Principles of Information Security

Problem Set - I | Answer Book

Problem 1: Security Through Obscurity

Scenario: A company designs a proprietary encryption system for internal communications. Initially, both the algorithm and secret key are known only to the company. After several years, the algorithm documentation is leaked publicly, but the secret keys remain unknown.

Part (a): Design Assumption Flaw

Question: Assume that the system faces a successful attack shortly after the algorithm is revealed, even though the keys are still secret. What does this suggest about the original design assumptions?

Answer:

The successful attack after the algorithm leak reveals a critical flaw in the system's design: the company relied on **security through obscurity**.

This means the system's security depended not just on the secrecy of the key, but also on keeping the algorithm itself secret. This is a fundamentally weak approach because:

1. **The algorithm was not robust:** A well-designed encryption algorithm should remain secure even when its internal workings are fully known. The only secret should be the key.
2. **False sense of security:** The company assumed that hiding the algorithm provided an additional layer of protection. Once this "layer" was removed (via the leak), the system collapsed—proving it was never truly secure.
3. **Violation of Kerckhoffs's Principle:** This principle states that a cryptographic system should be secure even if everything about the system, except the key, is public knowledge. The company's system clearly violated this.

> **Key Takeaway:** If knowing the algorithm is enough to break the system (even without the key), then the algorithm itself is flawed and the design assumptions were incorrect.

Part (b): General Design Guideline

Question: State a general design guideline for cryptographic systems regarding which components may be assumed public and which must remain secret.

Answer:

The fundamental guideline is **Kerckhoffs's Principle** (see Appendix E) (also known as Shannon's Maxim in its modern form):

"A cryptographic system should be secure even if everything about the system, except the key, is public knowledge."

This translates to the following design rules:

May Be Public	Must Remain Secret
• Encryption algorithm • Decryption algorithm • Protocol specifications • Implementation details • Mathematical foundations	• Secret keys • Private keys (in asymmetric systems) • Session keys • Key derivation seeds

Rationale:
- Algorithms can be reverse-engineered, leaked, or discovered over time
- Public algorithms undergo widespread scrutiny, leading to discovery and fixing of vulnerabilities
- Security concentrated in a small, manageable secret (the key) is easier to protect and rotate
- Keys can be changed easily; algorithms cannot be changed without massive overhaul

Part (c): Forward Secrecy vs. Backward Secrecy

Question: Explain and differentiate between forward and backward secrecy.

Answer:

These are properties that protect encrypted communications even if long-term secret keys are compromised.

1.3.1. Forward Secrecy (a.k.a. Perfect Forward Secrecy - PFS)

Definition: If a long-term secret key is compromised in the future, previously encrypted messages remain secure and cannot be decrypted.

How it works:
- Each session uses a unique, ephemeral session key
- Session keys are derived independently and deleted after use
- Compromising the long-term key doesn't reveal past session keys

Example: Alice and Bob communicate using TLS with Diffie-Hellman key exchange. Even if an attacker later obtains the server's private key, they cannot decrypt old recorded conversations because each session used a unique ephemeral key that no longer exists.

> **Forward Secrecy protects the PAST from future key compromise.**

1.3.2. Backward Secrecy (a.k.a. Future Secrecy)

Definition: If a session key or secret is compromised, future communications remain secure and cannot be decrypted.

How it works:
- Keys are updated or rotated regularly
- New keys are derived in a way that knowing the old key doesn't help compute the new one (one-way derivation)
- Often achieved through key ratcheting mechanisms

Example: In the Signal Protocol, after every message, the encryption key is "ratcheted" forward. If an attacker compromises the current key, they cannot decrypt future messages because new keys are derived using one-way functions.

> **Backward Secrecy protects the FUTURE from current key compromise.**

1.3.3. Comparison Table

Aspect	Forward Secrecy	Backward Secrecy
Protects	Past communications	Future communications
Threat	Future key compromise	Current key compromise
Mechanism	Ephemeral session keys	Key ratcheting / rotation
Key insight	Old keys are deleted	New keys are independent
Example	TLS with DHE/ECDHE	Signal Protocol

Practical Note: Modern secure messaging protocols (like Signal, WhatsApp's encryption) implement **both** forward and backward secrecy using a "double ratchet" algorithm, ensuring that compromise of any single key has limited impact on overall communication security.

◆ ◆ ◆

Problem 2: Perfect Pseudo-Random Generators

Task: Provide a definition for perfect pseudo-random generators $G : \{0,1\}^n \to \{0,1\}^{n+1}$. Furthermore, prove that such perfect PRGs do not exist.

Definition of a Perfect PRG

A **perfect pseudo-random generator** (PRG) would be a deterministic function:

$$G : \{0,1\}^n \to \{0,1\}^{n+1}$$

that satisfies the following property:

Perfect Indistinguishability: The output distribution of G is **identical** (not just computationally indistinguishable) to the uniform distribution over $\{0,1\}^{n+1}$.

Formally: For a seed s chosen uniformly at random from $\{0,1\}^n$:

$$G(s) \equiv U_{n+1}$$

where U_{n+1} denotes the uniform distribution over $\{0,1\}^{n+1}$.

In other words, no algorithm (even with unlimited computational power) can distinguish between:
- A string sampled from $G(U_n)$ — output of the PRG on a random seed
- A string sampled uniformly from $\{0,1\}^{n+1}$

Proof: Perfect PRGs Cannot Exist

We prove this using a **counting argument** (pigeonhole principle).

2.2.1. Setup

- **Domain (seeds):** $\{0,1\}^n$ has exactly 2^n elements
- **Codomain (outputs):** $\{0,1\}^{n+1}$ has exactly $2^{n+1} = 2 \cdot 2^n$ elements
- G is a **deterministic** function

2.2.2. The Counting Argument

Since G is a function from a set of size 2^n to a set of size 2^{n+1}:

> **Key Observation:** G can produce **at most** 2^n distinct outputs (one for each possible seed).

But the codomain has 2^{n+1} possible strings. Therefore:

$$|\text{Image}(G)| \leq 2^n < 2^{n+1} = |\{0,1\}^{n+1}|$$

This means:
- At least $2^{n+1} - 2^n = 2^n$ strings in $\{0,1\}^{n+1}$ are **never** output by G
- These strings have probability 0 under $G(U_n)$
- But under the uniform distribution U_{n+1}, every string has probability $\frac{1}{2^{n+1}} > 0$

2.2.3. Conclusion

The output distribution of G **cannot** be identical to U_{n+1} because:

- Under U_{n+1}: Every string has probability $\frac{1}{2^{n+1}}$
- Under $G(U_n)$: At least half the strings have probability 0

Therefore, **perfect PRGs do not exist**.

2.2.4. Visual Intuition

Seeds: $\{0,1\}^n$	**Outputs: $\{0,1\}^{n+1}$**
2^n elements	$2^{n+1} = 2 \times 2^n$ elements
All used as inputs	Only 2^n can be reached

Since G is deterministic and "stretches" n bits into $n+1$ bits, it simply cannot cover the entire output space. The expansion creates a gap that makes perfect indistinguishability impossible.

> **Why Computational PRGs Work:** In practice, we relax the requirement to **computational indistinguishability** — no **efficient** (polynomial-time) algorithm can distinguish. This is achievable because we only need to fool limited adversaries, not omniscient ones.

◆ ◆ ◆

Problem 3: Hard-Core Predicates for DLP

Context: Consider the Discrete Logarithm Problem (DLP) (see Appendix B.1) with one-way function $f(x) = g^x \bmod p$ in $\mathbb{Z}_p^*$ for a prime p, where $(p-1) = s \cdot 2^r$ for some odd s (see Appendix B.2).

Tasks:
1. Prove the MSB (most significant bit) is a hard-core predicate for DLP
2. Prove the $(r+1)^{\text{th}}$ LSB is a hard-core predicate
3. Design a provably secure PRG assuming DLP is hard in $\mathbb{Z}_p^*$

Background: Hard-Core Predicates

Definition: A predicate $B : X \to \{0,1\}$ is **hard-core** for a one-way function f if:
- $B(x)$ is efficiently computable given x
- Given only $f(x)$, no efficient algorithm can predict $B(x)$ with probability significantly better than $\frac{1}{2}$

Formally: For all PPT adversaries A:

$$\Pr[A(f(x)) = B(x)] \leq \frac{1}{2} + \text{negl}(n)$$

For DLP: Given $y = g^x \bmod p$, hard-core predicates are bits of x that cannot be efficiently computed from y.

Part (a): MSB is Hard-Core for DLP

Claim: The most significant bit of x is a hard-core predicate for $f(x) = g^x \bmod p$.

Proof by Reduction:

Suppose there exists an efficient algorithm A that, given $y = g^x \bmod p$, predicts $\text{MSB}(x)$ with advantage $\varepsilon > \text{negl}(n)$:

$$\Pr[A(g^x) = \text{MSB}(x)] \geq \frac{1}{2} + \varepsilon$$

We show this would break DLP:

Step 1: Relating MSB to magnitude

For $x \in \{0, 1, ..., p-2\}$ (the valid range of discrete logs):

$$\text{MSB}(x) = \begin{cases} 0 \text{ if } x < \frac{p-1}{2} \\ 1 \text{ if } x \geq \frac{p-1}{2} \end{cases}$$

Step 2: Using the group structure

Key property: If $y = g^x$, then $y \cdot g^k = g^{x+k \bmod (p-1)}$

This means we can "shift" the discrete log by any known amount.

Step 3: Binary search using MSB oracle

Given $y = g^x$, we can find x using A:

1. Query $A(y)$ to get $\text{MSB}(x)$ — determines if x is in upper or lower half
2. Shift: compute $y' = y \cdot g^{-\frac{(p-1)}{4}}$ and query $A(y')$
3. Each query narrows the range by half
4. After $O(\log p)$ queries, we recover x exactly

Conclusion: If MSB is predictable, DLP is solvable in polynomial time. By contrapositive, if DLP is hard, MSB is hard-core. □

Part (b): The $(r+1)^{\text{th}}$ LSB is Hard-Core

Setup: Let $p - 1 = s \cdot 2^r$ where s is odd. We prove the $(r+1)^{\text{th}}$ least significant bit of x is hard-core.

Why $(r+1)^{\text{th}}$ specifically?

The first r LSBs of x can actually be computed efficiently from $y = g^x \bmod p$! Here's why:

Computing low-order bits: Since $|\mathbb{Z}_p^*| = p - 1 = s \cdot 2^r$:

- $y^s = g^{x \cdot s} = (g^s)^x$
- The element g^s has order exactly 2^r (a power of 2)
- This allows computing $x \bmod 2^r$ via the Pohlig-Hellman algorithm efficiently

Therefore, bits $0, 1, ..., r-1$ are **not** hard-core!

Proof that $(r+1)^{\text{th}}$ LSB is hard-core:

The $(r+1)^{\text{th}}$ LSB corresponds to $\lfloor \frac{x}{2^r} \rfloor \bmod 2$, i.e., the LSB of the "hard part" $\lfloor \frac{x}{2^r} \rfloor$.

Reduction: Suppose adversary A predicts this bit with advantage ε.

We can write $x = x_0 + 2^r \cdot x_1$ where:

- $x_0 = x \bmod 2^r$ (computable via Pohlig-Hellman (see Appendix B.3))
- $x_1 = \lfloor \frac{x}{2^r} \rfloor$ is in $\{0, 1, ..., s-1\}$

The $(r+1)^{\text{th}}$ LSB is precisely $x_1 \bmod 2$.

Key insight: Computing $y' = y \cdot g^{-x_0} = g^{2^r \cdot x_1}$, we get:

$$(y')^{\frac{1}{2^r}} = g^{x_1}$$

(where $\frac{1}{2^r}$ is computed $\bmod(p-1)$, possible since $\frac{\gcd(2^r, p-1)}{2^r}$ divides evenly)

The LSB of x_1 being predictable from g^{x_1} would allow binary search to find x_1, solving DLP in the subgroup of order s.

Since DLP in the subgroup of odd order s is assumed hard (this is where the actual DLP hardness lies), the $(r+1)^{\text{th}}$ bit is hard-core. □

Part (c): PRG Construction from DLP

Using the hard-core predicate, we construct a **Blum-Micali style PRG**:

3.4.1. Construction

PRG G: Given seed $x_0 \in \mathbb{Z}_p^*$, output n pseudorandom bits:

```
for i = 1 to n:
    y_i = g^(x_(i-1)) mod p     // One-way function
    b_i = MSB(x_(i-1))          // Hard-core bit
    x_i = y_i                   // Update state (treat y as next x)
output b_1 || b_2 || ... || b_n
```

More precisely, define the iteration as:

$$x_{i+1} = g^{x_i} \bmod p$$

$$b_i = \text{MSB}(x_i)$$

Output: $b_1, b_2, ..., b_n$

3.4.2. Why This Works

Property	Justification
Expansion	Seed of $\log p$ bits $\rightarrow$ n output bits (for any polynomial n)
Efficiency	Each step requires one modular exponentiation
Security	Each bit b_i is hard-core for the function $f(x) = g^x \bmod p$

3.4.3. Security Proof (Sketch)

Claim: The output is computationally indistinguishable from random.

Proof by hybrid argument:

Define hybrids $H_0, H_1, ..., H_n$ where:
- H_0: Real PRG output $(b_1, b_2, ..., b_n)$
- H_k: First k bits are truly random, rest from PRG
- H_n: All n bits are truly random

> **Key step:** Adjacent hybrids H_{k-1} and H_k are indistinguishable because distinguishing them requires predicting $b_k = \text{MSB}(x_{k-1})$ from $g^{x_{k-1}}$, which contradicts the hard-core property.

By transitivity: $H_0 \underset{c}{\approx} H_n$, so the PRG output is pseudorandom. □

3.4.4. Alternative: Using $(r+1)^{\text{th}}$ LSB

The same construction works with the $(r+1)^{\text{th}}$ LSB:

$$b_i = \text{bit}_{r+1}(x_i)$$

This may be preferable in some settings as the LSB can be slightly more efficient to extract.

Summary: The Blum-Micali PRG based on DLP:
- **Security:** Reduces to hardness of DLP
- **Efficiency:** One exponentiation per output bit
- **Output:** Can generate arbitrarily many pseudorandom bits from a short seed

◆ ◆ ◆

Problem 4: Modified Substitution Cipher

Task: Consider a modification where we first apply a substitution cipher (Appendix A.2), then apply a shift cipher (A.1) on the substituted values. Give a formal description and show how to break this scheme.

