MESSAGE INTEGRITY PROBLEM

CS2362 - Computer Security and Privacy

Bhumika Mittal

Contents

One of the most basic goals of cryptography is to allow two parties to communicate securely over an open communication (insecure) channel. When we talk about secure communication, we are talking about three main goals: confidentiality, integrity, and authenticity. Confidentiality ensures that the message is not read by anyone other than the intended recipient. Authenticity ensures that the message is indeed sent by the claimed sender. Integrity ensures that the message is not altered in transit.

During the communication, there is an implict assumption that the message sent by the sender is the same as the message received by the receiver. However, this is not always the case. This expectation of *message integrity* is a source of a major security problem. We need to ensure that the sender is indeed the sender (no spoofing!) and the message received is the same as the message sent.

In this report, we will discuss the problem of message integrity and how to cryptographically prevent the message from being altered in transit through an insecure channel.

1 The Problem Statement

Since we have already talked about encryption and we know that encryption ensures confidentiality, a natural question arises: why can't we use encryption to ensure integrity? The answer is that *encryption does not ensure integrity.*

Consider the following scenario: Alice sends a message to Bob. The message is intercepted by an attacker, who alters the message and sends it to Bob. Bob receives the message and decrypts it. The message is altered, but Bob has no way of knowing that. Since encryption does not ensure integrity, we need a way to ensure that the message is not altered in transit.

The message integrity is about ensuring that the message received by the receiver is the same as the message sent by the sender.

2 Why CRC-like error detection check will not work

One might think that we can use a CRC-like error detection check to ensure message integrity. However, this is not a good solution as those methods are designed to detect *random* transmission errors, not **malicious** errors.

Firstly, CRCs lack authentication, allowing attackers to modify a message and recompute the CRC without detection. Secondly, CRC functions are easily reversible and the linear/affine nature of CRC polynomials enables manipulation of both the message and its CRC without the encryption key's knowledge. Additionally, due to the linearity of the CRC, for the same length of x, y, z,

$$\mathrm{CRC}(x \oplus y \oplus z) = \mathrm{CRC}(x) \oplus \mathrm{CRC}(y) \oplus \mathrm{CRC}(z)$$

This means that even if the CRC is encrypted with a stream cipher that uses XOR as its combining operation (or mode of block cipher which effectively turns it into a stream cipher, such as OFB or CFB), both the message and the associated CRC can be manipulated without knowledge of the encryption key.

3 First Solution: $\mathbf{msg}||\mathbf{Enc}(\mathbf{msg}, k)$

At first glance, it may seem that encryption should immediately solve the problem of message authentication as well. This is due to the fact that a ciphertext completely hides the contents of the message. Therefore, it seems that an adversary cannot possibly modify an encrypted message en route; all that it sees is "random garbage". Despite its intuitive appeal, the claim that encryption solves the problem of message authentication is completely false.

3.1 Stream ciphers and message authentication

First, consider the case that a message m is encrypted using a stream cipher. That is, $E_k(m) = G(k) \oplus m$ where G is a PRG. Such ciphertexts are very easy to manipulate. Specifically, flipping any bit in c results in the same bit being flipped in m upon decryption. Thus, given a ciphertext c that encrypts a message m, it is possible to modify c to c_0 such that $D_k(c_0)$ equals $D_k(c)$ except for the least significant or any other bit which is flipped. Note that such a modification may be very useful. For example, if the message is a bank transfer, then the adversary can modify the amount of money being transferred.

3.2 Block ciphers and message authentication

The above attacks utilize the fact that flipping a single bit in a ciphertext generated via a stream cipher results in the flipping of the same bit in the decrypted plaintext. On the other hand, block ciphers seem to be significantly harder to attack. This is because a block cipher is a pseudorandom function and so flipping a single bit in the ciphertext of a block results in the entire block becoming scrambled upon decryption. Despite this, we argue that encryption with a block cipher still does not afford protection against message tampering. On the most basic level, one needs to assume that the recipient will be able to detect that one of the blocks has become scrambled. In addition to the above, we note that the ability to tamper with a message depends on the mode of operation being used.

