COL759 Cryptography Semester 1, 2023-24

Problem Set 2

Instructor: Venkata K *Due Date: 27 September 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 theoretical/coding 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.
- For the coding questions, you would need to install the `pycryptodome` python3 package to run the given files. Installation instructions can be found on this link.
- 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. (8 marks) **Cryptosystems secure against side-channel attacks**[1]

   When we were discussing the padding oracle attack, we also talked about *timing attacks* — attacks that exploit the amount of time required for decryption. As was pointed out by one of the students in class, it is important to consider such attacks for all cryptographic primitives.

   Timing attacks are a special case of a broad class of vulnerabilities called *side-channel attacks*. These include attacks based on power analysis, studying the amount of electromagnetic radiation, etc. How do we capture such attacks in a theoretical security game? This is done by allowing the adversary to leak some bounded information about the secret key. In this problem, we define leakage resilience for pseudorandom functions.

Security Game for Leakage resilient PRFs

Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a keyed function.

(a) Challenger chooses a bit $b \leftarrow \{0,1\}$ and key $k \leftarrow \mathcal{K}$. If $b = 0$, challenger sets function $F_0 \equiv F(k, \cdot)$. If $b = 1$, challenger chooses a truly random function $F_1 \leftarrow \mathsf{Func}[\mathcal{X}, \mathcal{Y}]$.

(b) **Leakage query:** First, the adversary can query for any ℓ bits of the key k. The challenger sends the corresponding bits. [a] Note that even if $b = 1$, the challenger sends these ℓ bits of the key k.

(c) **PRF queries:** After receiving the leakage, the adversary can send polynomially many queries (adaptively). For each query x_i, the challenger sends $F_b(x_i)$.

(d) $\mathcal{A}$ finally sends a guess b' and wins if $b' = b$.

[a] This leakage represents the information that the adversary learns about the secret key. The general leakage-resilience security game allows the adversary to receive any arbitrary function of the secret key. Here, we are working with a weaker definition where the adversary gets a subset of the key's bits.

Figure 1: ℓ-Leakage Resilient PRF

A keyed function F is said to be an ℓ-leakage resilient PRF if no ppt adversary can win the above security game with non-negligible advantage.

Having defined leakage resilience, a natural question is whether we get leakage resilience *for free*. That is, given any secure PRF, is it already ℓ-leakage resilient for some $\ell > 0$? Unfortunately, no! There exist secure PRFs where leaking even one bit of the key breaks PRF security.

Let $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a secure PRF. Construct a new PRF F' with appropriate key space $\mathcal{K}'$, input space $\mathcal{X}'$ and output space $\mathcal{Y}'$ such that the F' is a secure PRF (assuming F is); however, if the adversary learns even one bit of the key, then the PRF is no longer secure.

[1] This problem is taken from the textbook ([BS23], 4.14). You can refer to the textbook for hints.

(a) Prove (using formal reduction(s)) that if F is a secure PRF, then F' is also a secure PRF.

(b) Show that F' does not satisfy 1-leakage resilience.

2. (8 marks) **MACs: unique queries vs non-unique queries**

In the minor, we saw how to convert a MAC with randomized signing into one which has deterministic signing. This also ensures that if an adversary sends repeated queries for the same message, it gets the same response (and therefore repeated queries are useless when the signing algorithm is deterministic). However, is this true in general?

Consider the UFCMA security game for MACs (Figure 18 in the lecture notes), but the adversary is allowed only **unique** signature queries. A MAC scheme is said to be *unforgeable against unique chosen message attacks* (UFCMA-Unique) if, for any ppt adversary $\mathcal{A}$, the advantage in this restricted security game is negligible. This security game is strictly weaker than the regular MAC security game.

Let $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a secure pseudorandom function. Use F to construct a MAC scheme $\mathcal{I}_{\text{uq}} = (\mathsf{Sign}_{\text{uq}}, \mathsf{Verify}_{\text{uq}})$ with message space $\{0,1\}^n$, and appropriate key space that is UFCMA-Unique secure, but not UFCMA secure.

(a) Describe the signing and verification algorithms.

(b) Show that the MAC scheme $\mathcal{I}_{\text{uq}}$ is UFCMA-Unique secure, assuming F is a secure pseudorandom function. If your security proof requires multiple hybrid-games, describe the games formally, and complete the proof by giving appropriate reductions.

(c) Show that the MAC scheme $\mathcal{I}_{\text{uq}}$ is not UFCMA secure. Construct a ppt adversary $\mathcal{A}$ that wins the UFCMA security game against $\mathcal{I}_{\text{uq}}$ with non-negligible advantage. Analyse the success probability of $\mathcal{A}$.

