Lecture 17: Message Authentication Codes

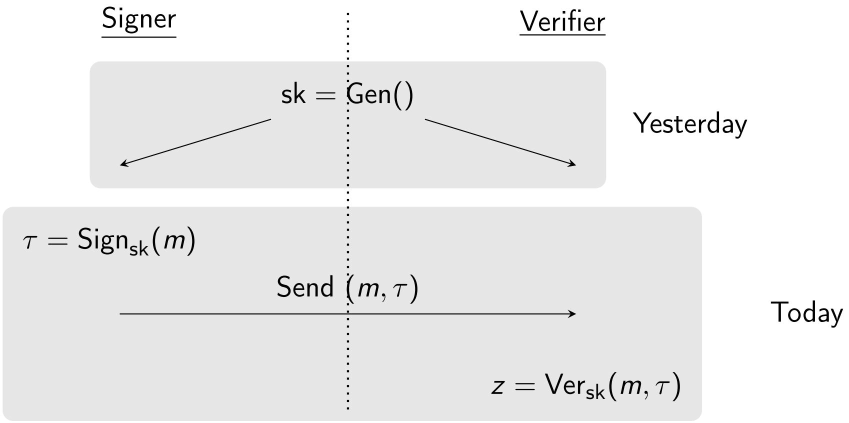
Outline

- 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 pseudorandom functions

Introduction: MAC

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

Pictorial Summary


Signer
Verifier
$\mathsf{sk} = \mathsf{Gen}()$
Yesterday
$\tau = \mathsf{Sign}_{\mathsf{sk}}(m)$
Send (m, τ)
Today
$z = \mathsf{Ver}_{\mathsf{sk}}(m, \tau)$

Comments

- No 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 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 sign 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 is generated by the honest signer should always verify
- Mathematically, we can write this as: For every message $m \in \mathcal{M}$, we have

$$\mathbb{P}\left[z = 1 \colon \mathsf{sk} = \mathsf{Gen}(), \tau = \mathsf{Sign}_{\mathsf{sk}}(m), z = \mathsf{Ver}_{\mathsf{sk}}(m, \tau)\right] = 1$$

- English Translation: The probability that $z = 1$ is 1, where the secret-key $\mathsf{sk} = \mathsf{Gen}()$, the tag $\tau = \mathsf{Sign}_{\mathsf{sk}}(m)$, and the output $z = \mathsf{Ver}_{\mathsf{sk}}(m, \tau)$.
- Note that this guarantee is for $\underline{\text{every}}$ 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 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 τ 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

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

  - 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

Unforgeability

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

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

  - Again, 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

Food for thought

- Suppose we want to design a MAC that remains unforgeable even when the adversary has seen $(k-1)$ message-tag pairs. What probability guarantee will be needed?

Motivation of the Construction I

- 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

Motivation of the Construction II

Goal.

- Suppose we have n-bit messages, i.e., the message space is $\{0,1\}^n$
- We will generate $n/100$-bit tags, i.e., the space of tags is $\{0,1\}^{n/100}$

Motivation of the Construction III

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 share the secret-key $\mathsf{sk} = F$
- Tagging Algorithm.
  - The tag $\tau\{0,1\}^{n/100}$ for a message $m\{0,1\}^n$ using the secret key $\mathsf{sk} = F$ is computed by: $\tau = F(m)$
  - To endorse the message m, the sender will send the pair (m, τ)
- Verification Algorithm.
  - The verifier will receive a pair $(\widetilde{m}, \widetilde{\tau})$
  - The verifier will check whether $\widetilde{\tau} = F(\widetilde{m})$, where the secret-key $\mathsf{sk} = F$

Motivation of the Construction IV

Analysis of Adversarial Attack.

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

$$\mathbb{P}\left[F(m') = \tau' | F(m) = \tau\right]$$

  - 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 $F(m') = \tau'$.

Motivation of the Construction V

- 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. Because, for a random function, given the evaluation of a function 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

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

Motivation of the Construction VI

Conclusion.

- It is highly unlikely that an adversary will be able to forge a tag given one (m, τ) pair

Motivation of the Construction VII

Extension.