As we have seen, encryption does not solve the problem of message authentication. Rather, an additional mechanism is needed that will enable communicating parties to know whether or not a message was tampered with. Such mechanisms are called message authentication codes. Since there is no way of preventing an adversary from modifying a message en route. The aim of a message authentication code is therefore to detect any such modification, so that modified messages can be discarded.

4 MAC and EU-CMA Security - Definitions

A message authentication code is an algorithm that is applied to a message. The output of the algorithm is a MAC tag that is sent along with the message.

Definition 4.1. A **message authentication code** or MAC is a tuple of probabilistic polynomial-time algorithms (`Gen, Mac, Vrfy`) fulfilling the following:

1. Upon input 1^n, the algorithm `Gen` outputs a uniformly distributed key k of length n: $k \xleftarrow{\$} \texttt{Gen}(1^n)$.
2. The algorithm `Mac` receives for input some $k \in \{0,1\}^n$ and $m \in \{0,1\}^*$, and outputs some $t \in \{0,1\}^*$. The value t is called the `Mac tag`
3. The algorithm `Vrfy` receives for input some $k \in \{0,1\}^n$, $m \in \{0,1\}^*$, and $t \in \{0,1\}^*$, and outputs a bit $b \in \{0,1\}$.
4. For every n, every $k \in \{0,1\}^n$, and every $m \in \{0,1\}^*$ it holds that $\texttt{Vrfy}_k(m, \texttt{Mac}_k(m)) = 1$.

If there exists a function $l(\cdot)$ such that $\texttt{Mac}_k(\cdot)$ is defined only over messages of length $l(n)$ and $\texttt{Vrfy}_k(m,t)$ outputs 0 for every m that is not of length $l(n)$, then we say that (`Gen, Mac, Vrfy`) is a fixed-length `MAC` with length parameter l.

The security of a MAC is defined by the following game:

Definition 4.2. The **existential unforgeability under a chosen message attack** or **EU-CMA** security of a MAC is defined by the following game between a challenger and an adversary. The adversary is given access to an oracle that it can query with messages of its choice. The oracle will respond with the MAC tag of the message. The adversary's goal is to output a message and a MAC tag such that the message was not queried to the oracle and the MAC tag is valid. The adversary wins if it outputs a valid message-tag pair that was not queried to the oracle. The MAC is said to be EU-CMA secure if no polynomial-time adversary can win the game with non-negligible probability.

Mathematically, the EU-CMA security of a MAC is defined by the following game:

1. $k \leftarrow \{0,1\}^n$ is chosen uniformly at random.
2. $(m,t) \leftarrow \mathcal{A}^{\texttt{Mac}_k(\cdot)}(1^n)$ is computed. The adversary is given access to the oracle $\texttt{Mac}_k(\cdot)$.
3. The adversary wins if $\texttt{Vrfy}_k(m,t) = 1$ and m was not queried to the oracle.

5 Second Solution: $\text{MAC}_\text{F} : \mathbf{msg}||\mathbf{F}_k(\mathbf{msg})$

Intuitively, if the `MAC` tag t is obtained by applying a PRF F to the message m, then the adversary should not be able to forge a valid `MAC` tag as that would be equivalent to guessing the output of the PRF. Consider the following construction:

Definition 5.1. Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^n$ be a pseudorandom function such that for every k, the function $F_k(\cdot)$ maps n-bit strings to n-bit strings. The **message authentication code** $\mathrm{MAC_F}$ is defined as follows:

1. $\mathrm{Gen}(1^n)$: Choose a random key $k \in \{0,1\}^n$.
2. $\mathrm{Mac}_k(m)$: Compute $t = F_k(m)$. If m is not of length n, then output $\perp$.
3. $\mathrm{Vrfy}_k(m,t)$: If $t = F_k(m)$, then output 1. Otherwise, output 0.

6 EU-CMA Security of $\mathrm{MAC_F}$

The MAC is said to be EU-CMA secure if for every polynomial-time adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the game is negligible.

Theorem 6.1. If F is a pseudorandom function, then $\mathrm{MAC_F}$ is EU-CMA secure.

