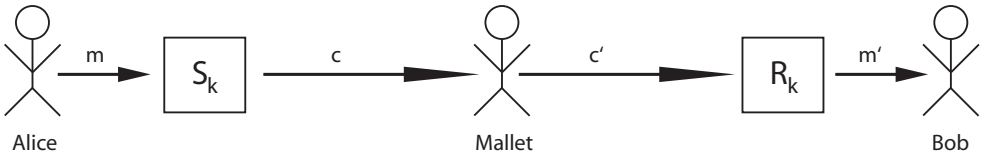
Lecture 9

Message Authentication

So far, we have only considered encryption of messages, ensuring that the information in a message remains secret from eavesdroppers. Instead of or in addition to encryption, we may care that the receiver of a message knows that the message came from a trusted sender, that he or she can verify the authenticity of the message. The standard scenario we are considering is of Alice sending messages to Bob through some channel, in which an adversary Mallet can interrupt the flow of messages, intercepting messages that Alice is sending, and either forwarding them on to Bob as is, altering them in some way, creating new messages, or severing the channel and not delivering messages at all. The picture for a single message is below. Alice puts her message m through some keyed function S_k intended to facilitate the preservation of message integrity, and this processed message c is sent over the channel, where Mallet may or may not tamper with it. The (possibly modified) message Bob receives is c', and Bob pases this through a verification function R_k to produce either a plaintext m' or an error message $\perp$ indicating tampering. We still refer to c and c' as "ciphertexts", even though they are not necessarily encrypted.



If Alice is sending a sequence of messages, we would of course have

$$m_1, m_2, \ldots \to c_1, c_2, \ldots \to c'_1, c'_2, \ldots \to m'_1, m'_2, \ldots$$

The are two notions of message integrity that we'll discuss.

Connection Semantics

Here we are interested in having an overall secure channel that is for the most part tamper-proof, short of severing the entire channel.

(1) Correctness: We would like Bob to obtain after verification the same messages that Alice sent, and in the same order. That is, we would like

$$(m_1, m_2, \ldots, m_n) = (m'_1, m'_2, \ldots, m'_n)$$

(2) Security: We would like it if Mallet is unable to forge messages, so that her best bet is simply to sever the channel so that no further messages can be transmitted. In other words, we would like the received messages $(m'_1, m'_2, \ldots)$ to be a prefix of the sent messages $(m_1, m_2, \ldots, m_n)$.

A more realistic, practical notion is the following:

Packet Semantics

(1) Correctness: We will let the verification method R be stateless and deterministic, though S can be stateful and randomized. We would then like

$$R_k(S_k(m)) = m, \forall m, k.$$

(2) Security of Plaintext Integrity: We would like Bob to only obtain actual messages that Alice sent, or perhaps error messages indicating tampering. In other words, we would like the received messages to be a subset of the sent messages:

$$\{m'_1, m'_2, \ldots\} \subseteq \{m_1, m_2, \ldots, \perp\}$$

This particular notion of security is known as INT-PTXT (for "integrity of plaintext").

(2′) Security of Ciphertext Integrity: As an alternative notion of security, we would like the verification method to throw an error flag on any ciphertext that didn't originate from Alice, so that Mallet is unable to forge a ciphertext of her own that Bob is tricked into accepting. Specifically, we would like

$$\{c'_i : R_k(c'_i) \neq \perp\} \subseteq \{c_1, c_2, \ldots\}$$

Similarly, this notion of security is known as INT-CTXT.