Formal Description

4.1.1. Notation

- **Alphabet:** $\Sigma = \{A, B, C, ..., Z\}$ with $|\Sigma| = 26$
- **Plaintext:** $m = m_1 m_2 ... m_n$ where each $m_i \in \Sigma$
- **Ciphertext:** $c = c_1 c_2 ... c_n$

4.1.2. Key Space

The key consists of two components:

$$K = (\pi, k)$$

where:

- $\pi : \Sigma \to \Sigma$ is a **permutation** (bijection) — the substitution key
- $k \in \{0, 1, 2, ..., 25\}$ — the shift amount

Key space size: $|\mathcal{K}| = 26! \times 26 \approx 1.04 \times 10^{\{28\}}$

4.1.3. Encryption

For each plaintext character m_i:

$$c_i = (\pi(m_i) + k) \bmod 26$$

Or equivalently:

$$\text{Enc}_{(\pi,k)}(m) = \text{Shift}_k(\text{Sub}_\pi(m))$$

4.1.4. Decryption

For each ciphertext character c_i:

$$m_i = \pi^{-1}((c_i - k) \bmod 26)$$

Or equivalently:

$$\text{Dec}_{(\pi,k)}(c) = \text{Sub}_{\pi^{-1}}(\text{Shift}_{-k}(c))$$

Breaking the Scheme

Despite the large key space, this cipher is **no more secure than a simple substitution cipher**. Here's why and how to break it:

4.2.1. Key Observation

> **The composition of a substitution and a shift is just another substitution!**
>
> Define $\sigma = \text{Shift}_k \circ \pi$, i.e., $\sigma(x) = (\pi(x) + k) \bmod 26$
>
> Since both π and Shift_k are permutations, their composition σ is also a permutation.

This means the "enhanced" cipher with key (π, k) is equivalent to a simple substitution cipher with key σ.

The shift adds **no additional security** — it's redundant!

4.2.2. Attack: Frequency Analysis

Since the scheme reduces to a substitution cipher, we use **frequency analysis**:

4.2.2.1. Step 1: Compute Ciphertext Frequencies

Count the frequency of each letter in the ciphertext:

$$f_c(x) = \frac{|\{i : c_i = x\}|}{n} \text{ for each } x \in \Sigma$$

4.2.2.2. Step 2: Compare with Known Language Frequencies

English letter frequencies (approximate):

E	T	A	O	I	N	S	H	R	D	L	U	C
12.7%	9.1%	8.2%	7.5%	7.0%	6.7%	6.3%	6.1%	6.0%	4.3%	4.0%	2.8%	2.8%

4.2.2.3. Step 3: Map Most Frequent Ciphertext Letters

Procedure:

1. Rank ciphertext letters by frequency
2. Match the most frequent ciphertext letter to 'E' (most common in English)
3. Match the second most frequent to 'T', and so on
4. Refine using common patterns (TH, THE, AND, etc.)

4.2.2.4. Step 4: Iterative Refinement

- Look for common digraphs: TH, HE, IN, ER, AN
- Look for common words: THE, AND, OF, TO, A
- Adjust mappings based on context and word patterns
- Use trial decryption and check for meaningful text

4.2.3. Complete Attack Algorithm

```
Input: Ciphertext c
Output: Plaintext m and key (π, k)

1. Compute frequency distribution of c
2. Initialize σ by matching frequencies to English
3. Decrypt using current σ guess
4. While decryption not readable:
     - Identify likely errors (nonsense patterns)
     - Swap suspected letter mappings
     - Re-decrypt and evaluate
5. Output final σ (which equals the composition of π and shift_k)
```

4.2.4. Why the Shift Doesn't Help

What you might expect	Reality
Two layers = harder to break	Composition = single substitution
Key space: $26! \times 26$	Effective key space: $26!$ (shift is absorbed)
Need to find both π and k	Only need to find $\sigma = \pi \circ \text{Shift}_k$

Conclusion: The modified cipher can be broken using standard frequency analysis techniques with $O(n)$ letter counting and human-guided (or automated) pattern matching. The shift cipher layer provides **zero additional security** because it merely relabels the already-permuted alphabet.

◆ ◆ ◆

Problem 5: Chosen Plaintext Attacks

Scenario: The adversary can obtain ciphertexts for arbitrary plaintexts of their choosing (without knowing the secret key). Show how to use this to learn the secret key for shift (Appendix A.1), substitution (A.2), and Vigenère (A.3) ciphers.

Part (a): Chosen Plaintext Attacks on Classical Ciphers

5.1.1. Attack on Shift Cipher

Cipher: $c_i = (m_i + k) \bmod 26$ where $k \in \{0, 1, ..., 25\}$

Key to recover: The shift amount k

Attack:
1. Choose plaintext: $m =$ "A" (just the single letter A, which has value 0)
2. Request encryption: Get $c = \text{Enc}_k(\text{A}) = (0 + k) \bmod 26 = k$
3. Recover key: $k = c$ (the ciphertext letter's position directly gives k)

Minimum plaintext length: 1 character

By encrypting 'A', the ciphertext directly reveals the shift value k.

5.1.2. Attack on Substitution Cipher

Cipher: $c_i = \pi(m_i)$ where $\pi : \Sigma \to \Sigma$ is a secret permutation

Key to recover: The entire permutation π (26 mappings)

Attack:

1. Choose plaintext: $m =$ "ABCDEFGHIJKLMNOPQRSTUVWXYZ" (the entire alphabet)
2. Request encryption: Get $c = \pi(\text{A})\pi(\text{B})...\pi(\text{Z})$
3. Recover key: The i^{th} character of c gives $\pi(\text{letter}_i)$

> **Minimum plaintext length: 26 characters**
>
> We need to query each letter exactly once to learn the complete mapping.

5.1.3. Attack on Vigenère Cipher (Period t Known)

Cipher: $c_i = (m_i + k_{i \bmod t}) \bmod 26$ where key $= k_0 k_1 ... k_{t-1}$

Key to recover: The t shift values $k_0, k_1, ..., k_{t-1}$

Attack:
1. Choose plaintext: $m =$ "AAA…A" (t copies of 'A')
2. Request encryption: Get $c = c_0 c_1 ... c_{t-1}$
3. Recover key: $k_i = c_i$ for each $i \in \{0, 1, ..., t-1\}$

> **Minimum plaintext length: t characters** (when period t is known)
>
> Each 'A' at position i encrypts to $k_{i \bmod t}$, directly revealing each key byte.

Part (b): Vigenère with Unknown Period

5.2.1. Case (i): Period t is Known

As shown above:

Plaintext	"AAA…A" (t times)
Length required	t
Key recovery	$k_i = c_i$ directly

5.2.2. Case (ii): Period t Unknown, Upper Bound $t_{\max}$ Known

We know the period $t \leq t_{\max}$ but don't know exact t.

Strategy: Choose a plaintext that works for **any** possible period up to $t_{\max}$.

> **Approach 1: Brute Force over Periods**
>
> For each candidate period $t' \in \{1, 2, ..., t_{\max}\}$:
> - Use a plaintext of length t' (all A's)
> - Check if decryption is consistent
>
> Total queries: $t_{\max}$ plaintexts, but we want a **single** plaintext.

Optimal Single-Plaintext Attack:

Choose plaintext $m =$ "AAA…A" of length $t_{\max}$.

1. Request encryption of m: Get $c = c_0 c_1 ... c_{t_{\max}-1}$
2. The ciphertext directly reveals: $c_i = k_{i \bmod t}$

3. To find t: Look for the **period** of the sequence $c_0, c_1, c_2, ...$

> **Finding the period:** The true period t is the smallest value such that:
>
> $$c_i = c_{i+t} \text{ for all } i \in \{0, 1, ..., t_{\max} - t - 1\}$$
>
> Once t is found, the key is simply $k = c_0 c_1 ... c_{t-1}$.

Asymptotic Analysis:

Minimum plaintext length: $O(t_{\max})$

More precisely: $t_{\max}$ characters suffice.

Reasoning: With $t_{\max}$ characters, we observe at least one complete period of the key (since $t \leq t_{\max}$), which is sufficient to both:
1. Determine t by finding the period of the ciphertext
2. Extract all t key bytes

5.2.3. Summary Table

Cipher	Key Size	Min Plaintext Length
Shift	1 value	**1**
Substitution	26 mappings	**26**
Vigenère (known t)	t values	$\boldsymbol{t}$
Vigenère (unknown t)	t values, $t \leq t_{\max}$	$\boldsymbol{O(t_{\max})}$

> **Key Insight:** In the chosen plaintext model, all classical ciphers can be broken with a single, carefully chosen plaintext. The required length equals the size of the key space that needs to be determined.

◆ ◆ ◆

Problem 6: Negligible or Not?

Topic: Analysis of negligible functions — fundamental concept in cryptographic security definitions (see Appendix C.5).

Background: What is a Negligible Function?

Definition: A function $f : \mathbb{N} \to \mathbb{R}$ is **negligible** if for every positive polynomial $p(n)$, there exists $N \in \mathbb{N}$ such that for all $n > N$:

$$|f(n)| < \frac{1}{p(n)}$$

Intuition: $f(n)$ decreases faster than the inverse of any polynomial — it's "super-polynomially small."

Notation: We write $f(n) = \text{negl}(n)$ to denote that f is negligible.

Key characterization: f is negligible $\Leftrightarrow$ for all $c > 0$: $\lim_{n\to\infty} n^c \cdot f(n) = 0$

Part (a): Properties of Negligible Functions

Let $f, g : \mathbb{N} \to \mathbb{R}$ be negligible functions, and let $p : \mathbb{N} \to \mathbb{R}$ be a polynomial with $p(n) > 0$ for all $n \in \mathbb{N}$.

6.2.1. (i) $h(n) = f(n) + g(n)$

Claim: $h(n)$ is negligible.

Proof:

Let $q(n)$ be any positive polynomial. We need to show $|h(n)| < \frac{1}{q(n)}$ for sufficiently large n.

Since f is negligible, there exists N_1 such that for $n > N_1$:

$$|f(n)| < \frac{1}{2q(n)}$$

Since g is negligible, there exists N_2 such that for $n > N_2$:

$$|g(n)| < \frac{1}{2q(n)}$$

For $n > \max(N_1, N_2)$:

$$|h(n)| = |f(n) + g(n)| \leq |f(n)| + |g(n)| < \frac{1}{2q(n)} + \frac{1}{2q(n)} = \frac{1}{q(n)}$$

Conclusion: The sum of two negligible functions is negligible. ✓

Generalization: The sum of polynomially many negligible functions is also negligible.

6.2.2. (ii) $h(n) = f(n) \cdot p(n)$

Claim: $h(n)$ is negligible.

Proof:

Let $q(n)$ be any positive polynomial. We need $|h(n)| < \frac{1}{q(n)}$ for large n.

Define $r(n) = q(n) \cdot p(n)$. Since q and p are both polynomials, $r(n)$ is also a polynomial.

Since f is negligible, there exists N such that for $n > N$:

$$|f(n)| < \frac{1}{r(n)} = \frac{1}{q(n) \cdot p(n)}$$

Therefore, for $n > N$:

$$|h(n)| = |f(n) \cdot p(n)| = |f(n)| \cdot p(n) < \frac{1}{q(n) \cdot p(n)} \cdot p(n) = \frac{1}{q(n)}$$

> **Conclusion:** A negligible function multiplied by a polynomial is still negligible. ✓
>
> **Intuition:** Polynomials can't "catch up" to super-polynomial decay.

6.2.3. (iii) $f(n) := f'(n) \cdot p(n)$ for some negligible f' and some polynomial p

Question: Is such an $f(n)$ always negligible?

Answer: Yes, such an $f(n)$ is always negligible.

Proof:

This follows directly from part (ii). Given:

- $f'(n)$ is negligible (by assumption)
- $p(n)$ is a polynomial (by assumption)

By the result of part (ii), the product $f'(n) \cdot p(n)$ is negligible.

> **Key insight:** The statement "for some negligible f' and some polynomial p" simply means we're given a specific negligible function and a specific polynomial. Their product is always negligible, regardless of which specific functions they are.

Summary of Part (a) — Closure Properties:

Operation	Result
$\text{negl}_1 + \text{negl}_2$	Negligible ✓
$\text{negl} \times \text{poly}$	Negligible ✓
$\text{negl}_1 \times \text{negl}_2$	Negligible ✓ (even smaller!)

Part (b): Analyzing Specific Functions

For each function, we determine whether it is negligible by checking if it decays faster than any inverse polynomial.

Key technique: Take logarithms and compare growth rates.

6.3.1. (i) $f(n) = \frac{1}{2^{100 \log n}}$

Simplification:

$$f(n) = \frac{1}{2^{100 \log n}} = \frac{1}{(2^{\log n})^{100}} = \frac{1}{n^{100 \cdot \log 2}} \approx \frac{1}{n^{30.1}}$$

Wait, let's be more careful. Assuming log is base 2:

$$2^{100 \log_2 n} = \left(2^{\log_2 n}\right)^{100} = n^{100}$$

So $f(n) = \frac{1}{n^{100}} = n^{-100}$.

Analysis: This is exactly an inverse polynomial (n^{-100}).

For f to be negligible, we need $n^c \cdot f(n) \to 0$ for all $c > 0$.

But with $c = 101$:

$$n^{101} \cdot n^{-100} = n \to \infty$$

> **Verdict: NOT negligible ✗**
>
> $f(n) = n^{-100}$ is polynomial decay, not super-polynomial.

6.3.2. (ii) $f(n) = \frac{1}{(\log n)^{\log n}}$

Analysis using logarithms:

$$\log f(n) = -\log n \cdot \log(\log n)$$

Compare with inverse polynomial $\frac{1}{n^c}$, which has $\log\left(\frac{1}{n^c}\right) = -c \log n$.

We need: Is $\log n \cdot \log(\log n)$ eventually larger than $c \log n$ for any c?

$$\frac{\log n \cdot \log(\log n)}{c \log n} = \frac{\log(\log n)}{c} \to \infty$$

as $n \to \infty$

So $f(n)$ decays faster than any n^{-c}.

Formal verification: For any polynomial $p(n) = n^c$:

$$n^c \cdot f(n) = \frac{n^c}{(\log n)^{\log n}}$$

Taking logs: $c \log n - \log n \cdot \log(\log n) = \log n(c - \log(\log n))$

As $n \to \infty$, $\log(\log n) \to \infty$, so this becomes $-\infty$, meaning $n^c \cdot f(n) \to 0$.

> **Verdict: Negligible ✓**
>
> $(\log n)^{\log n}$ grows super-polynomially (faster than any n^c).

6.3.3. (iii) $f(n) = n^{-100} + 2^{-n}$

Analysis:

- n^{-100}: Polynomial decay (NOT negligible by itself)
- 2^{-n}: Exponential decay (negligible)

The sum is dominated by the slower-decaying term:

$$f(n) \approx n^{-100} \text{ for large } n$$

For $c = 101$:

$$n^{101} \cdot f(n) \geq n^{101} \cdot n^{-100} = n \to \infty$$

> **Verdict: NOT negligible ✗**
>
> The n^{-100} term dominates and is only polynomial decay.

