COL759 Cryptography Semester 1, 2023-24

Problem Set 1

Instructor: Venkata K *Due Date: 28 August 2023*

Instructions:

- Assignment must be done in groups of size at most 2. Each group must submit one pdf on Gradescope, and mention the partner's name (if any).
- The questions are divided into two parts. The first section (Part A.) consists of four theoretical questions (32 marks). In the second part, you can either attempt the coding questions (in Part B.1.) or the theoretical question (in Part B.2.). In case both sections are attempted, we will consider the higher score.
- All solutions must be typeset in LaTeX. For the coding questions, provide a brief explanation of your approach and upload the relevant files on Gradescope.
- Students who are interested in a BTP/MTP in theoretical cryptography are strongly encouraged to attempt the theoretical question (Part B.2.).
- (Optional) Discuss how much time was spent on each problem. This will not be used for evaluation. We will use this for calibrating future assignments.

Notations:

- $\{0,1\}^{\leq \ell}$ denotes the set of all strings of length at most ℓ. For any string $x \in \{0,1\}^{\ell}$ and $i \in \{1,\ldots,\ell\}$, $x[i]$ denotes the i^{th} bit of x.
- $x \mid\mid y$ denotes the concatenation of x and y.

Part A. (32 marks)

1. **Perfect Two-time Security (7 marks)**

   In class, we saw that Shannon's one-time pad satisfies perfect one-time security. We also briefly discussed in class that no encryption scheme with deterministic encryption can satisfy perfect two-time security (even with key space much larger than the message space). In this problem, we will see how to use randomness to achieve almost-perfect two-time security.

   First, let us define ϵ-perfect two-time security. A symmetric-key encryption scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$, key space $\mathcal{K}$ and ciphertext space $\mathcal{C}$ satisfies ϵ-perfect two-time security if any adversary $\mathcal{A}$ has winning probability at most $1/2 + \epsilon$ in the following security game:

   - The adversary $\mathcal{A}$ sends two pairs of messages $(m_{0,0}, m_{0,1})$ and $(m_{1,0}, m_{1,1}) \in \mathcal{M}^2$.
   - The challenger chooses a uniformly random key $k \leftarrow \mathcal{K}$ and a bit $b \leftarrow \{0,1\}$. It computes $\mathsf{ct}_0 \leftarrow \mathsf{Enc}(k, m_{b,0})$ and $\mathsf{ct}_1 \leftarrow \mathsf{Enc}(k, m_{b,1})$ and sends $(\mathsf{ct}_0, \mathsf{ct}_1)$.
   - The adversary sends a guess b' and wins if $b = b'$.

   We say that the scheme satisfies perfect two-time security if any adversary $\mathcal{A}$ has winning probability equal to $1/2$.

   (a) Prove that no symmetric-key encryption scheme can satisfy the perfect two-time security definition (even if the encryption algorithm is allowed to be randomized). You must describe your adversary $\mathcal{A}$ in detail, and compute its winning probability in the above security game. You can assume the message space is $\{0,1\}^{\ell_1}$, key space is $\{0,1\}^{\ell_2}$ and the randomness used for encryption is drawn from $\{0,1\}^{\ell_3}$.

   Interestingly, we can construct encryption schemes that are ϵ-perfect two-time secure, **for any** $\epsilon > 0$! For simplicity, we will fix $\mathcal{M} = \{0,1\}^{\ell}$, and we would like to achieve $O(2^{-\ell})$-perfect two-time security. In order to construct such an encryption scheme, we will use *pairwise independent hash functions.*

   Definition 1 (Pairwise Independent Hash Function) *A pairwise independent hash function family $\mathcal{H}$ is a set of keyed functions $\{h_k : \mathcal{X} \to \mathcal{Y}\}_{k \in \mathcal{K}}$ that satisfies the following property: for any **distinct** $x_0, x_1 \in \mathcal{X}$ and any $y_0, y_1 \in \mathcal{Y}$,*

   $$\mathsf{Pr}_k \left[h_k(x_0) = y_0 \wedge h_k(x_1) = y_1 \right] = \frac{1}{|\mathcal{Y}|^2}$$

   We will see how to construct such hash functions later in the course.