Formalization of Security Notions

We naturally would like to formalize these security notions in terms of adversary advantage so that we can determine computational security bounds. For the plaintext notion, we will define

$$\mathrm{Adv}^{\text{int-ptxt}} A = \Pr[k \xleftarrow{\$} K; A^{S_k, R_k} \text{ forges}]$$

Here we say that A forges if A queries R_k and gets a response $m \neq \perp$ that wasn't previously queried to S_k. In other words, suppose $\{(x_1, y_1), (x_2, y_2), \ldots\}$ are queries to S_k (meaning that y_i is the result of applying S_k to plaintext x_i), and $\{(y'_1, x'_1), (y'_2, x'_2), \ldots\}$ are queries to R_k (meaning that x'_i is the result of applying R_k to ciphertext y'_i). Then A forges if $\exists i$ with $x'_i \neq \perp$ and $\forall j\ x'_i \neq x_j$.

We say that (S, R) is (t, q_s, q_r, ϵ)-INT-PTXT secure if $\mathrm{Adv}^{\text{int-ptxt}} A \leq \epsilon$, for all adversaries A running in time less than t, making at most q_s queries to S and at most q_r queries to R. We can similarly construct a formal notion of security for INT-CTXT by changing what it means to forge: using the same sequence of queries to S_k and R_k above, we say that A forges if $\exists i$ with $x'_i \neq \perp$ and $\forall j\ y'_i \neq y_j$.

Observation: INT-CTXT $\implies$ INT-PTXT
This is trivial to show, for if a scheme can detect inauthentic ciphertexts, there's no way to possibly transmit to Bob a forged message. The converse, however, is not true. The adversary can change a ciphertext in such a way that the original message can still be intact. For example, if S simply appends a 0 bit to the front of messages, and R just throws away that first bit, the adversary can flip the first bit, and the message will still be intact.

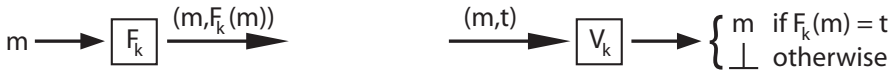
Encryption Does Not Provide Integrity

We might think encrypting our messages with a secure encryption scheme will take care of the whole authentication problem. Unfortunately, it does not. In fact, we can demonstrate this with the one-time pad–even though it is an extremely secure encryption method, it does nothing for authentication.

If the adversary intercepts c and passes $c \oplus \ldots 01$ as c', then we already haven't preserved INT-PTXT, because when Bob decrypts by doing $c' \oplus k$, the message he recovers has a flipped bit. Bob thus has no way to know whether to trust the ciphertext he receives. We could try to patch up this scheme, say using a CRC checksum. Using CRC is very bad, however, since it's a linear function, and is easy to fool. We can, however, try a related but better idea.

MACs

We could design our function F that given a plaintext message m, returns a tag $F_k(m)$ that we can append to our message. Then Alice sends $(m, F_k(m))$ over the channel. Bob receives (m, t) where the tag t may or may not be the original tag. The method on the receiving end should return the message m if the tag is authentic, that is, if $t = F_k(m)$, or return the error $\perp$ otherwise. We can define the receiving method in terms of a verification prodedure V that returns 1 if $t = F_k(m)$ and 0 otherwise. Such a scheme is called a **Message Authentication Code**, or **MAC**. The picture of this scheme is below.



We can choose to define security notions for a MAC if the scheme satisifies INT-PTXT or INT-CTXT. For the INT-PTXT notion, we can define the advantage of an adversary against the MAC as

$$\text{Adv}^{\text{MAC}} A = \Pr[A^{F_k, V_k} \text{ forges}]$$

where we say that A forges if A makes a query (m, t) such that $V_k(m, t) = 1$ and m was not previously queried to F_k. In other words, if the adversary can create a valid tag for his or her own message based on learning information from queries to oracles for F_k and V_k. We say that a scheme is a (t, q_s, q_v, ϵ)-MAC if $\text{Adv}A \leq \epsilon$ for all adversaries A running in time less than t and making at most q_s queries to F_k and at most q_v queries to V_k.

We define INT-CTXT security for a MAC in exactly the same way, except that we change the notion of forgery. Under this notion, we say that A forges if it makes a query (m, t) such that $V_k(m, t) = 1$ and there was no prior query with input m and output t. That is, this is a more difficult notion of forgery than before, since passing an authentic message with a new tag will be detected.