6.3.4. (iv) $f(n) = 1.01^{-n}$

Analysis:

$$f(n) = \left(\frac{1}{1.01}\right)^n = \left(\frac{100}{101}\right)^n$$

This is exponential decay with base < 1.

For any $c > 0$:

$$n^c \cdot (1.01)^{-n} = \frac{n^c}{(1.01)^n}$$

The exponential $(1.01)^n$ grows faster than any polynomial, so this ratio $\to 0$.

Formal: Using L'Hôpital's rule or the fact that exponentials dominate polynomials:

$$\lim_{n\to\infty} \frac{n^c}{a^n} = 0 \text{ for any } a > 1, c > 0$$

> **Verdict: Negligible ✓**
>
> Exponential decay (even with base as small as 1.01) is negligible.

6.3.5. (v) $f(n) = 2^{-(\log n)^2}$
Analysis:

$$\log f(n) = -(\log n)^2$$

Compare with $\frac{1}{n^c}$: $\log(n^{-c}) = -c \log n$

Is $(\log n)^2$ eventually larger than $c \log n$ for any c?

$$\frac{(\log n)^2}{c \log n} = \frac{\log n}{c} \to \infty$$

So $f(n)$ decays faster than any inverse polynomial.

Verification:

$$n^c \cdot f(n) = 2^{c \log n} \cdot 2^{-(\log n)^2} = 2^{c \log n - (\log n)^2} = 2^{\log n (c - \log n)}$$

For large n, $\log n > c$, so exponent $\to -\infty$, hence $n^c \cdot f(n) \to 0$.

> **Verdict: Negligible ✓**
>
> $(\log n)^2$ in the exponent grows faster than linear, causing super-polynomial decay.

6.3.6. (vi) $f(n) = 2^{-\sqrt{n}}$
Analysis:

$$\log f(n) = -\sqrt{n}$$

Compare with $\frac{1}{n^c}$: need $\sqrt{n}$ vs $c \log n$.

$$\frac{\sqrt{n}}{c \log n} \to \infty \text{ as } n \to \infty$$

So $\sqrt{n}$ grows faster than $\log n$, meaning $f(n)$ decays faster than any n^{-c}.

Verification:

$$n^c \cdot f(n) = \frac{n^c}{2^{\sqrt{n}}} = \frac{2^{c \log n}}{2^{\sqrt{n}}} = 2^{c \log n - \sqrt{n}}$$

Since $\sqrt{n}$ grows faster than $\log n$, the exponent $\to -\infty$.

> **Verdict: Negligible ✓**
>
> $\sqrt{n}$ grows faster than any $c \log n$, so $2^{-\sqrt{n}}$ is negligible.

6.3.7. (vii) $f(n) = 2^{-\sqrt{\log n}}$
Analysis:

$$\log f(n) = -\sqrt{\log n}$$

Compare with $\frac{1}{n^c}$: need $\sqrt{\log n}$ vs $c \log n$.

$$\frac{\sqrt{\log n}}{c \log n} = \frac{1}{c\sqrt{\log n}} \to 0 \text{ as } n \to \infty$$

This means $\sqrt{\log n}$ grows **slower** than $c \log n$!

The critical test: For $c = 1$:

$$n \cdot f(n) = \frac{n}{2^{\sqrt{\log n}}} = \frac{2^{\log n}}{2^{\sqrt{\log n}}} = 2^{\log n - \sqrt{\log n}}$$

Since $\log n - \sqrt{\log n} = \sqrt{\log n}(\sqrt{\log n} - 1) \to \infty$, we get $n \cdot f(n) \to \infty$.

> **Verdict: NOT negligible ✗**
>
> $\sqrt{\log n}$ grows too slowly — slower than $c \log n$ for any $c > 0$.

6.3.8. (viii) $f(n) = \frac{1}{(\log n)!}$
Analysis using Stirling's approximation:

$$(\log n)! \approx \sqrt{2\pi \log n}\left(\frac{\log n}{e}\right)^{\log n}$$

So:

$$\log((\log n)!) \approx (\log n) \cdot \log(\log n) - (\log n) \cdot \log e + O(\log \log n)$$

$$\approx (\log n) \cdot (\log(\log n) - 1)$$

For large n, this is $\approx (\log n) \cdot \log(\log n)$.

Compare with $c \log n$:

$$\frac{(\log n) \cdot \log(\log n)}{c \log n} = \frac{\log(\log n)}{c} \to \infty$$

So $(\log n)!$ grows super-polynomially.

> **Verdict: Negligible ✓**
>
> $(\log n)!$ grows faster than any polynomial due to factorial growth.

Summary Table

No.	Function	Negligible?
(i)	$f(n) = \frac{1}{2^{100 \log n}} = n^{-100}$	NO ✗
(ii)	$f(n) = \frac{1}{(\log n)^{\log n}}$	YES ✓
(iii)	$f(n) = n^{-100} + 2^{-n}$	NO ✗
(iv)	$f(n) = 1.01^{-n}$	YES ✓
(v)	$f(n) = 2^{-(\log n)^2}$	YES ✓
(vi)	$f(n) = 2^{-\sqrt{n}}$	YES ✓
(vii)	$f(n) = 2^{-\sqrt{\log n}}$	NO ✗
(viii)	$f(n) = \frac{1}{(\log n)!}$	YES ✓

Key Insight — The Negligibility Threshold:

A function $f(n) = 2^{-g(n)}$ is negligible $\Leftrightarrow$ $g(n) = \omega(\log n)$

Equivalently, $\frac{g(n)}{\log n} \to \infty$ as $n \to \infty$.

Examples:

- $g(n) = \sqrt{n}$: negligible ($\frac{\sqrt{n}}{\log n} \to \infty$)
- $g(n) = (\log n)^2$: negligible
- $g(n) = \sqrt{\log n}$: NOT negligible ($\frac{\sqrt{\log n}}{\log n} \to 0$)
- $g(n) = 100 \log n$: NOT negligible (just polynomial decay)

◆ ◆ ◆

Problem 7: Unsuitable Primes for DLP-Based Cryptography

Question: Let p be a prime and consider the discrete logarithm problem in the group $\mathbb{F}_p^*$, which has order $p - 1$. Explain which primes p are unsuitable for discrete-logarithm-based cryptography due to the Pohlig-Hellman attack (see Appendix B.3). In particular, characterize the factorization of $p - 1$ that makes the discrete logarithm problem efficiently solvable.

Background: The Pohlig-Hellman Attack

The Pohlig-Hellman algorithm exploits the **factorization structure** of the group order to solve DLP efficiently when the order has only small prime factors.

Key Insight: If $|G| = p - 1 = \prod_{i=1}^{k} q_i^{e_i}$ where all q_i are "small", then:

1. Solve DLP separately in each subgroup of order $q_i^{e_i}$
2. Combine solutions using the Chinese Remainder Theorem

Complexity: $O\left(\sum_i e_i(\log n + \sqrt{q_i})\right)$ instead of $O(\sqrt{p})$ for generic algorithms.

Characterization of Unsuitable Primes

7.2.1. Definition: Smooth Numbers

B-smooth: An integer n is called ***B*-smooth** if all its prime factors are $\leq B$.

Example: $60 = 2^2 \times 3 \times 5$ is 5-smooth, but not 4-smooth.

7.2.2. The Vulnerability Condition

> **A prime p is UNSUITABLE for DLP-based cryptography if $p - 1$ is *B*-smooth for relatively small *B*.**
>
> More precisely, if all prime factors of $p - 1$ are polynomial in $\log p$, the DLP can be solved in polynomial time.

7.2.3. Why Smooth $p - 1$ is Dangerous

Let $p - 1 = q_1^{e_1} \times q_2^{e_2} \times ... \times q_k^{e_k}$.

The Pohlig-Hellman algorithm:

Step	Operation
1	For each prime power $q_i^{e_i}$ dividing $p - 1$:
	— Project to subgroup of order $q_i^{e_i}$: compute $y' = y^{\frac{p-1}{q_i^{e_i}}}$
	— Solve DLP in this subgroup (order $q_i^{e_i}$): find $x_i = x \bmod q_i^{e_i}$
2	Combine using CRT: $x \equiv x_i \bmod q_i^{e_i}$ for all i

Complexity Analysis:

- Solving DLP in subgroup of order q^e: $O(e(\log p + \sqrt{q}))$ using baby-step giant-step
- If largest prime factor $q_{\max} = O(\text{poly}(\log p))$, then total time is polynomial!

Precise Characterization

Theorem: The DLP in $\mathbb{F}_p^*$ can be solved in time $O\left(\sqrt{q_{\max}} \cdot \text{poly}(\log p)\right)$ where $q_{\max}$ is the **largest prime factor** of $p - 1$.

This leads to the following classification:

Factorization of $p-1$	Security	Verdict
$p-1 = 2 \times q$ (large prime q)	DLP hard: $O(\sqrt{q}) \approx O(\sqrt{p})$	**SAFE**
$p-1$ has large prime factor $\geq p^{\frac{1}{3}}$	DLP still hard	**SAFE**
$p-1 = 2^k$ (power of 2)	DLP trivial: $O(k^2)$	**UNSAFE**
$p-1$ is B-smooth, $B = O(\log p)$	DLP poly-time	**UNSAFE**
$p-1$ is B-smooth, $B = O(p^\epsilon)$	DLP subexponential	**WEAK**

Examples

7.4.1. Example 1: UNSAFE Prime

Let $p = 257 = 2^8 + 1$, so $p - 1 = 256 = 2^8$.

- Largest prime factor: $q_{\max} = 2$
- DLP complexity: $O\left(8 \times \left(\log 257 + \sqrt{2}\right)\right) = O(8)$ — trivial!

> **This prime is completely unsuitable.** The DLP can be solved immediately.

7.4.2. Example 2: SAFE Prime

A **safe prime** is a prime p such that $q = \frac{p-1}{2}$ is also prime.

Let $p = 23$, so $p - 1 = 22 = 2 \times 11$.

- Largest prime factor: $q_{\max} = 11$
- DLP complexity: $O\left(\sqrt{11}\right) \approx O(3)$ operations in subgroup

For cryptographic sizes (e.g., $p \approx 2^{\{2048\}}$):
- $p - 1 = 2q$ where q is a 2047-bit prime
- DLP complexity: $O(\sqrt{q}) \approx O\left(2^{\{1024\}}\right)$ — infeasible!

> **Safe primes are ideal for DLP-based cryptography.** They maximize the difficulty of Pohlig-Hellman.

7.4.3. Example 3: WEAK Prime (Smooth)

Let $p - 1 = 2^{\{10\}} \times 3^5 \times 5^3 \times 7^2 \times 11 \times 13$.

All prime factors are ≤ 13, so $p - 1$ is 13-smooth.

- DLP can be solved by combining solutions from 6 small subgroups
- Total complexity: polynomial in $\log p$

Summary: Selecting Safe Primes

Requirements for DLP-based Cryptography:

1. **Choose p such that $p - 1$ has at least one LARGE prime factor**
   - "Large" means $\geq p^{\frac{1}{3}}$ or comparable to $\sqrt{p}$

2. **Ideal: Use SAFE PRIMES where $p = 2q + 1$ with q prime**
   - This ensures largest factor of $p - 1$ is $q \approx \frac{p}{2}$
   - Pohlig-Hellman gives no advantage over generic algorithms
3. **Alternative: Use groups of prime order**
   - Choose a prime-order subgroup of $\mathbb{F}_p^*$ (e.g., Schnorr groups)
   - Work in subgroup of order q where $q \mid (p - 1)$ is large prime

Key Takeaway: The security of the discrete logarithm problem in $\mathbb{F}_p^*$ depends critically on the **largest prime factor** of $p - 1$. If $p - 1$ is smooth (all factors small), the Pohlig-Hellman attack makes DLP trivially solvable. Always use primes where $p - 1$ has a large prime factor — preferably safe primes of the form $p = 2q + 1$.

◆ ◆ ◆

Problem 8: Uniform vs Non-Uniform Key Distributions

Setting: Let $(\text{Gen}, \text{Enc}, \text{Dec})$ be an encryption scheme where Gen outputs a key K according to an arbitrary (not necessarily uniform) distribution over some finite key space $\mathcal{K}$. For any message m, let $C = \text{Enc}(K, m)$, where the probability is over the randomness of Gen (and Enc, if randomized).

Task: Prove that there exists an equivalent encryption scheme $(\text{Gen}', \text{Enc}', \text{Dec}')$ with a (possibly different) key space $\mathcal{K}'$ such that:
1. Gen' samples a key uniformly from $\mathcal{K}'$; and
2. For all messages m and ciphertexts c: $\Pr[C = c \mid M = m] = \Pr[C' = c \mid M = m]$

where $C' = \text{Enc}'(K', m)$ and the probability is over the randomness of Gen' (and Enc', if applicable).

Intuition

We need to show that **any** key distribution can be "simulated" by a uniform distribution over a (possibly larger) key space, without changing the distribution of ciphertexts.

The key idea: If some keys are more likely than others, we can create "multiple copies" of likely keys to make them appear uniform.

Construction of the Equivalent Scheme

8.2.1. Step 1: Analyze the Original Distribution

Let the original key distribution be:

$$\Pr[\text{Gen}() = k] = p_k \text{ for each } k \in \mathcal{K}$$

where $\sum_{k \in \mathcal{K}} p_k = 1$.

8.2.2. Step 2: Express Probabilities with Common Denominator

Since $\mathcal{K}$ is finite, all probabilities p_k are rational numbers (or can be approximated arbitrarily well by rationals).

Write each probability as:

$$p_k = \frac{n_k}{N}$$

where N is a common denominator and $n_k \in \mathbb{N}$ for all k.

Technical note: We have $\sum_{k \in \mathcal{K}} n_k = N$.

Each n_k represents "how many times" key k should appear in the new key space.

8.2.3. Step 3: Define the New Key Space

Define the new key space as:

$$\mathcal{K}' = \{(k, i) : k \in \mathcal{K}, i \in \{1, 2, ..., n_k\}\}$$

In other words, for each original key k, we create n_k "copies" of it, indexed by i.

Size of new key space: $|\mathcal{K}'| = \sum_{k \in \mathcal{K}} n_k = N$

8.2.4. Step 4: Define the New Algorithms

$\mathbf{Gen'}$: Sample $(k, i) \in \mathcal{K}'$ uniformly at random.

Since $|\mathcal{K}'| = N$ and key k has exactly n_k copies:

$$\Pr[\text{Gen}'() \text{ (has underlying key) } k] = \frac{n_k}{N} = p_k \checkmark$$

$\mathbf{Enc'((k, i), m)}$: Return $\text{Enc}(k, m)$.

The index i is ignored — encryption depends only on the underlying key k.

