Lecture 7 - Message Authentication Codes

Boaz Barak

October 10, 2005

Data integrity Until now we've only been interested in protecting *secrecy* of data. However, in many cases what we care about is *integrity.*

Maintaining integrity is about preventing an adversary from tampering with the data that was sent or stored by the legitimate users. For example, often people are not worried so much about secrecy of their email, but they definitely want to be assured that the email they received was indeed the one being sent.

Encryption and integrity Does encryption guarantee integrity? It might seem at first that yes: if an attacker can't read the message, how can she change it?

However, this is not the case. For example, suppose that we encrypt the message x with the PRF-based CPA-secure scheme to $\langle r, f_s(r) \oplus x \rangle$. The attacker can flip the last bit of $f_s(r) \oplus x$ causing the receiver to believe the sent message was $x_1, \ldots, x_{n-1}, \overline{x_n}$.

Checksums etc. A common device used for correcting errors is adding redundancy or checksums. A simple example is adding to x as a last bit the *parity* of x, that is $\sum_i x_i \pmod 2$.[1] When receiving a message, the receiver checks the parity, and if the check fails, considers the message corrupted (and if appropriate asks to resend it). This works against *random* errors but not against *malicious* errors: the attacker can change also parity check bit. In fact, as we saw above, the attacker can do this even if the message (including the parity check bit) is encrypted.

Message Authentication Codes (MAC) The cryptographic primitive that we use for this is a *message authentication code* (MAC). A message authentication code (MAC) consists of two algorithms $(\mathsf{Sign}, \mathsf{Ver})$ (for signing and verifying). There is a shared key k between the signer and the verifier. The sender of a message x computes $s = \mathsf{Sign}_k(x)$, s is often called a *signature* or a *tag.* Then, it sends (x, s) to the receiver. The receiver accepts the pair (x, s) as valid *only* if $\mathsf{Ver}_k(x, s) = 1$.

Security for MACs We define a MAC secure if it withstands a *chosen message attack.* (Notation: n - key length, m - message length, t - tag length)

Definition 1 (CMA secure MAC)**.** A pair of algorithms $(\mathsf{Sign}, \mathsf{Ver})$ (with $\mathsf{Sign} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^t$, $\mathsf{Ver} : \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^t \to \{0,1\}$) is a (T, ϵ)-CMA-secure MAC if:

Validity For every x, k, $\mathsf{Ver}_k(x, \mathsf{Sign}_k(x)) = 1$.

Security For every T-time Adv, consider the following experiment:

[1] Sometimes this is generalized to more bits, say, parity mod 2^{16}.

- Choose $k \leftarrow_{\mathrm{R}} \{0,1\}^n$
- Give adversary access to black boxes for $\mathsf{Sign}_k(\cdot)$ and $\mathsf{Ver}_k(\cdot)$.
- Adversary *wins* if it comes up with a pair $\langle x', s' \rangle$ such that **(a)** x' is *not* one of the messages that the adversary gave to the black box $\mathsf{Sign}_k(\cdot)$ and **(b)** $\mathsf{Ver}_k(x', s') = 1$.

Then the probability Adv wins is at most ϵ.

Naturally, we define $(\mathsf{Sign}, \mathsf{Ver})$ to be CMA-secure if for every n it is $(T(n), \epsilon(n))$-CMA-secure for super-polynomial T, ϵ. In other words, there is no polynomial-time Adv that succeeds with polynomial probability to break it.

Example As discussed above, the following are *not* MACs:

- A CPA-secure encryption scheme.
- A cyclic redundancy code (CRC)

Construction for a message authentication code. We prove the following theorem:

Theorem 1. *Let $\{f_k\}$ be a PRF. Then the following is a MAC:*

- $\mathsf{Sign}_k(x) = f_k(x)$.
- $\mathsf{Ver}_k(x, s) = 1$ *iff* $f_k(x) = s$.

Proof. We prove this in the typical way we prove constructions using PRFs are secure: we define an *ideal* MAC scheme that uses a *truly random* function, prove it secure, and then derive security for our real scheme.

