Lecture 21: Message Authentication Codes
In today’s lecture, we will learn about Message Authentication Codes (MACs)

We shall define security notions that we expect from such a primitive

Finally, we shall construct MACs using random functions that are secure against adversaries with unbounded computational power
A Message Authentication Scheme (MAC) is a private-key version of signatures involving two parties, the Signer and the Verifier.

- **Private-key**: This means that the signer and the verifier met yesterday and established a secret-key.
- **Signature**: This means that the verifier can verify that the signer endorses a particular message, and an eavesdropper cannot forge such endorsements.

Defined by three algorithms (Gen, Sign, Ver):

- **Secret-key Generation**: \(sk = \text{Gen}()\)
- **Signing Messages**: Compute the tag \(\tau = \text{Sign}_{sk}(m)\)
- **The Signer sends** \((m, \tau)\) to the verifier.
- **Verifying Message-tag pairs**: \(z = \text{Ver}_{sk}(\tilde{m}, \tilde{\tau}) \in \{0, 1\}\). Output \(z = 1\) indicates that the message-tag pair is accepted, while output \(z = 0\) indicates that the message-tag pair is not accepted.
Yesterday

Signer

\[ sk = \text{Gen}() \]

Verifier

\[ \tau = \text{Sign}_{sk}(m) \]

\[ \text{Send} (m, \tau) \]

Today

\[ z = \text{Ver}_{sk}(m, \tau) \]
• **No Message Secrecy**: Previously, we saw that primitives like encryption and secret sharing require hiding some information from the adversary. In MACs, the message $m$ is in the clear! We want to ensure that an adversary should not be able to generate tags that verify for new messages.

• **Secrecy of the secret key $sk$**: The secrecy of $sk$ is paramount. If the secret-key $sk$ is obtained by an adversary, then the adversary can use the signing algorithm to easily sign arbitrary messages!

  Intuitively, MACs ensure that the “holder of the secret key $sk$” endorses the message. If the eavesdropper obtains the secret-key, then the eavesdropper can endorse arbitrary messages.
Correctness

- Let the message space be $\mathcal{M}$
- Intuition: We want to ensure that the tag for any message $m \in \mathcal{M}$ that the honest signer generates should always verify
- Mathematically, we can write this as: For every message $m \in \mathcal{M}$, we have

$$
P [ z = 1 : sk = \text{Gen}(), \tau = \text{Sign}_{sk}(m), z = \text{Ver}_{sk}(m, \tau) ] = 1
$$

- English Translation: The probability that $z = 1$ is 1, where the secret-key $sk = \text{Gen}()$, the tag $\tau = \text{Sign}_{sk}(m)$, and the output $z = \text{Ver}_{sk}(m, \tau)$.
- Note that this guarantee is for every message $m$. We do not want the signing algorithm to create verifiable tags only for a subset of messages.
- The probability is over the choice of $sk$ output by the generation algorithm $\text{Gen}()$.
Message Integrity

- We want to ensure that an adversary cannot tamper the message $m$ into a different message $m'$ such that the original tag $\tau$ is also a valid tag for the adversarial message $m'$
- Let $\mathcal{T}$ be the range of the signing algorithm (i.e., the set of all possible tags)
- Message Integrity can be ensured if the following property holds. For all distinct $m, m' \in \mathcal{M}$, we have

$$\mathbb{P} \left[ \text{Sign}_{sk}(m') = \tau | \text{Sign}_{sk}(m) = \tau \right] \leq \frac{1}{|\mathcal{T}|}$$

- Note that we cannot insist on the above probability to be 0 when the set of all possible tags is smaller than the set of all messages
  Constructing an adversary who can violate message integrity with probability $\frac{1}{|\mathcal{T}|}$ is an interesting exercise.
Unforgeability

- We want to ensure that an adversary cannot forge the tag of a new message $m'$ by observing one message-tag pair $(m, \tau)$.
- Unforgeability can be ensured if the following property holds. For all distinct $m, m' \in M$, we have

$$\mathbb{P} \left[ \text{Sign}_{sk}(m') = \tau' | \text{Sign}_{sk}(m) = \tau \right] = \frac{1}{|T|}$$

- First, note that we insist that the forged message is different from the original message. That is, we have $m' \neq m$. We do not insist on $\tau' \neq \tau$.
- Next, note that unforgeability is a stronger requirement than message integrity. For example, “unforgeability restricted to $\tau = \tau''$” is identical to “message integrity.” Therefore, again, the forging probability above cannot be 0.
Suppose we want to design a MAC that remains unforgeable even when the adversary has seen \( t \) message-tag pairs. How to define \( t \)-unforgeability?
In the following slides, we will construct a MAC using Random Functions

Understand its properties and its shortcomings

Then, we shall replace the random function using a pseudorandom function family in the next lecture
Goal.

- Suppose we have $n$-bit messages, i.e., the message space is $\mathcal{M} = \{0, 1\}^n$.
- We will generate $n/100$-bit tags, i.e., the space of tags is $\mathcal{T} = \{0, 1\}^{n/100}$.
Construction using Random Functions

Scheme.