   (b) Let $\mathcal{H}$ be a pairwise independent hash function family with key space $\mathcal{K}$, input space $\{0,1\}^{\ell}$ and output space $\{0,1\}^{\ell}$. Use $\mathcal{H}$ to construct an encryption scheme

with message space $\{0,1\}^{\ell}$ such that the scheme satisfies $O(2^{-\ell})$-perfect two-time security. (Note: your encryption scheme must be *stateless*.) You must provide a detailed proof of security, including appropriate hybrid experiments.

2. **Secure/Insecure PRGs and PRFs (8 marks)**

(a) Let $\mathcal{G} = \left\{G_n : \{0,1\}^n \to \{0,1\}^{3n}\right\}_{n\in\mathbb{N}}$ be a secure pseudorandom generator family. Consider the following function families derived from $\mathcal{G}$. For each of them, either prove that the function family is a secure PRG, or disprove by showing a polynomial time attacker. For proving security, it suffices to provide an informal argument (no need to give a reduction).

i. $\mathcal{G}' = \left\{G'_n : \{0,1\}^{2n} \to \{0,1\}^{3n}\right\}_{n\in\mathbb{N}}$, where

$$G'_n(s_1 \mid\mid s_2) = G_n(s_1) \wedge G_n(s_2).$$

Here $\wedge$ denotes bitwise AND operation.

ii. $\mathcal{G}' = \left\{G'_n : \{0,1\}^{2n} \to \{0,1\}^{3n}\right\}_{n\in\mathbb{N}}$, where

$$G'_n(s_1 \mid\mid s_2) = G_n(s_1) \oplus G_n(s_2).$$

Here $\wedge$ denotes bitwise XOR operation.

(b) Let $\mathcal{F} = \{F_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n\}_{n\in\mathbb{N}}$ be a family of secure pseudorandom functions. Consider the following keyed function families derived from $\mathcal{F}$. For each of them, either prove that the function family is a secure PRF, or disprove by showing a polynomial time attacker. For proving security, it suffices to provide an informal argument (no need to give a reduction).

i. $\mathcal{F}' = \left\{F'_n : \{0,1\}^n \times \{0,1\}^{2n} \to \{0,1\}^n\right\}_{n\in\mathbb{N}}$ where

$$F'_n(k, (x_1, x_2)) = F_n(k, x_1) \oplus F_n(k, x_2).$$

ii. $\mathcal{F}' = \{F'_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n\}_{n\in\mathbb{N}}$ where

$$F'_n(k, x) = F_n(k, x) \oplus x.$$

3. **PRG Security does not imply Related-Key-PRG Security (7 marks)**

Whenever a PRG is used, the seed must be chosen afresh. However, sometimes, the seed is re-used with minor modifications, and this could lead to security attacks. Such attacks have been implemented in practice (against the RC4 stream cipher). First, let us formally define security with respect to related-key attacks (we will look at a special form of related-key attacks, although the security game can be modified to capture other related-key attacks).

Let $\mathcal{G} = \left\{G_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}\right\}_{n\in\mathbb{N}}$ be a family of deterministic length-expanding functions. We will define PRG Security against related-key attacks via the following security game between a challenger and an adversary:

PRG security w.r.t. Related Key Attacks

- Challenger picks a uniformly random bit $b \leftarrow \{0,1\}$ and a seed $s \leftarrow \{0,1\}^n$.
- The adversary makes polynomially many queries to the challenger. For each query, the challenger does the following:
  - if $b = 1$, it picks a uniformly random string $y \leftarrow \{0,1\}^{\ell(n)}$ and sends it to $\mathcal{A}$. If $b = 0$, it first computes $y = G_n(s)$ and sends y to the adversary.
  - it updates s as follows: interpret s as a number in $\{0, 1, \ldots, 2^n - 1\}$, set $s = s + 1 \bmod 2^n$.
- After polynomially many queries, the adversary sends its guess b' and wins if $b = b'$.

