**EE 720: An Introduction to Number Theory and Cryptography (Spring 2020)**

Lecture 10 — February 17, 2020

*Instructor: Saravanan Vijayakumaran* *Scribe: Saravanan Vijayakumaran*

# 1 Lecture Plan

- Describe the padding oracle attack
- Define message authentication codes

# 2 Padding Oracle Attack

- Recall the CBC block cipher mode used encrypt plaintext whose length is longer than the block length of a block cipher.
  - Let $m = m_1, m_2, \dots, m_l$ where $m_i \in \{0,1\}^n$.
  - Let $F$ be a length-preserving block cipher with block length $n$.
  - A uniform *initialization vector (IV)* of length $n$ is first chosen.
  - $c_0 = IV$. For $i = 1, \dots, l$, $c_i := F_k(c_{i-1} \oplus m_i)$.
  - For $i = 1, 2, \dots, l$, $m_i := F_k^{-1}(c_i) \oplus c_{i-1}$.
- The above scheme assumes that the plaintext length is a multiple of $n$. The plaintext is usually *padded* to satisfy this constraint. For convenience we will refer to the original plaintext as the *message* and the result after padding as the *encoded data*.
- A popular padding scheme is the PKCS #5 padding.
  - Assume that the original message $m$ has an integral number of bytes. Let $L$ be the blocklength of the block cipher in bytes.
  - Let $b$ denote the number of bytes required to be appended to the original message to make the encoded data have length which is a multiple of $L$. Here, $b$ is an integer from 1 to $L$ ($b = 0$ is not allowed).
  - We append to the message the integer $b$ (represented in 1 byte) repeated $b$ times. For example, if 4 bytes are needed then the `0x04040404` is appended. Note that $L$ needs to be less than 256. Also, if the message length is already a multiple of $L$, then $L$ bytes need to be appended each of which is equal to $L$.
- The encoded data is encrypted using CBC mode. When decrypting, the receiver first applies CBC mode decryption and then checks that the encoded data is correctly padded. The value $b$ of the last byte is read and then the final $b$ bytes of the encoded data is checked to be equal to $b$.

- If the padding is incorrect, the standard procedure is to return a “bad padding” error. The presence of such an error message provides the an adversary with a *partial* decryption oracle. While this may seem like meaningless information, it enables the adversary *to completely recover the original message for any ciphertext of its choice.*
- See pages 99–100 for a complete description of the attack.
- One solution is to use message authentication codes.

# 3 Message Authentication Codes

- The main goal of cryptography is enabling secure communication between parties over an open communication channel. In addition to message privacy, secure communication entails *message integrity or authentication.*
- Each party should be able to check that a message it receives was sent by the party claiming to send it and that it was not modified in transit.
- Consider a scenario when a bank receives a request to transfer amount $N$ from account $X$ to account $Y$.
  - Is the request authentic? Did the owner of account $X$ really raise the request?
  - Assuming the request is authentic, are the details exactly as specified by the owner of account $X$? Was the transfer amount modified?
- Message authentication codes prevent *undetected tampering* of messages sent over an open communication channel.
- In general, encryption schemes do not ensure message integrity. For example, given $c := G(k) \oplus m$, where $k$ is a secret key and $G$ is a pseudorandom generator, flipping a bit in $c$ will flip the corresponding bit in the decrypted plaintext.

## 3.1 The Syntax of a Message Authentication Code

- We will continue to assume that the communicating parties share a secret key.
- When Alice wants to send a message $m$ to Bob, she computes a `MAC tag` $t$ based on the message and the shared key. Let `Mac` denote the *tag-generation algorithm.* Alice computes tag $t \leftarrow \texttt{Mac}_k(m)$ and send $(m, t)$ to Bob.
- Upon receiving $(m, t)$, Bob verifies that $t$ is a valid tag on the message $m$ using a *verification algorithm* `Vrfy` which depends on the shared key $k$. $\texttt{Vrfy}_k(m, t) = 1$ if $t$ is a valid tag for $m$ and 0 otherwise.

**Definition.** *A* ***message authentication code*** *(MAC) consists of three PPT algorithms* (*Gen*, *Mac*, *Vrfy*) *such that:*

1. *The* ***key-generation algorithm*** *Gen takes as input the security parameter* $1^n$ *and outputs a key* $k$ *with* $|k| \geq n$*.*

2. *The **tag-generation algorithm** $\mathtt{Mac}$ takes as input a key $k$ and a message $m \in \{0,1\}^*$, and outputs a tag $t$. Since this algorithm may be randomized, we write $t \leftarrow \mathtt{Mac}_k(m)$.*
3. *The deterministic **verification algorithm** $\mathtt{Vrfy}$ takes as input a key $k$, a message $m$, and a tag $t$. It outputs a bit $b$, with $b = 1$ meaning **valid** and $b = 0$ meaning **invalid**. We write this as $b := \mathtt{Vrfy}_k(m, t)$.*

*It is required that for every $n$, every key $k$ output by $\mathtt{Gen}(1^n)$, and every $m \in \{0,1\}^*$, it holds that $\mathtt{Vrfy}_k(m, \mathtt{Mac}_k(m)) = 1$.*

*If there is a function $l$ such that for every $k$ output by $\mathtt{Gen}(1^n)$, algorithm $\mathtt{Mac}_k$ is only defined for messages $m \in \{0,1\}^{l(n)}$, then we call the scheme a **fixed length MAC for messages of length** $l(n)$.*

- **Canonical verification**: For deterministic message authentication codes (i.e. where Mac is a deterministic algorithm), the canonical way to perform verification is to simply re-compute the tag and check for equality.

### 3.2 Security of Message Authentication Codes

- The intuitive idea behind the security definition is that no efficient adversary should be able to generate a valid tag on any "new" message that was not previously sent (with tag) by one of the communicating parties.
- Consider the following **message authentication experiment** $\mathtt{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)$:
  1. A key $k$ is generated by running $\mathtt{Gen}(1^n)$.
  2. The adversary $\mathcal{A}$ is given input $1^n$ and oracle access to $\mathtt{Mac}_k(\cdot)$. The adversary eventually outputs $(m, t)$. Let $\mathcal{Q}$ denote the set of all queries that $\mathcal{A}$ asked its oracle.
  3. $\mathcal{A}$ succeeds if and only if (1) $\mathtt{Vrfy}_k(m, t) = 1$ and (2) $m \notin \mathcal{Q}$. If $\mathcal{A}$ succeeds, the output of the experiment is 1. Otherwise, the output is 0.
- A MAC is secure if no efficient adversary can succeed in the above experiment with non-negligible probability.

**Definition.** *A message authentication code $\Pi = (\mathtt{Gen}, \mathtt{Mac}, \mathtt{Vrfy})$ is **existentially unforgeable under an adaptive chosen-message attack**, or just **secure**, if for all PPT adversaries $\mathcal{A}$, there is a negligible function $\mathtt{negl}$ such that:*

$$\Pr\left[\mathtt{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1\right] \leq \mathtt{negl}(n).$$

- The above definition of MAC security offers no protection against *replay attacks*. These can be prevented using sequence numbers or timestamps.

## 4 References and Additional Reading

- Section 3.7.2 from Katz/Lindell
- Sections 4.1, 4.2, 4.3 from Katz/Lindell