CS 587 – Final Assignment

Sri Teja Kale

May 2025

Exercise 1. Katz–Lindell Exercise 1.7 [20 points]

Question: The index of coincidence method relies on a known value for the sum of the squares of plaintext-letter frequencies. Why would it not work using the sum $\sum_i p_i$ itself?

Answer: The index of coincidence (IC) relies on the statistic

$$\mathrm{IC} \;=\; \sum_i \frac{n_i(n_i-1)}{N(N-1)} \;\approx\; \sum_i p_i^2, \qquad p_i \;=\; \frac{n_i}{N},$$

where:

- N is the total number of letters in the text,
- n_i is the count of the i-th letter,
- p_i is the empirical frequency of that letter.

Squaring each p_i magnifies the contribution of common letters and diminishes that of rare ones.

Why $\sum_i p_i$ cannot replace $\sum_i p_i^2$:

1. **Normalization Property:** For any probability distribution, $\sum_i p_i = 1$. This constant provides no discriminatory power between natural and random texts.
2. **Discriminative Power:** The squared sum varies by distribution:

   $$\mathrm{IC}_{\text{English}} \approx 0.065, \quad \mathrm{IC}_{\text{uniform}} = \frac{1}{26} \approx 0.038.$$

   This measurable difference enables detection of structured language.
3. **Use in Vigenère Attacks:** When ciphertext is split using the correct key length, the columns reflect natural-language IC (≈ 0.065); incorrect splits remain closer to uniform IC (≈ 0.038).

Conclusion: Because $\sum_i p_i$ is always 1, it provides no discriminatory information. Squared probabilities, $\sum_i p_i^2$, reflect how uneven the frequency distribution is—making IC an effective test for language structure and polyalphabetic cipher analysis.

Exercise 2. Katz–Lindell Exercise 3.6 [40 points]

Let G be a pseudorandom generator. For each case, we determine whether G' is necessarily a pseudorandom generator.

1. Define $G'(s) = G(\overline{s})$, where $\overline{s}$ is the complement of s

G' is a pseudorandom generator.
Proof:

- When s is chosen uniformly at random, $\overline{s}$ (the bitwise complement of s) is also uniformly distributed over $\{0,1\}^n$.
- For any distinguisher D trying to distinguish $G'(s)$ from random:

$$\Pr[D(G'(s)) = 1] = \Pr[D(G(\overline{s})) = 1]$$

- Since $\overline{s}$ is uniformly random when s is, $G(\overline{s})$ is computationally indistinguishable from random (by the security of G).
- Therefore, G' maintains the pseudorandomness property of G.

2. Define $G'(s) = \overline{G(s)}$

G' is a pseudorandom generator.
Proof:

- Here, the overbar on $G(s)$ indicates the complement of $G(s)$, i.e., all bits flipped.
- When s is chosen uniformly at random, $G(s)$ is computationally indistinguishable from a random string.
- The complement of a random string is also random. If r is a uniformly random string, then $\overline{r}$ is also uniformly random.
- Thus, for any distinguisher D:

$$|\Pr[D(G'(s)) = 1] - \Pr[D(r) = 1]| = \left|\Pr[D(\overline{G(s)}) = 1] - \Pr[D(r) = 1]\right| \leq \text{negl}(n)$$

- Since G is a PRG, G' is also a PRG.

3. Define $G'(s) = G(0^{|s|}||s)$

G' is not necessarily a pseudorandom generator.
Counterexample:

- The seed $0^{|s|}||s$ is not uniformly distributed when s is uniformly chosen.

- We can construct a specific G such that:
- If the first $|s|$ bits are all zeros, it outputs a fixed pattern (e.g., all zeros)
- Otherwise, it outputs a pseudorandom string
- Such a G would be a valid PRG for uniformly random inputs.

- But $G'(s) = G(0^{|s|}||s)$ would always output the fixed pattern, making it trivially distinguishable from random.

