversary can determine if $g^{xy} \bmod p$ is a square modulo p.*

Proof.

$$\begin{aligned} g^{xy} \text{ is a square mod } p &\Leftrightarrow xy \pmod{p-1} \text{ is even} \\ &\Leftrightarrow xy \text{ is even} \\ &\Leftrightarrow x \text{ is even or } y \text{ is even} \\ &\Leftrightarrow x \pmod{p-1} \text{ is even or } y \pmod{p-1} \text{ is even} \\ &\Leftrightarrow g^x \bmod p \text{ or } g^y \bmod p \text{ is a square} \end{aligned}$$

This can be checked in polynomial time. □

Corollary 1. *Given $g^{xy} \cdot m \bmod p$, the adversary can determine whether m is a square modulo p, violating IND-CPA security.*

9.3 Background: Squares Modulo p

Definition 10. *Let p be prime. $x \in \mathbb{Z}_p^*$ is a **square modulo** p (also called a **quadratic residue**) if there exists $y \in \mathbb{Z}_p^*$ such that:*

$$x = y^2 \bmod p$$

Theorem 6. *Exactly half of $\mathbb{Z}_p^*$ are squares modulo p.*

9.3.1 Characterization via Discrete Logarithm

Claim 2. *Fix any generator g. Then $x \in \mathbb{Z}_p^*$ is a square if and only if $DLOG_g(x) \bmod (p-1)$ is even.*

Proof. **(If):** If $x = g^a \bmod p$ and a is even, then $(g^{a/2})^2 = g^a = x \bmod p$.
(Only if): If $x = g^a = (g^b)^2 \bmod p$, then $a = 2b \pmod{p-1}$. So a is even. □

9.3.2 Efficient Characterization

Claim 3. *$x \bmod p$ is a square if and only if $x^{(p-1)/2} = 1 \bmod p$.*

Proof. **(If):** If $x = y^2 \bmod p$, then:

$$x^{(p-1)/2} = (y^2)^{(p-1)/2} = y^{p-1} = 1 \bmod p$$

by Fermat's Little Theorem.
(Only if): Show that the discrete log of x must be even, and therefore by previous claim, x is a square. □

9.3.3 Computing Square Roots

For primes $p \equiv 3 \pmod 4$, computing square roots is easy:

Claim 4. *If $p \equiv 3 \pmod 4$, the square roots of $x \bmod p$ are $\pm x^{(p+1)/4}$.*

Proof.

$$(\pm x^{(p+1)/4})^2 = x^{(p+1)/2} = x \cdot x^{(p-1)/2} = x \cdot 1 = x \bmod p$$

□

9.4 The Fix: Working Over Prime-Order Subgroup

Lesson: Best to work over a group of prime order. Such groups have no non-trivial subgroups.
Danger: Working with groups that have non-trivial subgroups (like the subgroup of squares modulo p).
Solution: Use a safe prime $p = 2q + 1$ where q is prime. The group of quadratic residues modulo p (denoted QR_p) has prime order $q = \frac{p-1}{2}$.

9.5 Corrected El Gamal Construction

Key Generation: $\text{Gen}(1^n)$

1. Generate n-bit safe prime $p = 2q + 1$
2. Generate generator g of $\mathbb{Z}_p^*$
3. Let $h = g^2 \bmod p$ be a generator of QR_p
4. Choose random $x \in \mathbb{Z}_q$
5. Set $pk = (p, h, h^x)$ and $sk = x$

Encryption: $\text{Enc}(pk, m)$ where $m \in QR_p$

1. Generate random $y \in \mathbb{Z}_q$
2. Output $(h^y, h^{xy} \cdot m)$

Decryption: $\text{Dec}(sk = x, c)$

1. Compute h^{xy} using $(h^y)^x$
2. Divide second component to retrieve m

10 Decisional Diffie-Hellman Assumption

Definition 11 (DDH Assumption)**.** *It is hard to distinguish between g^{xy} and a uniformly random group element, given g, g^x, g^y.*

Formally, the following two distributions are computationally indistinguishable:

$$(g, g^x, g^y, g^{xy}) \approx (g, g^x, g^y, u)$$

where x, y are random and u is uniformly random in the group.

Theorem 7. *El Gamal encryption (corrected version) is IND-CPA secure under the DDH assumption on the given group.*

11 Implementation Details

11.1 How to Generate Prime p

Prime Number Theorem: Approximately $\frac{1}{n}$ fraction of n-bit numbers are prime.
Primality Tests:

- Miller-Rabin (1976, 1980): Probabilistic polynomial-time test
- Agrawal-Kayal-Saxena (2002): Deterministic polynomial-time test

OPEN Problem: Deterministically generate an n-bit prime (not just test)?
Exciting New Result: Pseudo-deterministic polynomial-time algorithm (announced at TOC Colloquium, Oct 3).

11.2 How to Generate Generator g

Density: Approximately $\frac{1}{\log n}$ fraction of $\mathbb{Z}_p^*$ are generators (for n-bit prime p).

Theorem 8 (Testing if g is a generator)**.** *Let $q_1, \ldots, q_k$ be the prime factors of $p-1$. Then g is a generator of $\mathbb{Z}_p^*$ if and only if:*

$$g^{(p-1)/q_i} \neq 1 \pmod{p} \quad \textit{for all } i$$

OPEN Problems:

- Can you test if g is a generator without knowing the prime factorization of $p-1$?
- Deterministically generate a generator?

11.3 Practical Algorithm

1. Pick random safe prime p (so $p-1=2q$ and factorization is known)
2. Pick random element of $\mathbb{Z}_p^*$ and test if it's a generator using the theorem
3. Repeat step 2 until hitting a generator

12 Which Group to Use?

12.1 Option 1: Quadratic Residues QR_p

Parameters: Safe prime $p = 2q+1$ where q is prime. Order of group is q.
Discrete Log Complexity: Can be broken in sub-exponential time:

$$2^{\sqrt{\log p \log\log p}}$$

Better than poly(p) but worse than poly($\log p$).

12.2 Option 2: Elliptic Curve Groups

Definition: Set of solutions (x, y) to the equation:

$$y^2 = x^3 + ax + b \pmod{p}$$

together with a special group addition law.

Discrete Log Complexity: Best known algorithm runs in $O(\sqrt{p})$ time.

Advantages:

- Much smaller keys needed
- 160-bit p suffices for "80-bit security"
- More efficient than traditional DH