$\mathbf{Dec'((k, i), c)}$: Return $\text{Dec}(k, c)$.

Similarly, decryption ignores the index and uses k.

Proof of Equivalence

8.3.1. Property 1: Gen' is Uniform over $\mathcal{K}'$

By construction, Gen' samples uniformly from $\mathcal{K}'$:

$$\Pr[\text{Gen}'() = (k, i)] = \frac{1}{N} \text{ for all } (k, i) \in \mathcal{K}'$$

This satisfies requirement 1. ✓

8.3.2. Property 2: Ciphertext Distributions are Identical

For any message m and ciphertext c:

Original scheme:

$$\Pr[C = c \mid M = m] = \sum_{k \in \mathcal{K}} \Pr[\text{Gen}() = k] \cdot \Pr[\text{Enc}(k, m) = c] = \sum_{k \in \mathcal{K}} p_k \cdot \Pr[\text{Enc}(k, m) = c]$$

New scheme:

$$\begin{aligned}
\Pr[C' = c \mid M = m] &= \sum_{(k,i) \in \mathcal{K}'} \Pr[\text{Gen}'() = (k, i)] \cdot \Pr[\text{Enc}'((k, i), m) = c] \\
&= \sum_{(k,i) \in \mathcal{K}'} \frac{1}{N} \cdot \Pr[\text{Enc}(k, m) = c] \\
&= \sum_{k \in \mathcal{K}} \sum_{i=1}^{n_k} \frac{1}{N} \cdot \Pr[\text{Enc}(k, m) = c] \\
&= \sum_{k \in \mathcal{K}} \frac{n_k}{N} \cdot \Pr[\text{Enc}(k, m) = c] \\
&= \sum_{k \in \mathcal{K}} p_k \cdot \Pr[\text{Enc}(k, m) = c]
\end{aligned}$$

> **The two expressions are identical!**
>
> $\Pr[C = c \mid M = m] = \Pr[C' = c \mid M = m]$✓

This satisfies requirement 2. □

Summary

Component	Original Scheme	Equivalent Scheme
Key Space	$\mathcal{K}$	$\mathcal{K}' = \{(k, i) : k \in \mathcal{K}, i \leq n_k\}$
Key Distribution	Arbitrary: $\Pr[K = k] = p_k$	Uniform: $\Pr[K' = (k, i)] = \frac{1}{N}$
Encryption	$\text{Enc}(k, m)$	$\text{Enc}'((k, i), m) = \text{Enc}(k, m)$
Decryption	$\text{Dec}(k, c)$	$\text{Dec}'((k, i), c) = \text{Dec}(k, c)$

> **Key Insight:** Any encryption scheme with non-uniform key distribution can be transformed into one with uniform key distribution by "unfolding" the probability mass — replicating keys in proportion to their original probability. This is essentially **inverse transform sampling** applied to key generation.
>
> **Security implication:** This shows that assuming uniform key generation is without loss of generality when analyzing encryption scheme security!

◆ ◆ ◆

Problem 9: PRG or Not?

Setup: Let $G : \{0,1\}^{2n} \to \{0,1\}^{2n+1}$ be a pseudorandom generator (PRG) (see Appendix C.3). For each construction below, determine whether $G' : \{0,1\}^{2n} \to \{0,1\}^{2n+1}$ is necessarily a PRG, regardless of which PRG G is used.

Background: PRG Definition

A function $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ with $\ell(n) > n$ is a **PRG** if for all PPT distinguishers D:

$$|\Pr[D(G(U_n)) = 1] - \Pr\left[D\left(U_{\ell(n)}\right) = 1\right]| \leq \text{negl}(n)$$

where U_k denotes the uniform distribution over $\{0,1\}^k$.

Part (a): $G'(x) := G(\pi(x))$ where π is a Bijection

Construction: $\pi : \{0,1\}^{2n} \to \{0,1\}^{2n}$ is a poly(n)-time computable bijection, but π^{-1} may NOT be poly-time computable.

> **Answer: YES, G' is a PRG.**

Proof by reduction:

Suppose G' is not a PRG. Then there exists a PPT distinguisher D such that:

$$|\Pr[D(G'(U_{2n})) = 1] - \Pr[D(U_{2n+1}) = 1]| > \text{non-negl}(n)$$

But $G'(U_{2n}) = G(\pi(U_{2n}))$.

Key observation: Since π is a bijection, $\pi(U_{2n})$ is **also** uniformly distributed over $\{0,1\}^{2n}$!

Why? A bijection is a one-to-one correspondence. When the input is uniform, each output value is hit by exactly one input, so the output is also uniform.

Formally: For any $y \in \{0,1\}^{2n}$:

$$\Pr[\pi(U_{2n}) = y] = \Pr[U_{2n} = \pi^{-1}(y)] = \frac{1}{2^{2n}}$$

Therefore:

$$G'(U_{2n}) = G(\pi(U_{2n})) \equiv G(U_{2n})$$

So if D distinguishes $G'(U_{2n})$ from U_{2n+1}, then D also distinguishes $G(U_{2n})$ from U_{2n+1}.

This contradicts that G is a PRG. $\square$

> **Note:** We don't need π^{-1} to be efficiently computable. The reduction only uses π in the forward direction, and the analysis only uses that π is a bijection.

Part (b): $G'(x \parallel y) := G(x \parallel (x \oplus y))$ where $|x| = |y| = n$

Construction: Split the $2n$-bit seed into two n-bit halves, then apply G to $x \parallel (x \oplus y)$.

> **Answer: YES, G' is a PRG.**

Proof:

Define the mapping $\varphi : \{0,1\}^{2n} \to \{0,1\}^{2n}$ by:

$$\varphi(x \parallel y) = x \parallel (x \oplus y)$$

Claim: φ is a bijection.

Proof of claim: The inverse is:

$$\varphi^{-1}(a \parallel b) = a \parallel (a \oplus b)$$

Check: $\varphi^{-1}(\varphi(x \parallel y)) = \varphi^{-1}(x \parallel (x \oplus y)) = x \parallel (x \oplus (x \oplus y)) = x \parallel y$ ✓

Since φ is a bijection, by the same argument as Part (a):

$$G'(U_{2n}) = G(\varphi(U_{2n})) \equiv G(U_{2n})$$

Therefore G' is a PRG. □

Part (c): $G'(x \parallel y) := G(x \parallel 0^n) \oplus G(0^n \parallel y)$ where $|x| = |y| = n$

Construction: Evaluate G on two different inputs and XOR the results.

> **Answer: NO, G' is NOT necessarily a PRG.**

Counterexample:

Let G be **any** PRG. Define a new PRG $\hat{G}$ by:

$$\hat{G}(s) = G(s) \oplus (s \parallel 0)$$

Note: $\hat{G}$ is still a PRG because XORing with a fixed function of the seed doesn't help a distinguisher (the seed is unknown).

Now apply the G' construction to $\hat{G}$:

$$\begin{aligned} G'(x \parallel y) &= \hat{G}(x \parallel 0^n) \oplus \hat{G}(0^n \parallel y) \\ &= [G(x \parallel 0^n) \oplus (x \parallel 0^n \parallel 0)] \oplus [G(0^n \parallel y) \oplus (0^n \parallel y \parallel 0)] \end{aligned}$$

The XOR of the "extra" terms produces:

$$(x \parallel 0^{n+1}) \oplus (0^n \parallel y \parallel 0) = (x \parallel y \parallel 0)$$

Problem: The output $G'(x \parallel y)$ now contains $(x \parallel y \parallel 0)$ XORed in!

A distinguisher can:
1. Receive output $z \in \{0,1\}^{2n+1}$
2. Check if the last bit is 0
3. Real PRG output: last bit is always $0 \to$ biased!
4. Random string: last bit is 0 with probability $\frac{1}{2}$

More directly: For any x, y, the last bit of $G'(x \parallel y)$ equals:

$$\text{last bit of } G(x\|0^n) \oplus \text{last bit of } G(0^n\|y) \oplus 0$$

But we can construct $\hat{G}$ so this is always 0, breaking pseudorandomness.

Simpler counterexample: Let $G(s) = s \parallel (\text{parity of } s)$.

Then $G(x \parallel 0^n) \oplus G(0^n \parallel y)$ has predictable structure. □

Part (d): $G'(x \parallel y) := G(x \parallel y) \oplus (x \parallel 0^{n+1})$ where $|x| = |y| = n$

Construction: XOR the PRG output with the first half of the seed (padded with zeros).

> **Answer: NO, G' is NOT necessarily a PRG.**

Counterexample:

Define a valid PRG G as follows. Let H be any PRG with the same parameters. Define:

$$G(x \parallel y) = H(x \parallel y) \oplus (x \parallel 0^{n+1})$$

Since H is a PRG and XORing with a function of the seed doesn't help distinguish (the adversary doesn't know the seed), G is also a PRG.

Now compute G':

$$\begin{aligned} G'(x \parallel y) &= G(x \parallel y) \oplus (x \parallel 0^{n+1}) \\ &= [H(x \parallel y) \oplus (x \parallel 0^{n+1})] \oplus (x \parallel 0^{n+1}) \\ &= H(x \parallel y) \end{aligned}$$

Wait, this gives us back H, which IS a PRG. Let me reconsider…

Correct counterexample:

Let $G(s) = s \parallel \text{MSB}(s)$ where $s \in \{0,1\}^{2n}$ and MSB is the most significant bit.

This is a valid PRG (the output $s \parallel \text{MSB}(s)$ is pseudorandom when s is random).

Now:

$$G'(x \parallel y) = G(x \parallel y) \oplus (x \parallel 0^{n+1}) = (x \parallel y \parallel \text{MSB}(x\|y)) \oplus (x \parallel 0^{n+1})$$

$$= (x \oplus x) \parallel (y \parallel \text{MSB}(x\|y)) \oplus (0^{n+1}) = 0^n \parallel y \parallel \text{MSB}(x\|y)$$

The first n bits of $G'(x \parallel y)$ are always 0^n!

A distinguisher simply checks if the first n bits are all zeros:

- Real G' output: always yes
- Random $(2n+1)$-bit string: probability $\frac{1}{2^n}$

This is easily distinguishable. □

Summary Table

Part	Construction	PRG?
(a)	$G'(x) = G(\pi(x))$, π bijection	YES ✓
(b)	$G'(x\|y) = G(x \parallel (x \oplus y))$	YES ✓
(c)	$G'(x\|y) = G(x\|0^n) \oplus G(0^n\|y)$	NO ✗
(d)	$G'(x\|y) = G(x\|y) \oplus (x\|0^{n+1})$	NO ✗

Key Insights:

- **Bijections preserve uniformity:** Composing with a bijection maintains pseudorandomness (parts a, b)
- **XORing parts of seed into output is dangerous:** Can create predictable patterns (part d)
- **Combining PRG evaluations:** XORing outputs of the **same** PRG on related inputs can leak structure (part c)

◆ ◆ ◆

Problem 10: 2-Time Perfectly Secure Encryption

Definition: An encryption scheme $(\text{Gen}, \text{Enc}, \text{Dec})$ over message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is **2-time perfectly secure** if for any $(m_1, m_2) \in \mathcal{M} \times \mathcal{M}$ and $(m_1', m_2') \in \mathcal{M} \times \mathcal{M}$ such that $m_1 \neq m_2$ and $m_1' \neq m_2'$, and for any $c_1, c_2 \in \mathcal{C}$:

$$\Pr[\text{Enc}(K, m_1) = c_1 \wedge \text{Enc}(K, m_2) = c_2] = \Pr[\text{Enc}(K, m_1') = c_1 \wedge \text{Enc}(K, m_2') = c_2]$$

Encryption Scheme over $\mathbb{Z}_{23}$:

- **Gen:** Sample two elements $a \leftarrow \mathbb{Z}_{23}$ and $b \leftarrow \mathbb{Z}_{23}$
- **Enc$((a,b),m)$:** Output $c = a \cdot m + b \bmod 23$
- **Dec$((a,b),c)$:** Compute $m = (c - b) \cdot a^{-1} \bmod 23$ if a is invertible; otherwise output error

Tasks:

1. Prove that for any message $m \in \mathbb{Z}_{23}$: $\Pr[\text{Dec}(K, \text{Enc}(K, m)) = m] = \frac{22}{23}$
2. Prove that this scheme is 2-time secure

Part (a): Correctness Probability is $\frac{22}{23}$

Claim: For any $m \in \mathbb{Z}_{23}$, $\Pr[\text{Dec}(K, \text{Enc}(K, m)) = m] = \frac{22}{23}$.

Proof:

The key $K = (a, b)$ is sampled uniformly from $\mathbb{Z}_{23} \times \mathbb{Z}_{23}$.

Given message m:
1. Encryption computes: $c = a \cdot m + b \bmod 23$
2. Decryption computes: $m' = (c - b) \cdot a^{-1} = (a \cdot m + b - b) \cdot a^{-1} = a \cdot m \cdot a^{-1} = m$

> **But wait!** Decryption requires a^{-1} to exist, which happens if and only if $\gcd(a, 23) = 1$.
>
> Since 23 is prime, a^{-1} exists $\Leftrightarrow a \neq 0$.

Probability analysis:

$$\Pr[\text{Dec}(K, \text{Enc}(K, m)) = m] = \Pr[a \neq 0]$$

Since a is sampled uniformly from $\mathbb{Z}_{23} = \{0, 1, 2, ..., 22\}$:

$$\Pr[a \neq 0] = \frac{22}{23}$$

> **Conclusion:** $\Pr[\text{Dec}(K, \text{Enc}(K, m)) = m] = \frac{22}{23}$ ✓

Note: When $a = 0$, we have $c = b$ regardless of m, and decryption fails because 0 has no multiplicative inverse in $\mathbb{Z}_{23}$.

Part (b): The Scheme is 2-Time Secure

Claim: The encryption scheme is 2-time perfectly secure.

Proof:

We need to show that for any (m_1, m_2) and (m_1', m_2') with $m_1 \neq m_2$ and $m_1' \neq m_2'$, and any c_1, c_2:

$$\Pr[\text{Enc}(K, m_1) = c_1 \wedge \text{Enc}(K, m_2) = c_2] = \Pr[\text{Enc}(K, m_1') = c_1 \wedge \text{Enc}(K, m_2') = c_2]$$

Setting up the equations:

For the key (a, b), the event $\{\text{Enc}(K, m_1) = c_1 \wedge \text{Enc}(K, m_2) = c_2\}$ means:

$$a \cdot m_1 + b \equiv c_1 \bmod 23$$
$$a \cdot m_2 + b \equiv c_2 \bmod 23$$

Solving for (a, b):

Subtracting the equations:

$$a \cdot (m_1 - m_2) \equiv c_1 - c_2 \bmod 23$$

Since $m_1 \neq m_2$ and 23 is prime, $(m_1 - m_2)$ has a multiplicative inverse in $\mathbb{Z}_{23}$.

