Lecture 9 - Message Authentication Codes

Boaz Barak

March 1, 2010

Reading: Boneh-Shoup chapter 6, Sections 9.1–9.3.

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.

Another important example is over-the-air software patches — you want to make sure that the software patch you are installing is the right one from the software company and not by some hacker, but there's nothing secret about the patch.

In general, integrity is more basic than secrecy, in the sense there are many situations where one cares about integrity and not secrecy, but not so many of the reverse. (In fact, as we saw last week, without integrity it's often possible to violate secrecy as well.)

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 $x_1, \ldots, x_{n-1}, \overline{x_n}$.

More generally, while encryption is supposed to be the digital analog of a sealed envelope, that provides both secrecy and integrity, one should not get confused by this metaphore. (Indeed, the closes thing to a digital analog of a sealed envelope is a CCA secure encryption, that provides some measure of integrity as well.)

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

[1]Sometimes this is generalized to more bits, say, parity mod 2^{32}.

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:

- 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 $\Pr[t = t'] = 2^{-n}$. □

Note that this MAC has the property that both signing and verification are *deterministic*, and moreover for every message x there is a *unique tag* that the verification accept. We call this the *unique tag* property— most of if not all MACs we'll consider have this technical property and it's sometime useful.

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 (E, D) is said to be (T, ϵ)-*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, ϵ) 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)

(Use only CRC for authentication 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* (E, D) *is CPA-secure and* $(\mathsf{Sign}, \mathsf{Ver})$ *is CMA-secure with unique tags property 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 (with unique tags) such that for every CPA-secure encryption, the A& E protocol is not even a CPA 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`. We now sketch the proof:

Item 1: EtA is CCA secure This is basically the proof we saw last time, where we used a PRF to convert a CPA secure encryption into a CCA secure encryption. By examining the proof, one can see that all we really used is the fact that a PRF is a MAC to ensure that the decryption box is useless for the adversary. (We also used the unique-tags property, but EtA will give a meaningful notion close to CCA security, namely authenticated encryption, even if the MAC doesn't have the unique-tags property.)

Item 3: E&A is not generally secure The idea is that a MAC does not have to preserve secrecy of the message.

Item 2: AtE is not generically secure We'll use "Sushant's cryptosystem". Take any CPA secure encryption (E, D) for one bit messages. Then you can construct from it a CPA-secure encryption for m bit messages by letting $E'_k(x_1 \cdots x_n) = E_k(x_1)E_k(x_2)\cdots E_k(x_n)$ (exercise). Now we can assume that the input is encoded so that every string ends with "0". This means that given an encryption $E'(x)$, we can by replacing the i^{th} block with a copy of the $m+1^{th}$ block convert it to an encryption of $x_1 \cdots x_{i-1}0x_{i+1}\cdots x_m$. (We can also assume the string ends with 01 and so also change the i^{th} bit to 1, moreover, for this proof the attacker can choose to use messages that only end with 0 or 01 so we don't even need this assumption.)

Now suppose we use E' in the AtE setting and so we get an encryption of the form $E'(x, \mathsf{Sign}(x))$. If we have access to a decryption box, we change the i^{th} bit of x to 0, and see if the MAC still passes verification. If it does, then we know that the original bit was 0, otherwise we know that it was 1. This allows to launch a successful CCA attack.

(In fact, there was a successful attack against SSL of similar nature, using knowledge of whether or not the Mac failed.)

Input length extension We showed how to construct a PRF from every pseudorandom generator, but Practical constructions of PRFs come from block ciphers or similar functions, that have a fixed and small block size, say 128 bits. On the other hand, the messages we want to sign — say programs — are often very large (megabytes or even gigabytes). So, given a pseudorandom function $f_k : \{0,1\}^n \to \{0,1\}^n$, (e.g. with $n = 128$) we'd want to transform it into a PRF $g_k : \{0,1\}^* \to \{0,1\}^n$ that can take as inputs strings of arbitrary length. Some desired properties for such a transformation are:

1. Security: obviously we want $\{g_k\}$ to be PRF if $\{f_k\}$ was. In fact for practical applications we'd want as tight as possible reduction relating the security of $\{g_k\}$ to the security of $\{f_k\}$.
2. Efficiency: ideally the transformation should be very efficient. One goal is to minimize the number of invocations of f_k. For starters, we might want to ensure that we use roughly $|x|/n$ invocations to evaluate g_k on x. (In fact, you might even be able to get away with *one* invocation, as we'll see next week.)

