# Lecture 8 - Message Authentication Codes

Benny Applebaum , Boaz Barak

October 12, 2007

**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 $(T, \epsilon)$-CMA-secure MAC if:

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

**Security** For every $T$-time $\mathsf{Adv}$, 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 $\mathsf{Adv}$ wins is at most $\epsilon$.

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 $T, \epsilon$. In other words, there is no polynomial-time $\mathsf{Adv}$ 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 $\Pr[t = t'] = 2^{-n}$. □

**Using Authentication to get CCA security** As we saw last time, CPA secure encryption is not always strong enough. For this purpose we defined CCA security as follows:

**Definition 2** (CCA security)**.** An encryption $(\mathsf{E}, \mathsf{D})$ is said to be $(T, \epsilon)$-*CCA secure* if it's valid $(\mathsf{D}_k(\mathsf{E}_k(x)) = x)$ and for every $T$-time $A$ if we consider the following game:

- Sender and receiver choose shared $k \leftarrow_{\mathrm{R}} \{0,1\}^n$.
- $A$ gets access to black boxes for $\mathsf{E}_k(\cdot)$ and $\mathsf{D}_k(\cdot)$.
- $A$ chooses $x_1, x_2$.
- Sender chooses $i \leftarrow_{\mathrm{R}} \{1,2\}$ and gives $A$ $y = \mathsf{E}_k(x_i)$.
- $A$ gets more access to black boxes for $\mathsf{E}_k(\cdot)$ and $\mathsf{D}_k(\cdot)$ but is restricted not to ask $y$ to the decryption box. More formally, $A$ gets access to the following function $D'_k(\cdot)$ instead of $\mathsf{D}_k(\cdot)$

$$D'_k(y') = \begin{cases} D_k(y') & y' \neq y \\ \perp & y' = y \end{cases}$$

($\perp$ is a symbol that signifies "failure" or "invalid input")

- $A$ outputs $j \in \{1, 2\}$.

$A$ is successful if $j = i$, the scheme is $(T, \epsilon)$ secure if the probability that $A$ is successful is at most $\frac{1}{2} + \epsilon$.

**Order of Encryption and Authentication** A natural approach to get CCA security is to add authentication. There are three natural constructions:

- Encrypt and then Authenticate (EtA): Compute $y = \mathsf{E}_k(x)$ and $t_y = \mathsf{Sign}_{k'}(y)$ and send $(y, t_y)$. (IPSec-style)
- Authenticate and then Encrypt (AtE): Compute $t_x = \mathsf{Sign}_{k'}(x)$ and then $\mathsf{E}_k(t_x)$. (SSL style)
- Encrypt and Authenticate (E& A): Compute $y = \mathsf{E}_k(x)$ and $t_x = \mathsf{Sign}_{k'}(x)$ and send $(y, t_x)$. (SSH style)

(Don't encrypt is WEP-style) Note that in all these methods we use independent keys for encryption and authentication.

It turns out that generically there is only one right choice.

**Theorem 2.**

1. *If* $(\mathsf{E}, \mathsf{D})$ *is CPA-secure and* $(\mathsf{Sign}, \mathsf{Ver})$ *is CMA-secure then the the EtA protocol gives a CCA secure encryption scheme.*
2. *There is a CPA-secure encryption such that for every CMA-secure MAC the AtE protocol is not a CCA secure encryption scheme.*
3. *There is a CMA-secure MAC such that for every CPA-secure encryption, the AtE protocol is not a CCA secure encryption scheme.*

**Note:** This does not by itself mean that, say, SSL is not secure. But it does mean that it is not *generically secure*. That is, the SSL protocol relies on specific (and not explicitly stated) properties of the encryption scheme used.

This theorem and its proof can be found in Hugo Krawczyk's CRYPTO 2001 paper "The order of encryption and authentication for protecting communications (Or: how secure is SSL?)", see `http://eprint.iacr.org/2001/045`.