Therefore:

$$a \equiv (c_1 - c_2) \cdot (m_1 - m_2)^{-1} \bmod 23$$

Once a is determined, b is uniquely determined:

$$b \equiv c_1 - a \cdot m_1 \bmod 23$$

Key observation: For any c_1, c_2 and any pair (m_1, m_2) with $m_1 \neq m_2$, there exists **exactly one** key $(a, b) \in \mathbb{Z}_{23} \times \mathbb{Z}_{23}$ such that:

$$\text{Enc}((a, b), m_1) = c_1 \text{ and } \text{Enc}((a, b), m_2) = c_2$$

Computing the probability:

$$\Pr[\text{Enc}(K, m_1) = c_1 \wedge \text{Enc}(K, m_2) = c_2] = \frac{\text{number of valid keys}}{\text{total keys}} = \frac{1}{23^2}$$

This probability is $\frac{1}{23^2}$ regardless of the choice of (m_1, m_2) (as long as $m_1 \neq m_2$)!

Applying to both message pairs:

For (m_1, m_2) with $m_1 \neq m_2$:

$$\Pr[\text{Enc}(K, m_1) = c_1 \wedge \text{Enc}(K, m_2) = c_2] = \frac{1}{23^2}$$

For (m_1', m_2') with $m_1' \neq m_2'$:

$$\Pr[\text{Enc}(K, m_1') = c_1 \wedge \text{Enc}(K, m_2') = c_2] = \frac{1}{23^2}$$

Since both probabilities equal $\frac{1}{23^2}$, the scheme is 2-time perfectly secure. □

Intuition: Why 2-Time but Not 3-Time?

The affine cipher $c = a \cdot m + b$ has 2 unknowns: a and b.

- **With 1 ciphertext:** 1 equation, 2 unknowns → many solutions → 1-time secure
- **With 2 ciphertexts:** 2 equations, 2 unknowns → unique solution → 2-time secure
- **With 3 ciphertexts:** 3 equations, 2 unknowns → overdetermined → reveals key structure!

Messages	Security
1 message	Perfectly secure (like OTP with 2-part key)
2 messages	Still perfectly secure (equations = unknowns)
3+ messages	NOT secure — the key is uniquely determined and can be checked

Analogy: This is similar to Shamir's secret sharing with threshold 2 — any 2 points determine a line, but 1 point reveals nothing about it.

◆ ◆ ◆

Problem 11: Perfect Secrecy for Variable-Length Messages

Setting: The message space is $\mathcal{M} = \{0,1\}^{\leq \ell}$, the set of all nonempty binary strings of length at most ℓ.

Tasks:

1. Consider the encryption scheme where Gen chooses a uniform key from $\mathcal{K} = \{0,1\}^\ell$, and $\text{Enc}_k(m) = k_{|m|} \oplus m$, where k_t denotes the first t bits of k. Show that this scheme is **not** perfectly secret for message space $\mathcal{M}$.
2. Design a perfectly secret encryption scheme for message space $\mathcal{M}$.

Part (a): The Scheme is NOT Perfectly Secret

The Encryption Scheme:

- **Key space:** $\mathcal{K} = \{0,1\}^\ell$
- **Key generation:** Sample $k \leftarrow \{0,1\}^\ell$ uniformly
- **Encryption:** $\text{Enc}_k(m) = k_{|m|} \oplus m$ (use first $|m|$ bits of k, XOR with m)

> **Claim:** This scheme is **NOT** perfectly secret.

Proof by counterexample:

Perfect secrecy requires that for all $m_0, m_1 \in \mathcal{M}$ and all ciphertexts c:

$$\Pr[\text{Enc}_K(m_0) = c] = \Pr[\text{Enc}_K(m_1) = c]$$

We will find m_0, m_1, and c that violate this.

Counterexample:

Let:

- $m_0 = 0$ (a single bit message of length 1)
- $m_1 = 00$ (a two-bit message of length 2)
- $c = 0$ (a single bit ciphertext of length 1)

Compute $\Pr[\text{Enc}_K(m_0) = c]$:

$\text{Enc}_k(m_0) = \text{Enc}_k(0) = k_1 \oplus 0 = k_1$ (the first bit of k)

For this to equal $c = 0$, we need $k_1 = 0$.

$$\Pr[\text{Enc}_K(0) = 0] = \Pr[k_1 = 0] = \frac{1}{2}$$

Compute $\Pr[\text{Enc}_K(m_1) = c]$:

$\text{Enc}_k(m_1) = \text{Enc}_k(00) = k_2 \oplus 00 = k_2$ (the first 2 bits of k)

The ciphertext is a 2-bit string. For this to equal $c = 0$ (a 1-bit string):

> **The ciphertext lengths don't match!**
>
> $\text{Enc}_k(00)$ produces a 2-bit output, but $c = 0$ is a 1-bit string.
>
> Therefore: $\Pr[\text{Enc}_K(00) = 0] = 0$

Conclusion:

$$\Pr[\text{Enc}_K(0) = 0] = \frac{1}{2} \neq 0 = \Pr[\text{Enc}_K(00) = 0]$$

> **The scheme is NOT perfectly secret because the ciphertext length reveals the message length!**

An adversary seeing a ciphertext of length t knows the message has exactly t bits.

□

Part (b): A Perfectly Secret Scheme for $\mathcal{M}$

Goal: Design an encryption scheme that hides both the message content AND the message length.

11.2.1. Key Idea: Pad All Messages to Maximum Length

Strategy:
1. Encode the message with its length
2. Pad to fixed length ℓ
3. Use one-time pad encryption

11.2.2. Construction

Key Space: $\mathcal{K} = \{0,1\}^{\ell+\lceil \log_2 \ell \rceil}$

Gen: Sample $k \leftarrow \mathcal{K}$ uniformly. Parse $k = k_{\text{pad}} \parallel k_{\text{len}}$ where:
- $k_{\text{pad}} \in \{0,1\}^{\ell}$ (for padding the message)
- $k_{\text{len}} \in \{0,1\}^{\lceil \log_2 \ell \rceil}$ (for hiding the length)

Encryption $\text{Enc}_k(m)$:
1. Let $t = |m|$ be the message length
2. Pad m to length ℓ: $m' = m \parallel 0^{\ell - t}$
3. Encode length: $t' = t \oplus k_{\text{len}}$
4. Encrypt: $c = m' \oplus k_{\text{pad}}$
5. Output (c, t')

Decryption $\text{Dec}_k(c, t')$:
1. Recover length: $t = t' \oplus k_{\text{len}}$
2. Decrypt: $m' = c \oplus k_{\text{pad}}$
3. Output first t bits of m'

11.2.3. Proof of Perfect Secrecy

Claim: This scheme is perfectly secret for $\mathcal{M} = \{0,1\}^{\leq \ell}$.

Proof:

For any two messages $m_0, m_1 \in \mathcal{M}$ and any ciphertext (c, t'):

$$\begin{aligned}
\Pr[\text{Enc}_K(m_0) = (c,t')] &= \Pr\left[m_0 \parallel 0^{\ell - |m_0|} \oplus k_{\text{pad}} = c \text{ and } |m_0| \oplus k_{\text{len}} = t'\right] \\
&= \Pr\left[k_{\text{pad}} = c \oplus \left(m_0 \parallel 0^{\ell - |m_0|}\right)\right] \times \Pr[k_{\text{len}} = t' \oplus |m_0|] \\
&= \frac{1}{2^{\ell}} \times \frac{1}{2^{\lceil \log_2 \ell \rceil}}
\end{aligned}$$

Similarly:

$$\Pr[\text{Enc}_K(m_1) = (c,t')] = \frac{1}{2^{\ell}} \times \frac{1}{2^{\lceil \log_2 \ell \rceil}}$$

Both probabilities are equal and independent of the message!

Therefore, the scheme is perfectly secret. □

11.2.4. Alternative Simpler Construction

If ciphertext length is allowed to be fixed at $\ell + \lceil \log_2 \ell \rceil$:

Simpler version:

- **Key:** $k \in \{0,1\}^{\ell + \lceil \log_2 \ell \rceil}$
- **Encryption:** $\mathrm{Enc}_k(m) = k \oplus \left(m \parallel 0^{\ell - |m|} \parallel \mathrm{bin}(|m|)\right)$

where $\mathrm{bin}(|m|)$ is the binary encoding of the length.

This directly XORs the (padded message + length encoding) with the full key.

Original Scheme (NOT secure)	**Fixed Scheme (Secure)**
Ciphertext length = message length	Ciphertext length = fixed $(\ell + O(\log \ell))$
Leaks message length	Hides message length
Key size: ℓ bits	Key size: $\ell + \lceil \log_2 \ell \rceil$ bits

◆ ◆ ◆

Problem 12: PRF or Not?

Setup: Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a pseudorandom function (PRF). For each construction f' below, either prove that f' is a PRF (for all choices of f), or prove that f' is not a PRF.

1. $f'_k(x) := f_k(0 \parallel x) \parallel f_k(1 \parallel x)$
2. $f'_k(x) := f_k(0 \parallel x) \parallel f_k(x \parallel 1)$

Background: PRF Definition

Pseudorandom Function (PRF): A keyed function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a PRF if no efficient adversary can distinguish between:

- Oracle access to $f_k(\cdot)$ for random k
- Oracle access to a truly random function $R : \{0,1\}^n \to \{0,1\}^n$

Formally: For all PPT distinguishers D:

$$\left| \Pr\left[D^{f_k(\cdot)} = 1\right] - \Pr\left[D^{R(\cdot)} = 1\right] \right| \leq \mathrm{negl}(n)$$

Part (a): $f'_k(x) := f_k(0 \parallel x) \parallel f_k(1 \parallel x)$

Note: Here the input x has length $n-1$ bits (so that $0 \parallel x$ and $1 \parallel x$ are n bits each).

The output is $2n$ bits (concatenation of two n-bit values).

Answer: YES, f' is a PRF.

Proof by reduction:

Suppose f' is not a PRF. Then there exists a PPT distinguisher D' that can distinguish f'_k from a random function $R' : \{0,1\}^{n-1} \to \{0,1\}^{2n}$ with non-negligible advantage.

We construct a distinguisher D that breaks f:

Distinguisher D with oracle access to O (either f_k or random R):

1. When D' queries $x \in \{0,1\}^{n-1}$:
   - Query $O(0 \parallel x)$ to get y_0
   - Query $O(1 \parallel x)$ to get y_1
   - Return $y_0 \parallel y_1$ to D'
2. Output whatever D' outputs

Analysis:

Case 1: $O = f_k$ (real PRF)

D simulates f'_k perfectly:

$$D'\text{'s view} = f_k(0 \parallel x) \parallel f_k(1 \parallel x) = f'_k(x)$$

Case 2: $O = R$ (random function)

D simulates a random function on pairs of inputs:
- $R(0 \parallel x)$ and $R(1 \parallel x)$ are independent random values
- Their concatenation is uniformly random in $\{0,1\}^{2n}$
- Different inputs $x \neq x'$ give independent outputs

Key observation: The inputs $0 \parallel x$ and $1 \parallel x$ are **always distinct** (they differ in the first bit), so $R(0 \parallel x)$ and $R(1 \parallel x)$ are independent.

Furthermore, inputs $0 \parallel x$ and $0 \parallel x'$ for $x \neq x'$ are distinct, so all queries produce independent random outputs.

Therefore:

$$\Pr\left[D^{f_k} = 1\right] = \Pr\left[D'^{f'_k} = 1\right]$$

$$\Pr[D^R = 1] = \Pr\left[D'^{R'} = 1\right]$$

If D' has non-negligible advantage, so does D. This contradicts that f is a PRF. □

Part (b): $f'_k(x) := f_k(0 \parallel x) \parallel f_k(x \parallel 1)$

Note: Here the input x has length $n-1$ bits (so that $0 \parallel x$ and $x \parallel 1$ are n bits each).

Answer: NO, f' is NOT necessarily a PRF.

Counterexample:

Consider what happens when we query specific related inputs.

The attack:

Query f' on two specific inputs:
- $x_1 = 0^{n-2} \parallel 1$ (i.e., 00...01)
- $x_2 = 1 \parallel 0^{n-2}$ (i.e., 10...00)

Compute the corresponding PRF queries:

For $x_1 = 0^{n-2}1$:
- $0 \parallel x_1 = 0 \parallel 0^{n-2} \parallel 1 = 0^{n-1} \parallel 1$
- $x_1 \parallel 1 = 0^{n-2} \parallel 1 \parallel 1 = 0^{n-2} \parallel 11$

For $x_2 = 1 \parallel 0^{n-2}$:
- $0 \parallel x_2 = 0 \parallel 1 \parallel 0^{n-2} = 01 \parallel 0^{n-2}$
- $x_2 \parallel 1 = 1 \parallel 0^{n-2} \parallel 1 = 1 \parallel 0^{n-2} \parallel 1$

Hmm, these don't immediately collide. Let me reconsider…

Better attack — finding a collision:

Let x_1 such that $0 \parallel x_1 = x_2 \parallel 1$ for some x_2.

This means: $x_2 = 0 \parallel x_1[1..n-2]$ and $x_1[n-1] = 1$.

Specifically, if $x_1 = a \parallel 1$ for some $a \in \{0,1\}^{n-2}$, then:

$$0 \parallel x_1 = 0 \parallel a \parallel 1$$

And if $x_2 = 0 \parallel a$, then:

$$x_2 \parallel 1 = 0 \parallel a \parallel 1$$

Collision found!

For $x_1 = a \parallel 1$ and $x_2 = 0 \parallel a$ (where $a \in \{0,1\}^{n-2}$):

$$0 \parallel x_1 = x_2 \parallel 1$$

This means $f_k(0 \parallel x_1) = f_k(x_2 \parallel 1)$.

The distinguishing attack:

Distinguisher D:

1. Choose any $a \in \{0,1\}^{n-2}$
2. Let $x_1 = a \parallel 1$ and $x_2 = 0 \parallel a$
3. Query $f'(x_1) = f_{k(0 \parallel x_1)} \parallel f_{k(x_1 \parallel 1)}$ — call the first half A
4. Query $f'(x_2) = f_{k(0 \parallel x_2)} \parallel f_{k(x_2 \parallel 1)}$ — call the second half B
5. Check if $A = B$ (i.e., first half of $f'(x_1)$ equals second half of $f'(x_2)$)
6. If yes, output "real PRF"; if no, output "random"

Analysis:

If f' is built from f:

We have $0 \parallel x_1 = x_2 \parallel 1$ (by construction), so:

$$f_k(0 \parallel x_1) = f_k(x_2 \parallel 1)$$

Therefore $A = B$ **always**. The check passes with probability 1.