Do Such Schemes Exist?

These notions of security are all very nice, but they don't do us much good if we can't construct a scheme in practice that meets these requirements. However, we can in fact do so, and we've already seen all the tools we need.

Let F_k be a random function R that is known only to Alice and Bob. Say

$$R : \{0,1\}^* \to \{0,1\}^{80}.$$

Suppose that Alice is sending a sequence of messagse $m_1, m_2, \ldots$ with tags $R(m_1), R(m_2), \ldots$ appended to them. Mallet attempts to pass off a forgery (m', t'). What is the probability that she succeeds? It's the probability that her chosen tag t' is the appropriate tag for her message m', and the only information she has access to is that each message m_i that Alice sent corresponds to a tag $R(m_i)$. But R is a random function, so this is no help at all.

$$\Pr[R(m') = t' | R(m_i) = t_i] = \Pr[\text{Mallet forges}] = \frac{1}{2^{80}} \text{ (if } m' \text{ is new, i.e. } m' \neq m_1, m_2, \ldots)$$

So R provides us with a $(\infty, q_s, 1, \frac{1}{2^{80}})$-MAC. Here we only allowed Mallet one attempt at pushing one of her messages through V, which won't work in practice, but this is not an issue, as the following lemma shows.

Lemma. *If F is a $(t, q_s, 1, \epsilon)$-MAC, then it is a $(t, q_s, q_v, q_v\epsilon)$-MAC.*

Proof. Let A make q_v verification queries (to V_k). Because we're using a random function R, whether A is able to forge on a given attempt, if it hasn't forged already, is a completely independent event of the other attempts, and thus the probability of it happening is uniform.

$$\begin{aligned}\Pr[\text{A forges}] &\leq \sum_{i=1}^{q_v} \Pr[\text{A forges on } i\text{th } V_k \text{ query} \mid \text{A didn't forge on first } i-1 \text{ queries}] \\ &\leq \sum_{i=1}^{q_v} \epsilon = q_v\epsilon\end{aligned}$$

□

So using a random function R as F_k gives us a $(\infty, q_s, q_v, \frac{q_v}{2^{80}})$-MAC. Of course, in practice we can't use a real random function because of the difficulty of specifying it. Instead, we simply use a PRF as before.

Theorem. *If F is a $(t, q_s + q_v, \epsilon)$-PRF, then using F gives a $(t, q_s, q_v, \epsilon + \frac{q_v}{2^{80}})$-MAC.*

Proof. Straightforward, so omitted here, but can be found in the Bellare-Rogaway notes. □

Suppose we want to have $(\infty, 1, 1, \frac{1}{2^{80}})$ security. For even greater simplicity, suppose that the message we are trying to send is just a single bit. So let $x, x' \in \{0,1\}$ and $F : \text{Keys} \times \{0,1\} \to \text{Tags}$.

$$\boxed{\text{Alice}} \xrightarrow{x, y=F_k(x)} \boxed{\text{Adversary}} \xrightarrow{x', y' \text{ (where } x' \neq x)} \boxed{\text{Bob}}$$

A random function is now very easy to specify, since there are only two possible inputs. So say $R : \{0,1\} \to \{0,1\}^{80}$. We can specify R as follows:

$$\begin{aligned} 0 &\to -K_0- \\ 1 &\to -K_1- \end{aligned}$$

Now we can pass our message bit b through our sending function S which uses R to create the appropriate tag K_b.

$$b \to \boxed{S} \xrightarrow{b, K_b}$$

The attacker doesn't get to see the result of the other bit getting passed through S, so the attacker can only guess randomly what the other K_b tag is. Thus we have a $(\infty, 1, 1, \frac{1}{2^{80}})$-MAC. From the above theorem, this also gives a $(\infty, 1, q_v, \frac{q_v}{2^{80}})$-MAC.