The main idea behind the proof is that we know that F is PRF which is Ind-CPA secure and we can use this property to show that $\mathrm{MAC_F}$ is EU-CMA secure. The proof is a standard reduction argument. We assume that there exists an adversary $\mathcal{A}$ that can win the game with non-negligible probability. We then use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that can break the Ind-CPA security of F. This is a contradiction, and so we conclude that $\mathcal{A}$ cannot exist.

7 $\mathrm{MAC_F}$ as a digital analogue of a physical signatures

Digital signatures are the public-key version of MAC. They can be thought of as digital *analogue* of physical signatures. Unlike MACs, signatures are:

1. *Publicly verifiable* - anybody can verify their validity.
2. *Transferable* - recipient can show the signature to another party who can then verify that the signature is valid (this follows from public verifiability).
3. *Non-repudiable* - If Alice digitally signs a document, then Bob can prove to a third party (e.g. a court) that she signed it, by presenting the document and her signature. By definition, only Alice could have produced a valid signature.

Notice that MACs cannot have this property. None of the parties holding the key can claim the other one has signed. This is because it might be the case that the other party has actually signed.

8 Limitation: Message Size

The $\mathrm{MAC_F}$ construction has a limitation: it can only authenticate messages of a fixed length. This is because the PRF F is defined only over messages of a fixed length. This is a significant limitation as most messages are not of a fixed length. So how do we construct a secure MAC for longer messages?

The key idea is to break the message into blocks and apply a pseudorandom function to the blocks in some way. We can do the following:

1. *Apply a pseudorandom function to the first block:* This clearly is not a secure MAC because nothing prevents an adversary from changing all the other blocks apart from the first.
2. *Exclusively-OR all of the blocks and apply a pseudorandom function to the result:* In this case, all an adversary needs to do is to change the message so that the XOR of the blocks does not change (thus implying that the MAC tag remains the same). This can be carried out by changing two of the blocks so that their XOR remains the same.
3. *Apply a pseudorandom function to each block separately and output the results:* This is similar to ECB mode. In this case, no blocks can be easily modified. However, blocks can be removed, repeated, and their order can be interchanged. The method is therefore not secure. We also note that blocks from different messages can be combined into a new message.

Alternatively, we can break the message into blocks and apply a pseudorandom function to each block. For example, we want to send a message of length 256 bits through AES 128. We can break the message into two blocks of 128 bits each and apply AES to each block. The main idea is to use the tag of the previous block as the key for the next block.

$$\begin{aligned} F_k(m = m_1||m_2, k) &= F_k(F_k(m_1 \oplus 0^n) \oplus m_2) \\ &= F_k(t \oplus m_2) \\ &= F_k(t \oplus t \oplus m_1) \\ &= F_k(m_1) \\ &= t \end{aligned}$$

9 Compression

The solution to the problem of message size is to use a compression function. The idea is to compress the message into a fixed-size block and then apply a pseudorandom function to the compressed block.

The main idea is to use a compression function $C : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^n$ that takes a message and a key and compresses the message into a fixed-size block. Then, we can use the MAC_F construction to apply a pseudorandom function to the compressed block. In this case, if the compression function is not collision-resistant, then any adversary can find two messages that compress to the same block and then use the MAC_F construction to forge a valid MAC tag. Therefore, the compression function must be collision-resistant.

10 Hash Functions

Such collision-resistant compression functions are called **hash functions**. A hash function is a function that takes arbitrary-length strings and compresses them into shorter strings.

The hash function is required to have the following properties:

Definition 10.1. A hash function is said to be **collision resistant** if it is computationally infeasible to find two distinct inputs x, y such that $h(x) = h(y)$.

Definition 10.2. A hash function is said to be **pre-image resistant** if it is computationally infeasible to find an input x given its hash $h(x)$.

Definition 10.3. A hash function is said to be **second pre-image resistant** if it is computationally infeasible to find a second input y such that $h(x) = h(y)$ given an input x.

References

[1] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography.* Chapman and Hall/CRC, 2014.

[**?**]