If f' is a truly random function:

The first half of $f'(x_1)$ and the second half of $f'(x_2)$ are independent random n-bit strings.

$$\Pr[A = B] = \frac{1}{2^n}$$

Distinguishing advantage:

$$|1 - \frac{1}{2^n}| = 1 - \text{negl}(n) \approx 1$$

This is overwhelming! The construction is completely broken. □

Summary

Part	Construction	PRF?	Reason
(a)	$f_k(0\|\|x) \parallel f_k(1\|\|x)$	**YES**	Inputs always differ in first bit → no collisions
(b)	$f_k(0\|\|x) \parallel f_k(x\|\|1)$	**NO**	Collision: $0\|\|x_1 = x_2\|\|1$ is exploitable

Key Insight: When constructing PRFs from PRFs, ensure that the internal queries **never collide** for different external inputs. In part (a), prepending 0 and 1 guarantees distinct inputs. In part (b), the overlap between "prepend 0" and "append 1" creates exploitable collisions.

◆ ◆ ◆

Problem 13: Weakly-Secure PRF and Counter Mode

Setup: Let F be a PRF defined over $(\mathcal{K}, X, Y)$, where $X = \{0, ..., N-1\}$ and $Y = \{0,1\}^n$, where N is super-polynomial. For poly-bounded $\ell \geq 1$, consider the PRF F' defined over $(\mathcal{K}, X, Y^\ell)$ as follows:

$$F'(k, x) := (F(k, x), F(k, x+1 \bmod N), ..., F(k, x+\ell-1 \bmod N))$$

Tasks:
1. Show that F' is a weakly-secure PRF.
2. Prove that randomized counter mode is CPA secure.

Background: Weakly-Secure PRF

Weakly-Secure PRF: A PRF $F : \mathcal{K} \times X \to Y$ is **weakly-secure** if it is indistinguishable from a random function when the adversary is restricted to querying on **distinct, uniformly random** inputs (rather than adversarially chosen inputs).

This is weaker than standard PRF security, but sufficient for many applications.

Part (a): F' is a Weakly-Secure PRF

Claim: If F is a PRF, then F' is a weakly-secure PRF.

Proof by reduction:

Suppose F' is not weakly-secure. Then there exists a PPT distinguisher D that can distinguish F'_k from a random function $R : X \to Y^\ell$ with non-negligible advantage, even when queries are uniformly random and distinct.

We construct a distinguisher D_F that breaks the PRF security of F:

Distinguisher D_F with oracle access to O (either F_k or random R_F):

1. When D queries on random distinct input $x \in X$:
   - Query $O(x), O(x+1 \bmod N), ..., O(x+\ell-1 \bmod N)$
   - Return $(O(x), O(x+1), ..., O(x+\ell-1))$ to D
2. Output whatever D outputs

Analysis:

Case 1: $O = F_k$ (real PRF)

D_F perfectly simulates $F'_k(x)$ for each query.

Case 2: $O = R_F$ (random function)

Each $R_F(x+i \bmod N)$ is an independent random value in $\{0,1\}^n$.

The key question: Does the output look like a random function over Y^ℓ?

> **Potential collision issue:** If two queries x and x' are such that their ranges overlap, i.e., $x + i \equiv x' + j \bmod N$ for some $i, j \in \{0, ..., \ell-1\}$, then the outputs share a common value.
>
> A truly random function $R : X \to Y^\ell$ would have independent outputs, but our simulation produces correlated outputs when ranges overlap.

Why this is okay for weak security:

Since queries are **random and distinct**, and N is **super-polynomial**:

$$\Pr[\text{two ranges overlap}] \leq \frac{q^2 \cdot \ell^2}{N}$$

where q is the number of queries (polynomial). Since N is super-polynomial and ℓ is polynomial, this probability is **negligible**.

> **With overwhelming probability, all query ranges are disjoint.**

When ranges are disjoint, the outputs are independent random values, perfectly matching a random function.

Conclusion:

$$|\Pr\left[D_F^{F_k} = 1\right] - \Pr\left[D_F^{R_F} = 1\right]| \geq |\Pr\left[D^{F'_k} = 1\right] - \Pr[D^R = 1]| - \text{negl}(n)$$

If D has non-negligible advantage, so does D_F, contradicting that F is a PRF. □

Part (b): Randomized Counter Mode is CPA Secure

13.3.1. Randomized Counter Mode (CTR) Construction

Encryption scheme $\mathcal{E} = (\text{Gen}, \text{Enc}, \text{Dec})$:

- **Gen:** Sample $k \leftarrow \mathcal{K}$ uniformly
- **Encryption** $\text{Enc}_k(m_1, ..., m_\ell)$: where each $m_i \in \{0,1\}^n$
  1. Sample random starting point $r \leftarrow X = \{0, ..., N-1\}$
  2. Compute $c_i = m_i \oplus F(k, r + i - 1 \bmod N)$ for $i = 1, ..., \ell$
  3. Output $(r, c_1, ..., c_\ell)$
- **Decryption** $\text{Dec}_k(r, c_1, ..., c_\ell)$:
  1. Compute $m_i = c_i \oplus F(k, r + i - 1 \bmod N)$ for $i = 1, ..., \ell$
  2. Output $(m_1, ..., m_\ell)$

13.3.2. CPA Security Proof

Claim: If F is a PRF and N is super-polynomial, then randomized counter mode is CPA-secure.

Proof:

We prove this using a hybrid argument.

Hybrid 0 (Real world): Adversary interacts with Enc_k using the real PRF F_k.

Hybrid 1: Replace F_k with a truly random function $R : X \to \{0,1\}^n$.

Hybrid 2 (Ideal world): Ciphertexts are completely random.

Hybrid 0 ≈ Hybrid 1:

By PRF security of F, no PPT adversary can distinguish F_k from a random function R.

Any distinguisher between Hybrid 0 and Hybrid 1 can be converted to a PRF distinguisher for F.

Hybrid 1 ≈ Hybrid 2:

In Hybrid 1, for each encryption query with random r:

$$c_i = m_i \oplus R(r + i - 1)$$

Since R is a random function and r is random:
- Each $R(r + i - 1)$ is uniformly random (if not queried before)
- The key question: Are the values $R(r), R(r+1), ..., R(r+\ell-1)$ fresh?

Collision analysis: What's the probability that two encryption queries have overlapping counter ranges?

For q encryption queries, each using ℓ consecutive counters starting from random r_j:

$$\Pr[\text{some overlap}] \leq \frac{q^2 \cdot \ell^2}{N} = \text{negl}(n)$$

since N is super-polynomial.

Conditioning on no collisions:

When counter ranges don't overlap:

- Each $R(r_j + i)$ is an independent random value
- $c_i = m_i \oplus R(r_j + i)$ is uniformly random (one-time pad!)
- The ciphertext reveals nothing about the message

In the no-collision case, CTR mode is perfectly secret!

Since collisions happen with negligible probability, Hybrid 1 $\approx$ Hybrid 2.

CPA Security Conclusion:

By the hybrid argument:

$$\text{Adv}_{\text{CPA}(A)} \leq \text{Adv}_{\text{PRF}(B)} + \frac{q^2 \ell^2}{N}$$

Both terms are negligible, so the scheme is CPA-secure. □

Summary

Result	Key Insight
F' is weakly-secure PRF	Random, distinct queries cause disjoint counter ranges with overwhelming probability
CTR mode is CPA-secure	Random nonces + super-poly domain size → negligible collision probability → OTP-like security

Key Takeaway: The super-polynomial size of N is crucial! It ensures that even with polynomially many queries, the probability of counter range overlaps is negligible. This allows us to treat each encryption as if it uses fresh random pad values.

◆ ◆ ◆

Problem 14: Hard-Core Predicates and Universality

Background: A predicate $b : \{0,1\}^* \to \{0,1\}$ is a **hard-core predicate** of a one-way function $f(\cdot)$ if $b(x)$ is efficiently computable given x, and there exists a negligible function ν such that for every PPT adversary A and every n:

$$\Pr[A(f(x)) = b(x)] \leq \frac{1}{2} + \nu(n)$$

Tasks:
1. Construct a one-way function $f : \{0,1\}^n \to \{0,1\}^m$ for input $x = (x_1, x_2, ..., x_n)$ such that the first bit x_1 is **not** hardcore.
2. Prove that there is **no universal** hardcore predicate (i.e., no single predicate b that is hardcore for every OWF).
3. Construct a one-way function f for which **none** of the individual input bits $b_i(x_1, ..., x_n) = x_i$ are hardcore.

Part (a): A OWF Where the First Bit is NOT Hardcore

Goal: Construct f such that f is one-way but the predicate $b(x) = x_1$ can be predicted from $f(x)$.

14.1.1. Construction

Let $g : \{0,1\}^{n-1} \to \{0,1\}^m$ be any one-way function. Define:

$$f(x_1, x_2, ..., x_n) = x_1 \parallel g(x_2, ..., x_n)$$

That is, f outputs the first bit unchanged, followed by the one-way function applied to the remaining bits.

14.1.2. Proof that f is One-Way

Suppose f is not one-way. Then there exists PPT A such that:

$$\Pr[A(f(x)) \in f^{-1}(f(x))] > \text{non-negl}(n)$$

Given A, we construct an inverter B for g:

> **Inverter B for g:**
>
> On input $y = g(x_2, ..., x_n)$:
> 1. Pick random $b \in \{0,1\}$
> 2. Compute $A(b \parallel y)$ to get $(b', x_2', ..., x_n')$
> 3. Output $(x_2', ..., x_n')$

If A successfully inverts f, then $g(x_2', ..., x_n') = y$, which means B inverts g.

Since g is one-way, A cannot succeed with non-negligible probability. Therefore f is one-way. ✓

14.1.3. Proof that x_1 is NOT Hardcore

An adversary can predict x_1 from $f(x)$ with probability **1**!

Attack: Given $f(x) = x_1 \parallel g(x_2, ..., x_n)$, simply output the first bit.

$$\Pr[A(f(x)) = x_1] = 1 \gg \frac{1}{2} + \text{negl}(n)$$

Conclusion: The first bit x_1 is trivially computable from $f(x)$, so it is **not** a hardcore predicate. □

Part (b): No Universal Hardcore Predicate Exists

Definition: A predicate $b : \{0,1\}^n \to \{0,1\}$ is **universal hardcore** if it is hardcore for **every** one-way function.

Claim: There is no universal hardcore predicate.

Proof:

Let $b : \{0,1\}^n \to \{0,1\}$ be any polynomial-time computable predicate. We will construct a OWF f for which b is **not** hardcore.

Let $g : \{0,1\}^n \to \{0,1\}^m$ be any one-way function (we assume OWFs exist).

Construction:

$$f(x) = b(x) \parallel g(x)$$

The output of f is the predicate value $b(x)$ concatenated with the OWF output $g(x)$.

***f* is one-way:** Same argument as Part (a) — inverting f requires inverting g.

***b* is not hardcore for *f*:**

Given $f(x) = b(x) \parallel g(x)$, an adversary can compute $b(x)$ by simply reading the first bit of $f(x)$.

$$\Pr[A(f(x)) = b(x)] = 1$$

Conclusion: For any predicate b, we can construct a OWF that "leaks" $b(x)$ in its output. Therefore, no universal hardcore predicate exists. □

Part (c): A OWF Where NO Individual Bit is Hardcore

Goal: Construct $f : \{0,1\}^n \to \{0,1\}^m$ such that for all $i \in \{1, ..., n\}$, the predicate $b_i(x) = x_i$ is **not** hardcore.

14.3.1. Construction

Let $g : \{0,1\}^n \to \{0,1\}^m$ be any one-way function. Define:

$$f(x_1, ..., x_n) = x_1 \parallel x_2 \parallel ... \parallel x_n \parallel g(x_1, ..., x_n)$$

$$f(x) = x \parallel g(x)$$

The output is simply x (the entire input) followed by $g(x)$.

14.3.2. f is One-Way

Suppose adversary A inverts f with non-negligible probability:

$$\Pr[f(A(f(x))) = f(x)] > \text{non-negl}(n)$$

Given $f(x) = x \parallel g(x)$, if A outputs x', then:

$$f(x') = x' \parallel g(x') = x \parallel g(x)$$

This requires $x' = x$ and $g(x') = g(x)$.

> **Wait — there's an issue!** Given $f(x) = x \parallel g(x)$, the adversary can read x directly from the output!
>
> So f as defined is **not** one-way.

14.3.3. Corrected Construction

We need a more subtle approach. Use a one-way **permutation** and reveal the input:

Let $\pi : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation.

Define $f : \{0,1\}^{2n} \to \{0,1\}^{2n}$ by:

$$f(x, y) = (x, \pi(x) \oplus y)$$

where $x, y \in \{0,1\}^n$.

f is one-way:

Given $(x, \pi(x) \oplus y)$:

- x is known
- Computing y requires computing $\pi(x)$, but we have $\pi(x) \oplus y$
- To find y, we need $\pi(x)$, which requires computing $\pi(x)$ from x — easy!

Hmm, this also doesn't work. Let me try another approach.

14.3.4. Working Construction

Let $g : \{0,1\}^n \to \{0,1\}^{2n}$ be a length-doubling OWF.

$$f(x) = g(x) \oplus (x \parallel x)$$

The output XORs $g(x)$ with the input concatenated with itself.

Actually, the simplest working construction:

Final Construction:

Let $g : \{0,1\}^n \to \{0,1\}^m$ be a one-way function.

$$f(x) = (x, g(x))$$

But this reveals x, so it's not one-way in the usual sense…

14.3.5. Correct Approach: The Key Insight

The question asks for a OWF where **none of the input bit predicates** are hardcore. The trick is:

Key insight: Even though no **single** bit is hardcore, the function can still be one-way because recovering the **full** preimage might still be hard.

A function $f(x) = x \parallel g(x)$ reveals x, so it's **not one-way**.

However, we can construct a OWF where each bit can be **guessed** with probability $> \frac{1}{2} + \text{non-negl}$ but not with probability 1.

Correct Construction:

Let $g : \{0,1\}^n \to \{0,1\}^m$ be a OWF. Define:

$$f(x_1, ..., x_n) = g(x) \parallel (x_1 \oplus x_2) \parallel (x_2 \oplus x_3) \parallel ... \parallel (x_{n-1} \oplus x_n)$$

The output contains $g(x)$ plus all **consecutive XORs** of bits.

Each bit is not hardcore:

Given the XORs $(x_1 \oplus x_2), (x_2 \oplus x_3), ..., (x_{n-1} \oplus x_n)$:
- If we guess x_1, we can compute all other bits!
- So each x_i can be computed given a single bit guess.

An adversary can:
1. Guess $x_1 = 0$ or $x_1 = 1$ (50% chance of being correct)
2. Compute all other x_i from the XORs
3. Output the guess for any x_i

This gives $\Pr[\text{guess } x_i] = \frac{1}{2}$... which is still just random guessing.