Figure 1: Security game for capturing related-key attacks against PRGs

We say that G is secure against related-key attacks if, for any p.p.t. adversary $\mathcal{A}$, there exists a negligible function μ such that for all n,

$$\mathsf{Pr}\left[\mathcal{A} \text{ wins the above game against } \mathcal{G}\right] \leq \frac{1}{2} + \mu(n).$$

Let $\mathcal{G} = \left\{G_n : \{0,1\}^n \to \{0,1\}^{3n}\right\}_{n\in\mathbb{N}}$ be a secure pseudorandom generator. Use $\mathcal{G}$ to construct a new length-expanding function $\mathcal{G}'$ (with appropriate input and output space) such that $\mathcal{G}' = \{G'_n\}_{n\in\mathbb{N}}$ is a secure PRG, but not secure against related-key attacks.

(a) Describe $\mathcal{G}'$ in terms of $\mathcal{G}$. You can choose appropriate input and output space for G'_n; the function must be length-expanding.

(b) Describe the related-key attack against $\mathcal{G}'$, and analyze the adversary's winning probability.

(c) Show that if there exists a p.p.t. adversary $\mathcal{A}$ that breaks the PRG security of $\mathcal{G}'$, then there exists a p.p.t. reduction algorithm $\mathcal{B}$ that breaks the PRG security of $\mathcal{G}$.

4. **Constructing PRFs from PRGs (10 marks)**

In class, we saw that pseudorandom functions can be used to build pseudorandom generators. The converse also holds, and we will explore this direction below. Let $\mathcal{G} = \left\{G_n : \{0,1\}^n \to \{0,1\}^{2n}\right\}_{n\in\mathbb{N}}$ be a length-doubling secure pseudorandom generator family. We will use $\mathcal{G}$ to construct a pseudorandom function family

$$\mathcal{F} = \left\{F_n : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^n\right\}_{n\in\mathbb{N}}$$

as follows:

PRG $\to$ PRF

The PRF evaluation using key $k \in \{0,1\}^n$, on input $x \in \{0,1\}^{\log n}$ is defined as follows:

1. Let $s = k$. For $i = 1$ to $\log n$, do the following:
   a. Compute $(s_0, s_1) = G_n(s)$, where s_0 and s_1 are both n-bit strings.
   b. Set $s = s_{x[i]}$.
2. Output s.

Figure 2: Constructing PRF using PRG

(a) Prove that $\mathcal{F}$ is a secure pseudorandom function, assuming $\mathcal{G}$ is a secure pseudorandom generator.

First, carefully define the hybrid experiments for your proof. Then, show that the consecutive hybrids are computationally indistinguishable, assuming security of $\mathcal{G}$.

(b) The above construction can be easily extended to support input space $\{0,1\}^n$. This construction is also secure, but would your proof from Part 4a work for this construction?

(c) The above construction can also be easily modified to support input space $\{0,1\}^*$. For completeness, we provide this modified construction in Figure 3 below.

PRG $\to$ PRF: variable length inputs

The PRF evaluation using key $k \in \{0,1\}^n$, on input $x \in \{0,1\}^*$ is defined as follows:

1. Let $s = k$, and let ℓ denote the length of x. For $i = 1$ to ℓ, do the following:
   a. Compute $(s_0, s_1) = G_n(s)$, where s_0 and s_1 are both n-bit strings.
   b. Set $s = s_{x[i]}$.
2. Output s.

Figure 3: A candidate PRF construction for handling variable length inputs

This construction is **not secure**. Present a polynomial time adversary that breaks the above PRF construction, and calculate the winning probability of your adversary.