4. Define $G'(s) = G(s)\|G(s+1)$

G' is a pseudorandom generator.
Proof:

- Based on lecture content (slide 40): when G is a PRG, concatenating outputs of G with related but different inputs maintains pseudorandomness.
- **By the hybrid argument:**
- No efficient distinguisher can tell $G(s)$ from random
- No efficient distinguisher can tell $G(s+1)$ from random

- Therefore, no efficient distinguisher can tell $G(s)\|G(s+1)$ from random $\|random$
  G' also satisfies the expansion requirement: if G expands n bits to $\ell(n) > n$ bits, then G' expands n bits to $2 \cdot \ell(n) > 2n$ bits.
  - This aligns with pseudorandomness principles using the hybrid argument technique.

Exercise 3. Katz–Lindell Exercise 11.4 [40 points]

1. Alice chooses uniform $k, r \in \{0,1\}^n$, and sends $s := k \oplus r$ to Bob.
2. Bob chooses uniform $t \in \{0,1\}^n$, and sends $u := s \oplus t$ to Alice.
3. Alice computes $w := u \oplus r$ and sends w to Bob.
4. Alice outputs k , and Bob outputs $w \oplus t$.

Protocol Steps

The protocol from Katz-Lindell Exercise 11.4 works as follows:

1. Alice chooses uniform random values $k, r \in \{0,1\}^n$, and sends $s := k \oplus r$ to Bob.
2. Bob chooses uniform random value $t \in \{0,1\}^n$, and sends $u := s \oplus t$ to Alice.
3. Alice computes $w := u \oplus r$ and sends w to Bob.
4. Alice outputs k as her key, and Bob outputs $w \oplus t$ as his key.

Part 1: Proving Alice and Bob output the same key

To demonstrate that Alice and Bob establish the same key, we need to show that Alice's output k equals Bob's output $w \oplus t$.

Step-by-step derivation:
Starting with Bob's output: $w \oplus t$
Substituting $w = u \oplus r$ from step 3:
$w \oplus t = (u \oplus r) \oplus t = u \oplus r \oplus t$
Substituting $u = s \oplus t$ from step 2:
$u \oplus r \oplus t = (s \oplus t) \oplus r \oplus t = s \oplus t \oplus r \oplus t$
Using the XOR property that $a \oplus a = 0$ for any bit string a:
$s \oplus t \oplus r \oplus t = s \oplus r \oplus (t \oplus t) = s \oplus r \oplus 0 = s \oplus r$
Finally, substituting $s = k \oplus r$ from step 1:
$s \oplus r = (k \oplus r) \oplus r = k \oplus (r \oplus r) = k \oplus 0 = k$

Therefore, Bob's output $w \oplus t = k$, which is exactly what Alice outputs. This confirms they establish the same key.

Part 2: Security Analysis

Now let's analyze whether this protocol is secure against an eavesdropper who observes all the transmitted messages.

Messages visible to an eavesdropper:

- Message 1: $s = k \oplus r$ (sent from Alice to Bob)
- Message 2: $u = s \oplus t$ (sent from Bob to Alice)
- Message 3: $w = u \oplus r$ (sent from Alice to Bob)

Attack Method 1: Algebraic combination of messages

An attacker can recover the key directly as follows:

1. Compute $s \oplus w$:
   $s \oplus w = (k \oplus r) \oplus (u \oplus r) = k \oplus r \oplus u \oplus r = k \oplus u$ (since $r \oplus r = 0$)
2. Then compute $(s \oplus w) \oplus u$:
   $(k \oplus u) \oplus u = k \oplus (u \oplus u) = k$ (since $u \oplus u = 0$)

The attacker has now recovered the key k by simply performing XOR operations on the observed messages.

Attack Method 2: Recovering the random values first

Alternatively, the attacker can:

1. Compute r from messages u and w: Since $w = u \oplus r$, we can rearrange to get $r = w \oplus u$
2. With r known, compute k from message s: Since $s = k \oplus r$, we can rearrange to get $k = s \oplus r$
3. Substituting the value of r: $k = s \oplus (w \oplus u) = s \oplus w \oplus u$

Either way, the attacker can recover k using only XOR operations.