Better construction — using parity leak:

$$f(x) = g(x) \parallel (x_1 \oplus r_1) \parallel (x_2 \oplus r_2) \parallel ... \parallel (x_n \oplus r_n) \parallel r_1 \parallel ... \parallel r_n$$

Wait, this reveals everything.

The correct answer:

$$f(x) = x \parallel g(x) \text{ where } g : \{0,1\}^n \to \{0,1\}^m \text{ is length-preserving OWF}$$

This is NOT one-way (since x is revealed). The question may be asking for something subtler, or there's a specific construction involving the Goldreich-Levin theorem.

Alternative interpretation: The question may be showing that even though **individual** bits aren't hardcore, the **Goldreich-Levin inner product** $\langle x, r \rangle$ is still hardcore for any OWF.

□

◆ ◆ ◆

Problem 15: Breaking CPA Security with Polynomial Queries

Setup: Suppose $(\text{Gen}, \text{Enc}, \text{Dec})$ is a CPA-secure encryption scheme that encrypts messages belonging to a field $\mathbb{F}$. Construct a new encryption scheme as follows:

- $\text{Gen}_1(1^n)$: Sample $k' \leftarrow \text{Gen}(1^n)$, then sample p, a random degree-d polynomial over $\mathbb{F}$. The key is $k = (k', p)$.
- $\text{Enc}_1(k, m) = \text{Enc}(k', m) \parallel p(m)$
- $\text{Dec}_1(k, c)$: Runs $\text{Dec}(k', \cdot)$ on the first part of the ciphertext.

Question: In the CPA security experiment, what is the minimum number of queries to the Enc oracle needed to break the CPA security of the scheme $(\text{Gen}_1, \text{Enc}_1, \text{Dec}_1)$?

Analysis of the Scheme

15.1.1. The Vulnerability

The new encryption appends $p(m)$ to the ciphertext, where p is a secret degree-d polynomial.

> **Key observation:** A degree-d polynomial over $\mathbb{F}$ is uniquely determined by $d+1$ points!
>
> If the adversary learns $p(m)$ for $d+1$ distinct messages m, they can fully recover p via **Lagrange interpolation**.

15.1.2. The Attack Strategy

Attack to recover the polynomial p:

1. Query the encryption oracle on $d+1$ distinct messages: $m_0, m_1, ..., m_d \in \mathbb{F}$
2. From each ciphertext $c_i = \text{Enc}(k', m_i) \parallel p(m_i)$, extract $p(m_i)$ (the second part)
3. Use Lagrange interpolation to recover p from the $d+1$ point-value pairs $(m_i, p(m_i))$

15.1.3. Using the Recovered Polynomial

Once p is known, the adversary can break CPA security:

CPA Attack:

1. Choose distinct challenge messages $m_0^*, m_1^* \in \mathbb{F}$ (different from query messages)
2. Receive challenge ciphertext $c^* = \text{Enc}(k', m_b^*) \parallel p(m_b^*)$
3. Compute $p(m_0^*)$ and $p(m_1^*)$ using the recovered polynomial
4. Compare the second part of c^* with $p(m_0^*)$ and $p(m_1^*)$
5. Output $b' = 0$ if it matches $p(m_0^*)$, else output $b' = 1$

Success probability: 1 (deterministic!)

Minimum Number of Queries

> **Answer: $d+1$ queries are necessary and sufficient.**

15.2.1. Why $d+1$ is Sufficient

With $d+1$ queries, the adversary obtains $d+1$ point-value pairs, which uniquely determine the degree-d polynomial p. The attack above then succeeds with probability 1.

15.2.2. Why $d+1$ is Necessary

Claim: With only d queries, the scheme remains CPA-secure.

Proof sketch:

With d point-value pairs $(m_1, p(m_1)), ..., (m_d, p(m_d))$, infinitely many degree-d polynomials are consistent with the data. Specifically, for any value $v \in \mathbb{F}$ and any new point m^*, there exists exactly one polynomial p of degree d such that:

- $p(m_i) =$ observed values for $i = 1, ..., d$
- $p(m^*) = v$

This means $p(m_0^*)$ and $p(m_1^*)$ are **uniformly random** from the adversary's perspective (conditioned on the d queries), giving no advantage.

This is analogous to **Shamir's secret sharing** with threshold $d+1$: any d shares reveal nothing about the secret, but $d+1$ shares reveal everything.

Summary

Polynomial Degree	Minimum Queries to Break
d	$d+1$

Key Insight: The scheme leaks $p(m)$ for each encryption. Since a degree-d polynomial has $d+1$ coefficients (degrees of freedom), exactly $d+1$ evaluations are needed to determine it. With fewer queries, the polynomial remains information-theoretically hidden.

◆ ◆ ◆

Problem 16: Constructing Primitives from Other Primitives

Task: Give complete details of how to construct Y from X where:

1. $X =$ One-way permutation, $Y =$ Pseudorandom generator
2. $X =$ Pseudorandom generator, $Y =$ One-way function
3. $X =$ Pseudorandom generator, $Y =$ Pseudorandom function

Part (a): One-Way Permutation → PRG

Goal: Construct a PRG from a one-way permutation (OWP).

16.1.1. The Blum-Micali / Goldreich-Levin Construction

Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation.

Hard-core predicate (Goldreich-Levin): For any OWP f, the inner product:

$$b(x, r) = \langle x, r \rangle = \bigoplus_{i=1}^{n} x_i \cdot r_i \bmod 2$$

is a hard-core predicate for $f'(x, r) = (f(x), r)$.

16.1.2. PRG Construction

PRG $G : \{0,1\}^n \to \{0,1\}^{n+1}$:

$$G(s) = f(s) \parallel b(s)$$

where $b(s)$ is a hard-core bit for f (e.g., the parity of s if f is appropriately structured, or use Goldreich-Levin with a public random r).

Output: $n + 1$ bits from an n-bit seed $\to$ expansion by 1 bit.

16.1.3. Why This is a PRG

Proof sketch:

Suppose distinguisher D can distinguish $G(U_n)$ from U_{n+1}.

- $G(U_n) = (f(U_n), b(U_n))$
- $U_{n+1} = (U_n, U_1)$ (independent uniform bits)

Since f is a permutation, $f(U_n)$ is uniform over $\{0,1\}^n$.

The only difference is: Is the last bit $b(U_n)$ or independent random?

By the hard-core property, $b(s)$ cannot be predicted from $f(s)$ better than $\frac{1}{2} + \text{negl}(n)$.

Therefore, D cannot distinguish $\Rightarrow$ G is a PRG. $\square$

16.1.4. Extending to More Bits

To get $G : \{0,1\}^n \to \{0,1\}^{n+k}$ for polynomial k:

Iterate:

$$s_0 = s$$

$s_{i+1} = f(s_i)$, output bit $b_i = b(s_i)$

Final output: $s_k \parallel b_1 \parallel b_2 \parallel \ldots \parallel b_k$

Part (b): PRG $\to$ One-Way Function

Goal: Construct a OWF from a PRG.

16.2.1. Construction

Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a length-doubling PRG.

One-way function f:

$$f(s) = G(s)$$

Simply use the PRG as the one-way function!

16.2.2. Proof that f is One-Way

Claim: If G is a PRG, then $f(s) = G(s)$ is a one-way function.

Proof by contradiction:

Suppose f is not one-way. Then there exists PPT inverter I such that:

$$\Pr[f(I(f(s))) = f(s)] > \text{non-negl}(n)$$

We construct a PRG distinguisher D:

Distinguisher D:

On input $y \in \{0,1\}^{2n}$:
1. Run $s' = I(y)$
2. If $G(s') = y$, output "PRG" (i.e., $y = G(s)$ for some s)
3. Else output "random"

Analysis:

- If $y = G(s)$ for random s: I succeeds with non-negligible probability, so D outputs "PRG"
- If y is truly random: $y \in \text{Image}(G)$ with probability $|\{0,1\}^n| \over |\{0,1\}^{2n}| = \frac{2^n}{2^{2n}} = 2^{-n}$ (negligible!)

So D distinguishes with non-negligible advantage, contradicting PRG security. □

> **Key insight:** The PRG "compresses" the seed, so most strings in $\{0,1\}^{2n}$ are NOT in the image of G. This asymmetry allows distinguishing and proves one-wayness.

Part (c): PRG → PRF (The GGM Construction)

Goal: Construct a PRF from a PRG.

16.3.1. The GGM Tree Construction

Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a length-doubling PRG.

Write $G(s) = G_0(s) \parallel G_1(s)$ where $G_0, G_1 : \{0,1\}^n \to \{0,1\}^n$ are the left and right halves.

PRF $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$:

For key $k \in \{0,1\}^n$ and input $x = x_1 x_2 ... x_n \in \{0,1\}^n$:

$$F_k(x) = G_{x_n}\Big(G_{x_{n-1}}\Big(...G_{x_2}\Big(G_{x_1}(k)\Big)...\Big)\Big)$$

That is, start with k, and for each bit x_i of the input:
- If $x_i = 0$: take the left half of G
- If $x_i = 1$: take the right half of G

16.3.2. Pictorial Representation

Binary tree of depth n:

```
Level 0 (root):     k
                   / \
Level 1:       G₀(k)  G₁(k)
               / \     / \
Level 2:   G₀G₀(k) G₁G₀(k) G₀G₁(k) G₁G₁(k)
              ...
Level n:   F_k(0...00)  F_k(0...01)  ...  F_k(1...11)
```

Each leaf corresponds to one input x, and $F_k(x)$ is the value at that leaf.

16.3.3. Why This is a PRF

Proof by hybrid argument (sketch):

Define hybrids $H_0, H_1, ..., H_n$ where:

- H_0: Real PRF F_k
- H_i: First i levels of the tree are replaced with truly random values
- H_n: Completely random function

Adjacent hybrids are indistinguishable:

H_i and H_{i+1} differ only in whether level $i+1$ values are computed via G or are random.

At level i, the values are either:

- PRG outputs (derived from level $i-1$)
- Already random

Replacing $G(s)$ with random values is indistinguishable by PRG security (for each node at level i).

By a union bound over polynomially many nodes queried, adjacent hybrids are indistinguishable.

> **Conclusion:** $H_0 \underset{c}{\approx} H_n$, so F_k is indistinguishable from a random function.
>
> Therefore, F is a PRF. □

Summary Table

From (X)	To (Y)	Construction
OWP	PRG	$G(s) = f(s) \parallel b(s)$ (hard-core bit)
PRG	OWF	$f(s) = G(s)$ (direct use)
PRG	PRF	GGM tree: $F_k(x) = G_{x_n}\left(...G_{x_1}(k)\right)$

Relationships:

$$\text{OWP} \to \text{PRG} \to \text{PRF} \to \text{OWF}$$

And also: PRG $\to$ OWF directly.

This shows that OWPs are a strong assumption that implies all other primitives!

◆ ◆ ◆

Problem 17: EAV-Security vs CPA-Security

Setup: Let F be a length-preserving pseudorandom function and G be a pseudorandom generator with expansion factor $\ell(n) = n + 1$. For each encryption scheme below, state whether it is EAV-secure and whether it is CPA-secure. The shared key $k \in \{0,1\}^n$ is uniform.

1. To encrypt $m \in \{0,1\}^{n+1}$, choose uniform $r \in \{0,1\}^n$ and output ciphertext $\langle r, G(r) \oplus m \rangle$.
2. To encrypt $m \in \{0,1\}^n$, output ciphertext $m \oplus F_k(0^n)$.
3. To encrypt $m \in \{0,1\}^{2n}$, parse m as $m_1 \parallel m_2$ with $|m_1| = |m_2|$, then choose uniform $r \in \{0,1\}^n$ and send $\langle r, m_1 \oplus F_k(r), m_2 \oplus F_k(r+1) \rangle$.

Background: Security Notions

EAV-Security (Eavesdropper): Adversary sees ONE ciphertext of a chosen message. Cannot distinguish which of two messages was encrypted.

CPA-Security (Chosen Plaintext): Adversary has ACCESS to encryption oracle. Can request encryptions of arbitrary messages, then tries to distinguish challenge ciphertext.

Scheme 1: $\text{Enc}(m) = \langle r, G(r) \oplus m \rangle$ where $r \leftarrow \{0,1\}^n$

Note: This scheme does NOT use the key k at all!

17.2.1. EAV-Security Analysis

EAV-Secure: YES

Proof:

For a single encryption:
- r is uniformly random
- $G(r)$ is pseudorandom (indistinguishable from uniform U_{n+1})
- $G(r) \oplus m$ is therefore pseudorandom (XOR with fixed message preserves pseudorandomness)

An EAV adversary seeing $(r, G(r) \oplus m)$ cannot distinguish m_0 from m_1 because $G(r) \oplus m_0$ and $G(r) \oplus m_1$ are both pseudorandom.

17.2.2. CPA-Security Analysis

CPA-Secure: NO

Attack:

1. Request encryption of $m' = 0^{n+1}$
2. Receive $(r', G(r') \oplus 0^{n+1}) = (r', G(r'))$

3. Now the adversary knows $G(r')$ for some r'
4. Submit challenge messages m_0, m_1
5. If challenge uses same $r = r'$, can compute $G(r) \oplus m_b$ and determine b

Better attack: Since the scheme doesn't use key k, the adversary can compute $G(r)$ themselves for any r!

Given challenge (r^*, c^*), compute $G(r^*)$ and check if $c^* \oplus G(r^*) = m_0$ or m_1.

Success probability: 1 (deterministic attack)

Scheme 2: $\text{Enc}(m) = m \oplus F_k(0^n)$

Note: This is deterministic encryption — same message always produces same ciphertext.

17.3.1. EAV-Security Analysis

EAV-Secure: YES

Proof:

For a single encryption, $F_k(0^n)$ is a fixed pseudorandom value (depending only on k).

The ciphertext $m \oplus F_k(0^n)$ is pseudorandom because:
- $F_k(0^n)$ is indistinguishable from uniform (by PRF security)
- XOR with a fixed message preserves this property

An EAV adversary cannot distinguish which message was encrypted.

17.3.2. CPA-Security Analysis

CPA-Secure: NO

Attack:

Deterministic encryption is NEVER CPA-secure!

1. Request encryption of m_0: receive $c_0 = m_0 \oplus F_k(0^n)$
2. Submit challenge messages m_0, m_1
3. Receive challenge ciphertext $c^* = m_b \oplus F_k(0^n)$
4. If $c^* = c_0$, output $b = 0$; else output $b = 1$

Success probability: 1

Scheme 3: $\text{Enc}(m_1 \parallel m_2) = \langle r, m_1 \oplus F_k(r), m_2 \oplus F_k(r+1) \rangle$

Note: This is essentially CTR mode with a random starting counter.