3. Secret key length: you want the secret key of g to be not much longer than the secret key of f
4. Streaming: for very long messages, you often get them one block at a time, and you might not even know the length of the message until you are done. So you should be able to compute $g_k(x)$ even if you get only streaming access to x.
5. Parallelism: sometimes you might want to use hardware parallelism to compute the MAC on a large message, so you want to be able to ensure that if you have ℓ CPU's working on the MAC on x, you can actually compute it ℓ times faster.
6. Incremental: sometimes you might want the property that if you already computed a MAC t of a long message x, and then you modify only one block of x to get x', you don't have to do a long computation to compute the MAC on x'.

(There could be other requirements depending on the application.)

Achieving input length extension There is a general approach of converting a PRF $f_k : \{0,1\}^n \to \{0,1\}^n$ into a PRF mapping $\{0,1\}^*$ to $\{0,1\}^n$ in two stages:

1. Blockwise function: first transform $\{f_k\}$ into a PRF that works only on inputs of that are integer multiple of n.
2. General function: use padding to get rid of the requirement that the input length is an integer multiple of n.

The Boneh-Shoup book describes several different approaches used to achieve these goals. We'll focus on one elegant solution that combines two steps: PMAC to solve the first (and main) step, and CMAC to solve the second step.

We remark that often there is an intermediate step, in which one construct a function that is blockwise and also *prefix free*. That is, security is only guaranteed if the adversary never makes two queries such that one is a prefix of the other. The CMAC trick can be used to get rid of that condition as well.

PMAC Here's the PMAC construction (we'll actually work with a simplified variant very close to PMAC$_0$ described in Boneh-Shoup, but PMAC is basically just an optimized variant of PMAC$_0$). For simplicity we think that f_k maps $\mathbb{Z}_p = \{0, \ldots, p-1\}$ to $\mathbb{Z}_p$, where p is some prime. (We can think of p as being very close to 2^n, say $2^n - n$, and so we can embed $\mathbb{Z}_p$ in $\{0,1\}^n$ without much loss in efficiency.[2]) For $x = x_1, \ldots, x_\ell$ where $x_i \in \mathbb{Z}_p$, we define

$$g_{k,k',r} = f_{k'}(\sum_{i=1}^{\ell} f_k(x_i + ir))$$

where k, k' are random keys for the PRF f_k and r is random in $\mathbb{Z}_p$.

We remark that all we'll use about $\mathbb{Z}_p$ is that it's a field that has multiplication and addition, and everything works the same in the finite field $GF(2^n)$ of 2^n elements. The latter field is more convenient for current computer architecture, and in fact in that field addition corresponds to XOR. The main difference between PMAC and PMAC$_0$ is that PMAC uses $GF(2^n)$

[2]One subtle point is how do we ensure the *output* of the PRF is in $\mathbb{Z}_p$, but since in a random function the probability that a random n bit string, interpreted as a number, is larger than $2^n - n$ is negligible, we can assume this almost never happens for the PRF as well and treat this case arbitrarily.

Theorem 3. *If f_k was a PRF then $g_{k,k,r}$ is a PRF.*

The heart of the proof is obtained by showing that the adversary has only negligible probability to succeed in finding two inputs $x = x_1 \cdots x_\ell$ and $x' = x'_1 \cdots x'_{\ell'}$ such that

$$\sum_{i=1}^{\ell} f_k(x_i + ir) = \sum_{i=1}^{\ell'} f_k(x'_i + ir)$$

CMAC To get a bitwise MAC we need to use some padding to pad up the message to an integer multiple of n. The simplest padding is just to pad a message that is not of length an integer multiple of n with zeroes but although this is sometimes used, this is insecure (can you see why?). A secure padding is to add to every message a bit 1 at the end, and then pad it with zeroes. But this means that if you have a message that is exactly one block length, you'll need two blocks to encode it— a 100% overhead. The CMAC is a clever trick to get rid of this problem by using the following *randomized* padding scheme (this is again a simplified variant of CMAC):

If x is of length an integer multiple of n then do nothing. Otherwise, add 1 to x, pad it with zeroes, and xor a random secret $r \in \{0,1\}^n$ to the last block. One can show that the probability that an adversary finds two messages whose padding is the same is negligible.