3. (2 marks) **A mistake in the lecture notes**

In class, we discussed two definitions for MACs: one with verification queries, and one without. We argued that if there exists a p.p.t. adversary that makes q_v verification queries and wins with probability ϵ, then there exists a p.p.t. adversary that makes no queries and wins with probability at least ϵ/q_v.

Based on one of the in-class suggestions, I had included the following in the notes:

> If every message has a unique signature, then verification queries do not give any additional power to the adversary. For any p.p.t. adversary $\mathcal{A}$ that wins the security game in Figure 19 with probability ϵ using q_s signing queries and q_v verification queries, there exists a p.p.t. reduction $\mathcal{B}$ that wins the security game in Figure 18 with probability ϵ using $q_s + q_v$ signing queries. On receiving a verification query (m, σ), the reduction algorithm $\mathcal{B}$ sends m to the challenger, and receives a signature σ'. If $\sigma = \sigma'$, then it sends 1, else it sends 0.

In the above (informal and flawed) argument, we did not use the fact that $\mathcal{A}$ is p.p.t. As a result, using the same argument, we can state that for any adversary $\mathcal{A}$ that runs in time t, makes q_s signing queries and q_v verification queries and wins with probability ϵ, there exists a reduction that makes at most $q_s + q_v$ signing queries, runs in time $t + \mathsf{poly}(\lambda) \cdot (q_s + q_v)$ and wins with probability ϵ.

Show that this argument is incorrect. Let $\mathcal{I}$ be a MAC scheme where each message has a unique valid signature. Propose an adversary that runs in time t, makes q_s signing queries, q_v verification queries and wins with probability 1 (note that here we did not assume t, q_s or q_v are polynomial). However, the above reduction does not win with probability 1.

The important take-home message here is that one should avoid informal arguments (especially when dealing with MACs). Thanks to one of the students for bringing this to my attention.

4. (5 marks) **Even-Mansour instantiated with a bad permutation**

Even-Mansour cipher has been proved to be secure in the *ideal permutation model*. However, using a "bad" permutation may lead to attacks in the real world. In this question, your goal is to understand the given implementation, find vulnerabilities based on the public permutation being used and code an attack to break PRP security of the scheme using this permutation. Hopefully, this question successfully conveys that security in the ideal permutation model does not translate directly to security in the real world and that one needs to be rather careful while implementing these in practice.

Files Given: You are given the following Python files on Teams (`COL759_A2_Coding1.zip`)

- `perm.py`:
  - This file has a `prp_oracle` class which has randomly generated large prime p and two keys k_1, k_2 which is used by the scheme.
  - It has two functions that you can use. The first one is called `pi_func(x)` which takes an input number x between 0 to $p-1$ and returns $\pi(x)$.
  - The second function is called `oracle(x)` which takes an input number x between 0 to p-1 and
    - If $b = 0$ then it outputs PRP evaluation on x.
    - If $b = 1$ then it outputs random permutation evaluation on x.
  - There is also a third function which you don't have to use - `prp(x)`. It takes an input number x between 0 to $p-1$ and outputs the PRP evaluation on x. You can look at it to understand how the PRP is implemented.
- `attack.py`:
  - You are required to implement the `attack(oracle, pi_func, p)` function in this file.
  - It is supposed to return the correct guess of b (either 0 or 1) which is sampled by the challenger at the start.
  - You are allowed to make at most five calls to oracle(x). After five calls, the oracle will return `None`.

Instructions:

- You are only required to submit `attack.py` on Gradescope in the `Assignment2 Coding1` with your implementation of `attack(oracle, pi_func, p)`. 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 attack in the pdf submission.

NOTE: Your grade for this problem will be based on how many queries you will use.

5. (4 marks) **3-round Luby-Rackoff with inversion queries**

In class we discussed that the 3-round Luby-Rackoff scheme is a secure PRP. One of the students asked whether there's any benefit in having more rounds. Having more rounds helps us achieve stronger security. In particular, the 4-round construction is a *strong pseudorandom permutation*, but the 3-round construction is not. Below, we present the definition of strong PRPs. Your task is to show that the 3-round construction is NOT a strong PRP.

Definition 1 *Strong PRP: A keyed function $F : \mathcal{K} \times \mathcal{X} \mapsto \mathcal{X}$, together with an inverse function $F^{-1} : \mathcal{K} \times \mathcal{X} \mapsto \mathcal{X}$, is a strong pseudorandom permutation (sPRP) if F, F^{-1} are efficient, and the following correctness and security properties hold:*

- *Correctness: for any $k \in \mathcal{K}$, $x \in \mathcal{X}$, $F^{-1}(k, F(k, x)) = x$.*
- *Security: for any p.p.t. adversary $\mathcal{A}$, $\left|\Pr[\mathcal{A} \text{ wins the strong PRP security game}] - \frac{1}{2}\right|$ is negligible, where the strong PRP security game is defined in figure below.*

Security Game: Strong PRP

(a) Challenger chooses a bit $b \leftarrow \{0, 1\}$. If $b = 0$, challenger chooses a key $k \leftarrow \mathcal{K}$ and sets function $F_0 \equiv F(k, \cdot)$. If $b = 1$, challenger chooses a truly random function F_1 from the set of all permutations mapping $\mathcal{X}$ to $\mathcal{X}$.