**Construction of a CCA secure scheme.** We'll now show how to construct a CCA-secure encryption scheme. That is, we prove the following theorem:

**Theorem 3.** *Assuming Axiom 1, there exists a CCA secure (private key) encryption scheme.*

*Proof.* The proof is actually to use the EtA construction, assuming some extra condition on the MAC (which is satisfied by the PRF-based construction). We say that a MAC has *unique signatures* if for every $x$ there's at most one tag $t$ such that $\mathsf{Ver}_k(x,t)=1$. This is equivalent to saying that $\mathsf{Ver}_k(x,t)$ outputs 1 if and only if $t=\mathsf{Sign}_k(x,t)$ (note that this is how $\mathsf{Ver}$ worked in the PRF-based construction). Let $(\mathsf{Sign},\mathsf{Ver})$ be such a MAC and let $(\mathsf{E}',\mathsf{D}')$ be a CPA-secure scheme. Our CCA-secure scheme $(\mathsf{E},\mathsf{D})$ will be the following:

- Key: $\langle k,k'\rangle$ with $k,k' \leftarrow_{\mathrm{R}} \{0,1\}^n$.
- Encryption: To encrypt $x$ compute $y=\mathsf{E}_k(x)$, $t=\mathsf{Sign}_{k'}(y)$ and send $\langle y,t\rangle$.
- Decryption: To decrypt $\langle y,t\rangle$ first verify that $\mathsf{Ver}_{k'}(y,t)=1$, otherwise abort (i.e., output $\perp$). If check passes, compute $\mathsf{D}_k(y)$.

**Security:** Suppose that $A$ is a $T$-time algorithm attacking the encryption scheme $(\mathsf{E},\mathsf{D})$. We'll convert $A$ to an algorithm $A'$ that breaks the CPA-secure scheme $(\mathsf{E},\mathsf{D})$.

First, we need to remember what does it mean to have a CPA attack against $(\mathsf{E}',\mathsf{D}')$. The algorithm $A'$ gets black-box access to $\mathsf{E}'_k(\cdot)$ but *not* to $\mathsf{D}'_k(\cdot)$. The algorithm $A'$ will do the following:

- Choose $k' \leftarrow_{\mathrm{R}} \{0,1\}^n$.
- Run $A$ in "its belly"
- Whenever $A$ asks for an encryption of $x$, pass the request to the encryption box $\mathsf{E}'_k$ to obtain $y=\mathsf{E}'_k(x)$, compute $t=\mathsf{Sign}_{k'}(y)$ and give $\langle y,t\rangle$ to $A'$. Also record this query in a table.
- If $A$ asks for a decryption of $\langle y,t\rangle$ which was previously returned to it as an encryption of $x$ then return $x$ to $A$.
- If $A$ asks for a decryption of $\langle y,t\rangle$ which was *not* previously returned to from the encryption oracle, then check if $\mathsf{Ver}_{k'}(y,t)=1$. If check fails then return $\perp$ to $A$. If check succeeds then abort the computation. In this case we say that $A'$ failed to simulate $A$.
- When $A$ sends the challenge $x_1,x_2$ pass it on to the sender to obtain $y=\mathsf{E}'_k(x_i)$ and give $\langle y,t\rangle$ to $A$, where $t=\mathsf{Sign}_{k'}(y)$.
- When $A$ outputs a guess $j$, output the same guess $j$.

We see that the only case that $A'$ fails to simulate the CCA attack of $A$ is when $A$ manages to produce a pair $\langle y,t\rangle$ such that

1. $\langle y,t\rangle$ was *not* obtained as a previous response to a query $x$ of the encryption oracle.
2. $\langle y,t\rangle$ is *not* the encryption of the challenge.
3. $Ver_{k'}(y,t)=1$

However, if $A$ does that then he breaks the MAC. Indeed, because of the unique signatures property of the MAC, Properties 1 and 2 imply that $y$ was not previously signed by the MAC, and hence it should not be possible for $A$ to find a $t$ such that $\mathsf{Ver}_{k'}(y,t)=1$. □

Note that the unique signatures property is indeed crucial. Suppose that the MAC had the property that it had an "extra unused bit". That is, the tag is of the form $t\circ b$ where $b$ is

a single bit, but verification only looks at the tag $t$. Thus, $\mathsf{Ver}_{k'}(y, t0) = 1$ if and only if $\mathsf{Ver}_{k'}(y, t1) = 1$.

In this case the encryption scheme will be clearly *not* CCA secure. (If the adversary gets the challenge $\langle y, t0\rangle$ it will give $\langle y, t1\rangle$ to the decryption oracle.) This sensitivity of CCA security to extra unused bits is one reason why some people feel CCA security is a bit too strong, but the research community has yet to find a clean definition that is still sufficient for all the applications of CCA security.