(d) Modify the construction in Figure 3 so that the resulting scheme has input domain $\{0,1\}^*$, key space $\{0,1\}^n$, output space $\{0,1\}^n$, and it satisfies PRF security. You can assume the existence of secure PRG family $\mathcal{G} = \left\{G_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}\right\}_{n\in\mathbb{N}}$ where ℓ is a suitably chosen polynomial. Informally explain briefly why your construction is plausibly secure. You don't need to provide a formal proof here (although there exist constructions with provable security; see Question Part B.2.).

Part B. Coding/Theoretical Problems (8 marks)

Part B.1. Coding Problems

1. **CRIME Attack (4 marks)**

   The CRIME attack is a notable example of how seemingly unrelated elements, like compression and encryption, can interact to create vulnerabilities that attackers can exploit.

   Context: The HTTP protocol is one of the most widely used protocols over the Internet. This protocol is *stateless*, and therefore if a server and client must interact over multiple messages in one session, the client stores the state in the form of *cookies*, and includes this as part of the client message. Quite often, these cookies contain sensitive information, and therefore, the server and client's messages must be encrypted. This is achieved via the HTTPS protocol, which is an extension of the HTTP protocol. At a very high level, the server and the client share a secret key k.[1] Whenever the client wants to send a message cmsg, it first appends the current cookie, then encrypts "cookie || cmsg" using k. The server decrypts this ciphertext using k, then computes its response and then sends the encrypted response. The client receives this ciphertext, decrypts it, and then updates its cookie, and computes the next message. For simplicity, let us assume the client's cookie is not updated throughout the interaction.

   Certain protocols came up with the following optimization. Instead of sending an encryption of "cookie || cmsg", they instead suggested the use of a (lossless) compression algorithm to first compress "cookie || cmsg", and then encrypt the resulting compressed string. The server then first decrypts the ciphertext, then runs the decompression algorithm. While this does result in noticeably shorter ciphertexts (and hence lesser communication cost), it opens up a serious vulnerability, which was called the **CRIME (Compression Ratio Info-leak Made Easy)** attack.

   The adversary's goal, and the adversary' power: The adversary's goal is to learn the 'cookie'. It cannot break the encryption. However, the adversary has the power to influence what cmsg is sent. We will not discuss how the adversary manages to influence cmsg. [2]

   To simplify the setup, we will assume the adversary can get an encryption of the compressed string compress(cookie || cmsg) for any cmsg of its choice. The important thing to note here is that, depending on the cmsg, the size of the compressed string may vary, and as a result, the size of the ciphertext will vary! This leaks information about the cookie. In particular, if cmsg contains a substring of the cookie, then the compressed string is shorter. Using this, the adversary can query for sufficiently many cmsg strings, and based on the size of the compressed string, learn the entire cookie.

[1] We will discuss later in the course how the server and client arrive at a common shared key.

[2] This is beyond the scope of this course, however you are encouraged to read about the CRIME attack.

Problem Description: You need to implement the CRIME attack to find the 'cookie'. The length of the 'cookie' is fixed to be **24 bytes** for this assignment. The attack is based on the properties of the compressor being used.

Files Given: You are given the following python files on Teams (COL759_A1_Coding1.zip):

- encrypt.py:
  - This file has a 24-byte secret 'cookie' and 16-bit key for the AES scheme hard-coded into it
  - It has a function called encrypt(m), which takes a string m containing only [a-z] and returns the encrypted bytes using the key and the secret 'cookie'.
  - This script can be used to generate ciphertexts with the given key and the secret 'cookie'. You can use it to check the correctness of your code.
- attack.py:
  - You are required to implement the attack function in this file.
  - It is supposed to return the 'cookie' which is used during the encryption (as a string).
  - You are allowed to make calls to encrypt(m) function of encrypt.py.