Why This Protocol Fails

The protocol is insecure for several fundamental reasons:

1. **Linear operations only:** The protocol uses exclusively XOR operations, which are linear. This means that linear combinations of the messages can eliminate the random values (r and t) and expose the key.
2. **No computational hardness assumption:** Secure key exchange protocols like Diffie-Hellman rely on problems that are believed to be computationally difficult (e.g., the discrete logarithm problem). This protocol doesn't incorporate any such hardness.
3. **Insufficient hiding of secrets:** The random values r and t are intended to hide the key k, but the linear structure of the protocol allows an attacker to systematically eliminate these random values.
4. **Deterministic relationship between messages:** All messages have predictable algebraic relationships that an attacker can exploit.

Comparison with Diffie-Hellman

In contrast, the Diffie-Hellman protocol:

- Uses exponentiation in a cyclic group, which is non-linear
- Relies on the computational difficulty of the Discrete Logarithm problem
- Ensures that even knowing public values g^x and g^y, an attacker cannot feasibly compute the shared key g^{xy}

Conclusion

While the protocol successfully allows Alice and Bob to establish the same key (correctness property), it completely fails to provide security against eavesdroppers. An attacker can efficiently recover the key through simple algebraic manipulations of the observed messages.

This analysis demonstrates why cryptographic protocols must incorporate computational hardness assumptions and non-linear operations to achieve security.

Exercise 4. Katz–Lindell Exercise 12.15 [20 points]

Consider the RSA-based encryption scheme in which a user encrypts a message $m \in \{0,1\}^\ell$ with respect to the public key $\langle N, e\rangle$ by computing $\hat{m} := H(m) \parallel m$ and outputting the ciphertext $\hat{m}^e \bmod N$. (Here, let $H : \{0,1\}^\ell \to \{0,1\}^n$ and assume $\ell + n < \|N\|$.) Is this scheme CPA-secure if H is modeled as a random oracle?

Analysis of the Given RSA-based Encryption Scheme

Let's break down the encryption scheme first:

- A user encrypts a message $m \in \{0,1\}^\ell$ using public key $\langle N, e\rangle$
- Encryption computes $\hat{m} := H(m) \parallel m$ (concatenation of hash and message)
- Outputs ciphertext $\hat{m}^e \bmod N$
- $H : \{0,1\}^\ell \to \{0,1\}^n$ is modeled as a random oracle
- It's given that $\ell + n < \|N\|$ (the bit length of $\hat{m}$ is less than the bit length of N)

Determining CPA Security

To determine if this scheme is CPA-secure, I need to analyze whether an adversary with access to an encryption oracle can distinguish between the encryptions of two chosen messages.

Key Vulnerability: Deterministic Encryption

This scheme is **not CPA-secure**, even with H modeled as a random oracle. Here's why:

1. **Deterministic Nature**: For any fixed message m, the encryption will always produce the same ciphertext. This is because:
   - $H(m)$ will be consistent (even as a random oracle, it gives the same output for the same input)
   - The concatenation $H(m) \parallel m$ will be the same each time
   - The RSA operation $\hat{m}^e \bmod N$ is deterministic
2. **CPA Attack**: In a chosen-plaintext attack scenario, an adversary can:
   - Choose two messages m_0 and m_1
   - Receive a challenge ciphertext $c = (H(m_b) \parallel m_b)^e \bmod N$ for $b \in \{0,1\}$
   - Compute $c_0 = (H(m_0) \parallel m_0)^e \bmod N$ using the encryption oracle
   - If $c = c_0$, then $b = 0$; otherwise, $b = 1$
   - This gives the adversary a 100% success probability in the CPA game
3. **Lack of Randomization**: The scheme doesn't incorporate any fresh randomness in each encryption, which is a fundamental requirement for CPA security.

Additional Considerations

1. **Random Oracle Model**: Even though H is modeled as a random oracle, this doesn't introduce the necessary per-encryption randomness. A random oracle ensures unpredictability of the output for new inputs, but here the same input (message) will always yield the same output.

2. **RSA-OAEP Comparison**: This scheme resembles a simplified version of RSA-OAEP, but lacks critical components like:

   - The incorporation of random padding in each encryption
   - Proper domain separation and padding structure

Conclusion

This RSA-based encryption scheme is **not CPA-secure** when H is modeled as a random oracle. The fundamental issue is that it's deterministic, allowing an adversary to easily win the CPA security game by comparing ciphertexts.

For CPA security, the scheme would need to incorporate fresh randomness in each encryption operation, like RSA-OAEP does with its random padding element.