- Secret-key Generation Algorithm.
  - Let $f$ be a random function from the domain $\{0, 1\}^n$ to the range $\{0, 1\}^{n/100}$
  - Let the secret key $sk$ be the function table of $f$
  - Both the sender and the verifier will obtain the secret key $sk = f$

- Tagging Algorithm.
  - The tag $\tau \{0, 1\}^{n/100}$ for a message $m \in \{0, 1\}^n$ using the secret key $sk = f$ is computed by: $\tau = f(m)$
  - To endorse the message $m$, the sender will send the pair $(m, \tau)$

- Verification Algorithm.
  - The verifier will receive a pair $(\tilde{m}, \tilde{\tau})$
  - The verifier will check whether $\tilde{\tau} = f(\tilde{m})$ or not, where the secret-key $sk = f$
Analysis of Adversarial Forging Attack.

- Suppose the adversary sees a pair \((m, \tau)\)
- The adversary does not know the secret-key \(sk = f\), but it knows that \(f(m) = \tau\)
- Now, the adversary has to generate a different message \(m' \in \{0, 1\}^n\) and a tag \(\tau'\) such that the pair \((m', \tau')\) verifies
- The adversarial pair \((m', \tau')\) will verify if and only if \(f(m') = \tau'\)
- Let us look at this probability

\[
P [ f(m') = \tau' | f(m) = \tau ]
\]

The probability is over the choice of \(f\) from the set of all possible functions \(\mathcal{M} \to \mathcal{T}\).
Let us parse this mathematical expression. The adversary already knows the fact that “\( f(m) = \tau \)” So, we are conditioning on that fact in the probability expression. And, conditioned on this fact, we are interested in finding the probability that \( f(m') = \tau' \), where \( m' \neq m \).

First observation. Given the fact that \( f(m) = \tau \) (i.e., evaluation of a function at one input) the evaluation of \( f(m') \) is uniformly random over the range \( \mathcal{T} = \{0, 1\}^{n/100} \). Because, for a random function, given the evaluation of a function \( f \) at one input, the evaluation of the function \( f \) at any other input is uniformly random over the range.

So, conditioned on the knowledge of the adversary that \( f(m) = \tau \), the probability that \( f(m') = \tau' \), where \( m' \neq m \), is “1 divided by the size of the range.” In our case, that is

\[
\frac{1}{2^{n/100}}
\]
Therefore, we conclude

\[
\mathbb{P} \left[ f(m') = \tau' | f(m) = \tau \right] = \frac{1}{2^n/100}
\]

Note that the entire analysis remains identical if we fix \( \tau' = \tau \). Therefore, our construction preserves message integrity.
Conclusion.

- It is highly unlikely that an adversary will be able to forge a tag given one \((m, \tau)\) pair
Extension of the Unforgeability Attack.

- In fact, this scheme has an even more interesting property.
- Suppose the sender has sent several message-tag pairs. That is, the sender has sent \((m_1, \tau_1), (m_2, \tau_2), \ldots, (m_t, \tau_t)\). Note that they satisfy the following relation \(\tau_1 = f(m_1), \tau_2 = f(m_2), \ldots, \tau_t = f(m_t)\).
- The adversary has seen all these message-tag pairs. Can the adversary forge a new message-tag pair? Let us see.
Analysis of the Probability of Forging in the Extension.

- Let us write down what the adversary has seen. The adversary knows that

\[ f(m_1) = \tau_1, \quad f(m_2) = \tau_2, \ldots, f(m_t) = \tau_t \]

- Conditioned on this information, we are interested in the probability that \( f(m') = \tau' \), where \( m' \) is different from all the messages \( m_1, m_2, \ldots, m_t \)

- So, we are interested in the probability

\[ \mathbb{P} \left[ f(m') = \tau' \mid f(m_1) = \tau_1, f(m_2) = \tau_2, \ldots, f(m_t) = \tau_t \right], \]

where \( m' \notin \{m_1, m_2, \ldots, m_t\} \).
Main Observation. Even if we know the evaluation of the function $f$ at inputs $m_1, m_2, \ldots, m_t$, the evaluation of $f$ at a new input $m'$ is uniformly random over the range. So, we can conclude that the probability of forging is

$$P[f(m') = \tau' | f(m_1) = \tau_1, f(m_2) = \tau_2, \ldots, f(m_t) = \tau_t] = \frac{1}{2^{n/100}}$$
Conclusion.

- The MAC using a random function to generate tags is secure even when the adversary sees $t$ message-tag pairs.
Positive Features.

- Even if the adversary has unbounded computational power, the probability arguments bounding its probability to forge still holds.
- The scheme is secure even when the adversary has seen $t$ message-tag pairs.
Primary Shortcoming.

Let us compute the size of the function table for the function $f$. Recall that $f$ is from the domain $\{0, 1\}^n$ to the range $\{0, 1\}^{n/100}$. So, there are a total of $\left(\frac{2^n}{100}\right)^{2n} = 2^{(n/100)2^n}$ different functions. This implies that we need $(n/100)2^n$ (exponential in $n$) bits to represent this function! Even for $n = 512$, this number is larger than the number of atoms (which is $< 2^{273}$) in the entire universe.
To fix the shortcomings mentioned above, we set forth the following goals for ourselves

- We will construct functions that use a smaller key, i.e., length is polynomial in $n$

However, our security will hold only for **computationally bounded** adversaries (instead of adversaries with unbounded computational power) In the previous lecture, we have constructed pseudorandom functions, which shall serve this exact purpose!