Instructions:

- You would need to install the python3 and pycryptodome python packages to run the given files. Installation instructions can be found on this link.
- You are **only** required to submit attack.py on Gradescope in the Assignment1 Coding1 assignment, with your implementation of attack(). You don't need to submit any other files. All test cases are public; you should be able to find the number of test cases that your code passes on Gradescope.
- Provide a high-level description of your approach in the pdf submission.

Hint: You are expected to try various configurations for your attack in order to find some heuristic which works for all test cases - in reality cryptanalysis requires a lot of manual effort as well!

2. **Attack on 2DES encryption (4 marks)**

Data Encryption Standard(DES) is a symmetric-key encryption scheme that was developed by IBM in the 1970s. Over the years, its susceptibility to numerous attacks, particularly brute-force attacks, has been revealed. Consider the following variant of the DES scheme used in our problem:

- $\mathsf{DES.Enc}(m, k)$*:* This is the encryption function. In our case, the key k is restricted to be of $n = 20$ bits. DES can only handle messages having length in multiples of 64 bits
- $\mathsf{DES.Dec}(\mathsf{ct}, k)$*:* This is the decryption function. The key k is restricted to be of $n = 20$ bits. The input ciphertext ct must have a length in multiple of 64 bits.

A variant of the DES scheme known as 2DES attempts to enhance security by applying 2 rounds of DES Encryption sequentially. In this problem, our attack will be on the 2DES scheme defined below:

- $\mathsf{2DES.Enc}(\mathsf{k} = (\mathsf{k}_1, \mathsf{k}_2), \mathsf{m}) = \mathsf{DES.Enc}(\mathsf{k}_2, \mathsf{DES.Enc}(\mathsf{k}_1, \mathsf{m}))$
- $\mathsf{2DES.Dec}(\mathsf{k} = (\mathsf{k}_1, \mathsf{k}_2), \mathsf{ct}) = \mathsf{DES.Dec}(\mathsf{k}_1, \mathsf{DES.Dec}(\mathsf{k}_2, \mathsf{ct}))$

Unfortunately, as discussed in class, the above 2DES scheme does not provide any extra security over the original DES protocol due to "meet in the middle" attack on 2DES. You are required to implement this attack.

Files Given: You are given the following python files on Teams COL759_A1_Coding2.zip:

- des.py:
  - It has three functions. The first one is key_gen(index) which takes an index and returns you the corresponding key. For example, if n (the number of bits of key) = 2 then there would be 4 keys so you can access these 4 keys by giving indices from 0 to 3.
  - Second function is encrypt(key, message) which takes key and a string message to return the encrypted message using a single DES.
  - Third function is decrypt(key, message) which takes key and a ciphertext to return the message corresponding to this ciphertext using decryption of single DES.
  - You can use these function to implement the attack.
- attack.py:
  - You are required to implement the attack(message, ciphertext) function in this file which takes a message and corresponding ciphertext generated using 2DES.
  - It is supposed to return the two keys which are used during the encryption (as a tuple). So for example, if the keys are key1, key2 (key1 is used first and then key2) then you have to output (key1, key2) and not (key2, key1) or anything else.

- You are allowed to make calls to encrypt(key, message) and decrypt(key, CT) functions of des.py.

Instructions:

- You would need to install the python3 and pycryptodome python packages to run the given files. Installation instructions can be found on this link
- You are only required to submit attack.py, with your implementation of attack(message, ciphertext). You don't need to submit any other files.
- Submit the attack.py file on Gradescope in the Assignment1 Coding2 assignment
- Your submission will be checked on multiple keys pair and on multiple messages. All test cases will be public.

Part B.2. Theoretical Problem (8 marks)

Provide a detailed security proof for your PRF construction in Question 4d. As a first step, understand the security proof for the construction in Question 4b (you can refer to Theorem 4.10 in the textbook for this). Next, use a similar argument and show that your construction in Question 4d is secure, assuming G is a secure PRG.