Proof of security for ideal scheme. Let A be an adversary running a chosen-message attack against the ideal scheme. At the end of the attack it outputs a string x' that was *not* asked by it before from the signing oracle and some supposed tag t'. Since this is a random function, we can think of the oracle at this point choosing the tag t for x' at random and we have that $\Pr[t = t'] = 2^{-n}$. □

Authentication and secrecy In some settings, it may seem that we don't care so much about *authentication* but rather only about *secrecy*. Indeed, many early (and not so early) common cryptographic protocols did not use MACs at all and used only encryption (sometimes combined with CRCs). These include the first version of SSH (secure shell), WEP (wireless encryption protocol), and old version of the IPSEC protocol. However, it turns out that in all these cases, the lack of MACs leads to security vulnerabilities. (See Wagner's lecture 10 in CS276 2004, `http://www.cs.berkeley.edu/~daw/teaching/cs276-s04/10a.ps` and `http://www.cs.berkeley.edu/~daw/teaching/cs276-s04/10b.ps`)

We are going to give here a simple but tailored counterexample showing this:

The login problem. Assume that client and server share a secret PIN PIN that was chosen at random between 0 and 10000 (e.g. a 13 bit number). Suppose in addition they have a shared key $k \leftarrow_{\mathrm{R}} \{0,1\}^n$ between it and the server, which an adversary does not know. Now, suppose they run the following protocol: the client sends the server an encryption of PIN, and the server decrypts and verifies that this is the correct PIN. If not, the server aborts the communication.

Intuitively, the attacker has no chance to learn the PIN, since it is never sent "in the clear" but only encrypted. However, it turns out that the intuition is wrong.

Lemma 1. *There exists a CPA secure scheme* (E,D) *such that if the client and server use* (E,D) *in this protocol, an attacker that sits on the communication channel between the client and server can learn the PIN after at most* 13 *sessions.*

Proof. We saw that for some CPA-secure schemes the attacker can modify an encryption of x to an encryption of x' where x' is x with the i^{th} bit flipped. There are also such schemes where the attacker can modify an encryption of x to an encryption of $x\langle i\rangle$ where $x\langle i\rangle$ is x with the i^{th} bit made to zero (regardless of whether originally x_i was zero or one). We'll see such a scheme below.

Now the attacker can use the following strategy: for $1 \le i \le 13$, at the i^{th} session, it converts the encryption of PIN into the encryption of $PIN\langle i\rangle$ and sees whether the server aborts the communication or not. Within 13 sessions it will learn all the bits of the PIN.

Example of CPA-secure scheme (E,D) such that adversary can zero out the i^{th} bit. We define the following encryption scheme:

- Key: $s \in \{0,1\}^n$ (we'll use a PRF ensemble $\{f_s\}$)
- Encryption: to encrypt $x \in \{0,1\}^{n/2}$ encode each bit of x to two bits in the following way: if it is zero then encode it as 00 if it is one then encode it as 11. Let $\tilde{x} \in \{0,1\}^n$ be the encoded version of x. Then use the standard PRF based encryption to encrypt $\tilde{x}$. That is, send $\langle r, f_s(r) \oplus \tilde{x}\rangle$ for $r \leftarrow_{\mathrm{R}} \{0,1\}^n$.
- Decryption: to decrypt, decrypt in the usual way to get $\tilde{x}$, then decode $\tilde{x}$ in the following way: $00, 10, 01 \to 0$, $11 \to 1$.

This obviously satisfies $\mathsf{D}_s(\mathsf{E}_s(x)) = x$ and is also CPA-secure (the encoding of x to $\tilde{x}$ cannot hurt security, this is intuitive, but can also be proven easily from the definition of CPA security).

To change an encryption $\tilde{r,y}$ of x to an encryption of $x\langle i\rangle$ the attacker simply flip the $2i^{th}$ bit of y.

Why is this meaningful? On a first encounter a natural reaction to such a counterexample is that this is "cheating". This is an obviously contrived encryption scheme which no designer in his right mind would use in a login protocol. How can such an example teach us something about security?

There are several answers to this concern (in some sense these are all different ways to state the same answer):

1. Although this example is contrived, its only a simplified presentation of attacks which worked for real-world protocols such as WEP, IPSEC, SSH etc.
2. Such examples teach us what we need to assume about the underlying components that we use. If there is a login protocol that uses only encryption and not MAC, this example tells us that if the protocol is secure at all, its security is not based solely on the fact that the encryption scheme is CPA secure, but rather the protocol needs some additional property from the encryption scheme. This is important because even if the protocol is secure now, at some future date someone might decide to use a different encryption scheme for the protocol, and so it is crucial to explicitly state the security requirements from the encryption scheme used.
3. The fact that this example does not immediately imply that protocol X is insecure does not mean that protocol X is secure . The onus is always on the protocol designer to demonstrate that the protocol is secure. If the designer claims that his login protocol is secure, he should state under what conditions on the encryption scheme this will be the case.

$\square$