(b) The adversary can make the following two types of queries:

- Permutation: For query $x \in \mathcal{X}$, the challenger sends $F_b(x)$.
- Inverse: For query $x \in \mathcal{X}$, the challenger sends $F_b^{-1}(x)$

(c) After polynomially many queries by the adversary, it finally sends a guess b' and wins if $b' = b$.

Figure 2: Strong PRP

Specifically, given access to only the permutation, it is impossible to break security of the 3 round Luby-Rackoff scheme. But, you have been provided oracle access to the permutation as well as its inverse. Your goal is to break sPRP security of the Luby-Rackoff scheme.

Files Given: You are given the following Python files on Teams (`COL759_A2_Coding2.zip`)

- `attack.py`: Refer to the comments in this file for more information

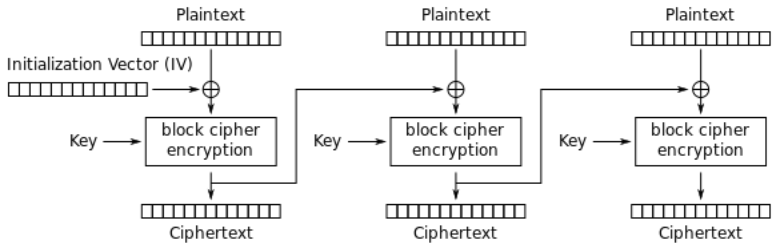
Instructions:

- You are only required to submit `attack.py` on Gradescope in the `Assignment2 Coding2` with your implementation of `predict(permute, inverse_permute)`. 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 attack in the pdf submission.

NOTE: Your grade for this problem will be based on how many queries you will use.

6. (5 marks) **CBC mode with bad initialization**

Problem Description: CBC mode is a commonly used provably secure encryption scheme. While the encryption scheme is in itself simple, it is still possible for things to go horribly wrong in practice when in order to make the implementation of the scheme simpler, implementations end up using "bad" initialization vector (IV), such as directly picking up the last block of the previous ciphertext or even the key itself. In this problem, we will explore an attack on the latter implementation. Hopefully, this shows that something as simple as choosing the IV can turn out to be fatal if done incorrectly and that a random IV is the correct way to use CBC mode of encryption.



Cipher Block Chaining (CBC) mode encryption

For the purpose of this question, consider the IV supplied at the start to be same as the key used with the block cipher(AES).

You are given an encryption of some message, and you are supposed to recover the secret key by making only one query to the decryption oracle.

How this attack is applicable in the real world — consider the following scenario: there is a server that communicates with clients using a shared secret key. Whenever the server receives a 'bogus' ciphertext (that is, a ciphertext that results in some nonsensical decryption), it outputs an error message containing the decrypted message (this happens quite often in practice when code for debugging is pushed into production). If the server receives a proper ciphertext, it computes the next message and outputs an encryption of the next message.

Therefore, the attacker has access to ciphertexts, and also has access to a partial 'decryption oracle'. In this assignment, we are giving you access to the full decryption oracle.

Files Given: You are given the following Python files on Teams (`COL759_A2_Coding3.zip`)

- `decrypt.py`:
  - This file has a `decryption_oracle` class which has 16-byte key for the AES scheme hard-coded (the keys will be changed during autograding)
  - It has a function called `decrypt(ct)`, which takes a byte array of size 48 and returns the decrypted message m of 48 bytes in the byte format.
    An example of how the message and the ciphertext (format of both) will look like is given below(not indicative of actual size used in this):
    m =`b'd2\xad\xbc\x8d\xc1R\xcc\xa3\x8c\x9d\x10\x8d\xfc'`

$ct = [137, 110, 41, 18, 99, 62, 3, 4, 197, 186, 163, 215, 12, 48, 176, 44]$

- This `decrypt` function can be used only once for your attack, If you use it for more than once then it will just return None.

- `attack.py`:
  - You are required to implement the `attack(ciphertext, decrypt)` function in this file.
  - You are given a cipher on some message of 48 bytes length(fixed).
  - It is supposed to return the key which is used during the encryption and decryption of the scheme.
  - You have to return the key in byte format only.
  - You are allowed to make calls to `decrypt(ct)` function of `decrypt.py`.