17.4.1. EAV-Security Analysis

EAV-Secure: YES

Proof:

For a single encryption with random r:
- $F_k(r)$ and $F_k(r+1)$ are pseudorandom (by PRF security)

- They are also independent (different inputs to PRF)
- $m_1 \oplus F_k(r)$ and $m_2 \oplus F_k(r+1)$ are both pseudorandom

The ciphertext reveals nothing about m_1, m_2 to an EAV adversary.

17.4.2. CPA-Security Analysis

CPA-Secure: YES

Proof:

This is randomized encryption using a PRF in counter mode.

Why CPA-secure:

1. Each encryption uses fresh random r
2. With overwhelming probability, all counter values $(r, r+1)$ across all encryptions are distinct (assuming counter space is large enough)
3. PRF outputs on distinct inputs are indistinguishable from independent random values
4. Therefore, each encryption is essentially a one-time pad with fresh randomness

Formally, by a hybrid argument:
- Replace F_k with truly random function R
- Condition on no counter collisions (negligible probability)
- In this case, all pad values are independent and uniform

Summary

Scheme	EAV-Secure?	CPA-Secure?
1: $\langle r, G(r) \oplus m \rangle$	YES	NO (key-less)
2: $m \oplus F_k(0^n)$	YES	NO (deterministic)
3: $\langle r, m_1 \oplus F_k(r), m_2 \oplus F_k(r+1) \rangle$	YES	YES (CTR mode)

Key Insights:

- **EAV $\not\Rightarrow$ CPA:** Schemes 1 and 2 show that EAV security does NOT imply CPA security
- **Deterministic encryption** is never CPA-secure (Scheme 2)
- **Key-less encryption** leaks the pad (Scheme 1)
- **Randomized PRF-based schemes** (like CTR mode in Scheme 3) can achieve CPA security

◆ ◆ ◆

Problem 18: P ≠ NP vs One-Way Functions

Task: Show that the existence of one-way functions implies $P \neq \text{NP}$.

Conversely, assume $P \neq \text{NP}$. Show that there exists a function f that is:
- Computable in polynomial time
- Hard to invert in the **worst case** (i.e., for all PPT algorithms A, $\Pr_{x \leftarrow \{0,1\}^n}[f(A(f(x))) = f(x)] \neq 1$)

but f is **not** a one-way function.

Part 1: One-Way Functions Imply P ≠ NP

Claim: If one-way functions exist, then $P \neq \text{NP}$.

Proof by contrapositive:

Assume $P = \text{NP}$. We show that no one-way function can exist.

Let $f : \{0,1\}^n \to \{0,1\}^*$ be any polynomial-time computable function.

Define the language:

$$L_f = \{(y, i, b) : \exists x \text{ such that } f(x) = y \text{ and the } i\text{-th bit of } x \text{ is } b\}$$

Key observation: $L_f \in \text{NP}$

Given (y, i, b), the witness is x such that $f(x) = y$ and $x_i = b$.

Verification: Check that $f(x) = y$ (polynomial time) and $x_i = b$.

If $P = \text{NP}$, then $L_f \in P$. This means we can decide in polynomial time whether there exists a preimage with a specific bit value.

Inverting f using the P = NP assumption:

Inverter $I(y)$:

For $i = 1$ to n:
1. Query: Is $(y, i, 0) \in L_f$?
2. If yes, set $x_i = 0$
3. If no, set $x_i = 1$ (assuming a preimage exists)

Output $x = x_1 x_2 ... x_n$

This runs in polynomial time and correctly inverts f whenever y has a preimage.

Conclusion: If $P = \text{NP}$, every polynomial-time function can be inverted in polynomial time, so no OWF exists.

By contrapositive: **OWF exists ⇒ P ≠ NP**. □

Part 2: P $\neq$ NP Does NOT Imply OWF

We now show the converse is **not** true: $P \neq$ NP does not necessarily imply OWFs exist.

18.2.1. Understanding the Distinction

One-Way Function (OWF): Requires **average-case** hardness.

For all PPT A: $\Pr_{x \leftarrow \{0,1\}^n}[f(A(f(x))) = f(x)] \leq \text{negl}(n)$

P $\neq$ NP: Only guarantees **worst-case** hardness.

There exists **some** input on which inversion fails.

18.2.2. Construction of Worst-Case Hard but NOT One-Way Function

Assume $P \neq$ NP. Then there exists an NP-complete language L (e.g., SAT).

Construction:

Define $f : \{0,1\}^* \to \{0,1\}^*$ as follows:

For input $x = (\varphi, w)$ where φ is a Boolean formula and w is a potential witness:

$$f(\varphi, w) = \begin{cases} (\varphi, 1) \text{ if } w \text{ is a satisfying assignment for } \varphi \\ (\varphi, 0) \text{ otherwise} \end{cases}$$

18.2.3. Analysis

f is polynomial-time computable:

Given φ and w, we can check in polynomial time whether w satisfies φ (just evaluate the formula).

f is worst-case hard to invert:

> Consider the output $(\varphi, 1)$ for a satisfiable formula φ.
>
> To invert, we must find a satisfying assignment w for φ.
>
> Since $P \neq$ NP, there is no polynomial-time algorithm that can find satisfying assignments for **all** satisfiable formulas.
>
> Therefore, f is **worst-case** hard to invert.

f is NOT one-way (fails average-case):

Attack on average-case:

For a random input (φ, w):

- Compute $f(\varphi, w) = (\varphi, b)$
- If $b = 0$: Simply output $(\varphi, 0^n)$ — this is a valid preimage!
- If $b = 1$: Output (φ, w) — we already have the witness!

Wait, but we don't know w when inverting...

Correct analysis: Given $y = (\varphi, b)$, we need to find (φ, w) such that $f(\varphi, w) = y$.

- If $b = 0$: Any w that is NOT a satisfying assignment works. Almost all w are non-satisfying (for random φ), so just output $(\varphi, 0^n)$.

- If $b = 1$: We need to find a satisfying assignment. This is hard for SOME formulas, but random satisfiable formulas are EASY!

Why random instances are easy:

For a **random** Boolean formula φ (or random instance of an NP problem):

- Most formulas are either trivially unsatisfiable or have many satisfying assignments (easy to find one)
- The "hard" instances form a negligible fraction of all instances
- Algorithms like DPLL, WalkSAT, etc., solve random instances efficiently with high probability

Therefore: $\Pr_{x \leftarrow \{0,1\}^n}[A \text{ inverts } f(x)] = 1 - \text{negl}(n)$ for an appropriate A

But this is exactly what OWF security forbids!

Summary

Statement	**Status**
OWF exists $\Rightarrow$ P $\neq$ NP	TRUE — proven above
P $\neq$ NP $\Rightarrow$ OWF exists	UNKNOWN — widely believed but unproven

Key Insight:

- **P $\neq$ NP** is about **worst-case** complexity: some instances are hard
- **OWF** requires **average-case** hardness: random instances must be hard

The gap between worst-case and average-case hardness is fundamental. NP-complete problems can have hard worst-case instances but easy random instances (like SAT).

Whether P $\neq$ NP implies OWF existence remains one of the deepest open questions in cryptography and complexity theory!

The constructed f:

- Poly-time computable ✓
- Worst-case hard to invert ✓ (follows from P $\neq$ NP)
- NOT one-way ✓ (random instances are easy)

This demonstrates that P $\neq$ NP alone does not give us cryptographic security!

Appendix: Background Concepts & Prerequisites

A. Classical Ciphers

A.1 Shift Cipher (Caesar Cipher)

The **shift cipher** encrypts by shifting each letter by a fixed amount.

- **Key:** $k \in \{0, 1, 2, ..., 25\}$
- **Encryption:** $c_i = (m_i + k) \bmod 26$
- **Decryption:** $m_i = (c_i - k) \bmod 26$

Example: With $k = 3$: A→D, B→E, ..., Z→C

Security: Only 26 possible keys — trivially broken by brute force.

A.2 Substitution Cipher

The **substitution cipher** replaces each letter with another according to a fixed permutation.

- **Key:** A permutation $\pi : \{A, ..., Z\} \to \{A, ..., Z\}$
- **Encryption:** $c_i = \pi(m_i)$
- **Decryption:** $m_i = \pi^{-1}(c_i)$

Key space: $26! \approx 4 \times 10^{\{26\}}$

Security: Despite large key space, vulnerable to **frequency analysis** because each letter always maps to the same ciphertext letter.

A.3 Vigenère Cipher

The **Vigenère cipher** uses a repeating keyword to apply different shifts at different positions.

- **Key:** A keyword of length t, represented as $k = (k_0, k_1, ..., k_{t-1})$
- **Encryption:** $c_i = (m_i + k_{i \bmod t}) \bmod 26$
- **Decryption:** $m_i = (c_i - k_{i \bmod t}) \bmod 26$

Example: Keyword "KEY" ($t = 3$) with values $(10, 4, 24)$:
- Position 0: shift by 10
- Position 1: shift by 4
- Position 2: shift by 24
- Position 3: shift by 10 (repeats)

Security: Stronger than simple substitution but breakable via **Kasiski examination** (finding period) followed by frequency analysis on each position.

B. Number Theory Foundations

B.1 Discrete Logarithm Problem (DLP)

Let p be a prime and g a generator of the multiplicative group $\mathbb{Z}_p^* = \{1, 2, ..., p-1\}$.

Problem: Given $y = g^x \bmod p$, find x.

Properties:
- **Easy direction:** Computing $g^x \bmod p$ is efficient (square-and-multiply)
- **Hard direction:** Finding x from y is believed computationally infeasible for large p

Why it matters: DLP hardness is the foundation of Diffie-Hellman key exchange and many cryptographic constructions.

B.2 Group Structure of $\mathbb{Z}_p^*$

For a prime p:
- $\mathbb{Z}_p^* = \{1, 2, ..., p-1\}$ under multiplication mod p
- Order of the group: $|\mathbb{Z}_p^*| = p - 1$
- A **generator** g satisfies: $\{g^0, g^1, g^2, ..., g^{p-2}\} = \mathbb{Z}_p^*$

When $p - 1 = s \cdot 2^r$ (with s odd):
- The group has a subgroup of order 2^r (easy DLP via Pohlig-Hellman)
- The "hard" part of DLP lives in the subgroup of odd order s

B.3 Pohlig-Hellman Algorithm

An algorithm that efficiently solves DLP in groups whose order has only **small prime factors**.

Key insight: If $|G| = p_1^{e_1} \cdot p_2^{e_2} \cdot ... \cdot p_k^{e_k}$, then:
1. Solve DLP in each subgroup of order $p_i^{e_i}$ separately
2. Combine results using the Chinese Remainder Theorem

Implication: For safe cryptographic use, $p - 1$ should have a large prime factor.

C. Cryptographic Concepts

C.1 One-Way Functions

A function $f : X \to Y$ is **one-way** if:
- $f(x)$ is efficiently computable for all $x \in X$
- For a random x, given $f(x)$, no efficient algorithm can find x' with $f(x') = f(x)$ with non-negligible probability

Examples:
- $f(x) = g^x \bmod p$ (assuming DLP is hard)
- Cryptographic hash functions (e.g., SHA-256)

C.2 Hard-Core Predicates

A predicate $B : X \to \{0,1\}$ is **hard-core** for one-way function f if:
- $B(x)$ is efficiently computable given x
- Given only $f(x)$, predicting $B(x)$ is no better than random guessing

Formal definition: For all PPT (probabilistic polynomial-time) adversaries A:

$$\Pr[A(f(x)) = B(x)] \leq \frac{1}{2} + \text{negl}(n)$$

Goldreich-Levin Theorem: For any OWF f, there exists a hard-core predicate (the inner product with a random vector).

C.3 Pseudo-Random Generators (PRGs)

A **PRG** is a deterministic function $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ where $\ell(n) > n$ such that:
- **Expansion:** Output is longer than input
- **Pseudorandomness:** No efficient algorithm can distinguish $G(s)$ from truly random $r \in \{0,1\}^{\ell(n)}$

Construction from OWF + Hard-Core Bit (Blum-Micali):

$$x_{i+1} = f(x_i), \quad b_i = B(x_i)$$

Output: $b_1 \parallel b_2 \parallel ... \parallel b_n$

C.4 Computational vs Information-Theoretic Security

Information-Theoretic (Perfect)	**Computational**
Secure against unbounded adversaries	Secure against efficient (PPT) adversaries
Cannot be broken even with infinite time	Could theoretically be broken with enough time
Example: One-Time Pad	Example: AES, RSA
Requires key ≥ message length	Short keys can protect long messages

C.5 Negligible Functions

A function $f : \mathbb{N} \to \mathbb{R}$ is **negligible** if it decreases faster than the inverse of any polynomial.

Formal Definition: For every positive polynomial $p(n)$, there exists $N \in \mathbb{N}$ such that for all $n > N$:

$$|f(n)| < \frac{1}{p(n)}$$

Equivalent Characterization: f is negligible $\Leftrightarrow$ for all $c > 0$: $\lim_{n\to\infty} n^c \cdot f(n) = 0$

Key Test for $f(n) = 2^{-g(n)}$:
- Negligible if $g(n) = \omega(\log n)$, i.e., $\frac{g(n)}{\log n} \to \infty$
- NOT negligible if $g(n) = O(\log n)$

Examples:

- 2^{-n}, $2^{-\sqrt{n}}$, $2^{-(\log n)^2}$: negligible
- n^{-100}, $2^{-\sqrt{\log n}}$: NOT negligible

Why it matters: In cryptography, security proofs show that adversary's advantage is negligible in the security parameter n.

D. Proof Techniques

D.1 Security Reduction

A **reduction** proves that breaking scheme S implies breaking a hard problem P.

Structure:
1. Assume an adversary A breaks S with advantage ε
2. Construct algorithm B that uses A as a subroutine
3. Show B solves P with related advantage

Contrapositive: If P is hard, then S is secure.

D.2 Hybrid Argument

A technique for proving two distributions are indistinguishable.

Method:
1. Define a sequence of hybrid distributions: $H_0, H_1, ..., H_n$
2. H_0 = first distribution, H_n = second distribution
3. Prove adjacent hybrids H_i and H_{i+1} are indistinguishable
4. By transitivity: $H_0 \underset{c}{\approx} H_n$

Why it works: If H_0 and H_n were distinguishable, some adjacent pair must also be distinguishable (pigeon-hole).

E. Kerckhoffs's Principle

Stated by Auguste Kerckhoffs in 1883:

> **"A cryptosystem should be secure even if everything about the system, except the key, is public knowledge."**

Modern interpretation (Shannon's Maxim): "The enemy knows the system."

Implications:
- Security must rely solely on key secrecy
- Algorithms should be publicly scrutinized
- Never rely on "security through obscurity"

— End of Appendix —