- 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, τ_1), (m_2, τ_2), $\dots$, (m_t, τ_t). Note that they satisfy the following relation $\tau_1 = F(m_1)$, $\tau_2 = F(m_2)$, $\dots$, $\tau_t = F(m_t)$.
- The adversary has seen all these message-tag pairs. Can the adversary forge a new message-tag pair? Let us see.

Motivation of the Construction VIII

Analysis of the Probability of Forging in the Extension.

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

$$F(m_1) = \tau_1, F(m_2) = \tau_2, \ldots, F(m_t) = \tau_t$$

- 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

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

Motivation of the Construction IX

- Main Observation. Even if we know the evaluation of the function F at inputs m_1, m_2, $\dots$, 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

$$\mathbb{P}\left[F(m') = \tau' | F(m_1) = \tau_1, F(m_2) = \tau_2, \dots, F(m_t) = \tau_t\right] = \frac{1}{2^{n/100}}$$

Motivation of the Construction X

Conclusion.

- The MAC using random function to generate tags is secure even when the adversary sees t message-tag pairs (for any value of t less than the size of the range, i.e., $t < 2^n$)

Features of MAC using Random Function I

Positive Features.

- Even if the adversary has unbounded computational power, the probability arguments bounding its probability to forge still holds
- The scheme is secure for any $t < 2^{n/100}$

Features of MAC using Random Function II

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 $\left(2^{n/100}\right)^{2^n} = 2^{(n/100)2^n}$ 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.

What Next?

To fix the shortcoming mentioned above, we set forth the following goals for ourselves

- We will construct functions that use 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!

MAC using Pseudorandom Functions I

Scheme.

- Secret-key Generation. Sample sk uniformly at random from $\{0,1\}^{n/100}$ and provide sk to both the sender and the verifier
- Tagging a message $m \in \{0,1\}^n$. The sender computes tag $\tau = F_{\mathsf{sk}}(m)$ (evaluate using the GGM construction)
- Verifying a message-tag pair $(\widetilde{m}, \widetilde{\tau})$. Check whether $\widetilde{\tau}$ is same as $F_{\mathsf{sk}}(\widetilde{m})$ or not

MAC using Pseudorandom Functions II

Security

- An adversary cannot forge if it sees t message-tag pairs, where $t = \mathrm{poly}(n)$ and the adversary is computationally bounded

Subtlety I

The scheme mentioned above is secure ONLY for messages in $\{0,1\}^n$ and NOT $\{0,1\}^*$

What does it mean?

- The set $\{0,1\}^n$ represents n-bit messages, and $\{0,1\}^*$ represents arbitrary-length messages. This scheme is secure only when an adversary see message-tag pairs for messages $m_1, m_2, \ldots, m_t$ such that all of them have identical length n. Moreover, the adversary has to forge by producing (m', τ') pair such that the length of the message m' is exactly n.
- The scheme is <u>not</u> secure if the adversary can produce a message of a different length. The attack is explained in the next slide

Subtlety II

Adversarial strategy to forge a message-tag pair of different length.

- Suppose the adversary has seen a message-tag pair (m, τ) such that $\tau = F_{\mathsf{sk}}(m)$
- The adversary creates $m' = m0$ (i.e., the message m concatenated at the end with 0). The adversary computes τ' as the first half of $G(\tau)$.
- Verify that $F_{\mathsf{sk}}(m') = \tau'$
- In fact, the adversary can successfully tag any m' such that m is the prefix of m'

Lesson Learned (Very Important)

- The sender and the verifier should establish one secret-key sk for EACH length of the message that they want to sign. For example
  - They establish a secret-key $\mathsf{sk} \in \{0,1\}^k$ for 1024-bit messages and use $F_{\mathsf{sk}}(m)$ as the tag for 1024-bit messages m
  - If they want to tag 2048-bit messages, then they establish a new secret-key $\mathsf{sk}' \in \{0,1\}^k$ and use $F_{sk'}(m)$ as the tag for 2048-bit messages m
  - The verifier should <u>only</u> check the validity of the tags corresponding to 2048-bit messages using the secret-key associated with message-length 2048 (in our case, it is the secret-key sk')

Food for thought

- Suppose we want to construct a MAC so that if t-parties among a set of n-parties decide to endorse a message m, then they can add a tag that the verifier can verify. How to construct such a scheme?