Instructions:

- You are only required to submit `attack.py` on Gradescope in the `Assignment2 Coding3` with your implementation of `attack(ciphertext, decrypt)`. 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 attack in the pdf submission.

Part B. Coding/Theoretical Problems (8 marks)

Part B.1. Coding Problem: Padding Oracle Attack

Consider the following encryption scheme:

- $\mathsf{Enc}(k, m)$: The message m is an arbitrary sequence of bytes. Here the key k is a 16 byte string. We use AES in CBC mode to encrypt this message. Since AES can only handle messages whose length (in bits) is a multiple of 128, we have to pad m appropriately.

  Padding Scheme: Instead of padding at the right-most end, we would instead pad at the left-most end (as suggested by one of the students in class). Let p be the number of bytes to be padded – then include the number p (in binary) in each of the p bytes.

  Examples:

  - $m = (11\ 42\ 33\ 01\ 89\ 12)$. This message is 6 bytes long. We need to pad it with 10 bytes. The resulting message m' would be

    $$m' = (10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 11\ 42\ 33\ 01\ 89\ 12).$$

  - $m = (02\ 02\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 16)$. This message is 16 bytes long. Add a new padding block to avoid ambiguity. The padded message

    $$m' = (16\ 16\ \ldots 16\ 16\ 02\ 02\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 16).$$

- $\mathsf{Dec}(\mathsf{ct}, k)$: Decryption algorithm simply decrypts the ciphertext to obtain the padded message $m' = (y_1, y_2, ..., y_\ell)$ where each y_i is 16 bytes long. It checks if y_1 is a valid padded string. That is, check that the first byte of y_1 is a number between 1 and 16. If the number is z, then check that the next $z - 1$ bytes after it have the value z. If any of these is violated, output "Error: Bad Padding". Otherwise, output the decrypted string (without the padding).

Files Given: Since the messages and the cipher-texts are arbitrary sequences of bytes, we represent each of them by a list of integers within the range [0, 255]. You are given the following `python` files on MS Teams (`A2_Coding_Students.zip`):

- `encrypt.py`:
  - This file has a 16-bit key hardcoded into it for the AES scheme
  - It has a function called `encrypt(message)`, which takes a list of integers as input, pads it appropriately and returns the encrypted bytes
  - This script can be used to generate cipher-texts with the given key. You can use it to check the correctness of your code.
- `decrypt.py`:
  - This file has the same key hardcoded for AES as in `encrypt.py`

  - It has a function called `check_padding(encd)`, which takes a ciphertext as input (in the form of an integer list), decrypts it and checks for a valid padding.
  - It returns 0 if it was a valid padding, else returns 2. It does NOT return the decrypted message

- `attack.py`:
  - You are required to implement your attack in this file. The function to be implemented is `attack(cipher_text)`
  - It is supposed to take a ciphertext as input, and the return the original message (as a list of integers)
  - You are allowed to make calls to `check_padding()` from `decrypt.py`

Instructions:

- You are only required to submit `attack.py`, with your implementation of `attack()`. You don't need to submit `encrypt.py`, `decrypt.py`.
- Submit the `attack.py` file on Gradescope in the `Assignment2 Coding4`
- Your submission will be checked on ciphertexts which are encryptions of some messages with some key k
- During grading, we would change the key so you cannot simply decrypt the ciphertext

Part B.2. Theoretical Problem (8 marks)

Consider a variant of the CPA security definition for secret key encryption, called CPA-Unique. In this security definition, the adversary is allowed to send only unique encryption queries. The challenge messages, however, could be one of the queried messages.

Let F be a secure pseudorandom function. Let $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ be a CPA secure encryption scheme with keyspace $\{0,1\}^n$ and message space $\{0,1\}^n$. Construct an encryption scheme $\mathcal{E}_{\text{uq}} = (\mathsf{Enc}_{\text{uq}}, \mathsf{Dec}_{\text{uq}})$ such that the scheme is CPA-Unique secure, but not CPA secure.

1. Formally define the keyspace and the algorithms of $\mathcal{E}_{\text{uq}}$, using Enc, Dec and F only. Note that the domain and range of F can be chosen appropriately.
2. Show that $\mathcal{E}_{\text{uq}}$ is CPA-Unique secure, assuming $(\mathsf{Enc}, \mathsf{Dec})$ is CPA secure and F is a secure pseudorandom function. Define appropriate hybrid experiment(s) for this proof.
3. Show that $\mathcal{E}_{\text{uq}}$ is not CPA secure by constructing a ppt algorithm $\mathcal{A}$ that wins the CPA security game against $\mathcal{E}_{\text{uq}}$ with non-negligible advantage.

References

[BS23] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography.* 2023.