U.C. Berkeley — CS276: Cryptography
Professor David Wagner
Scribe: Arindam Chakrabarti (arindam@cs)

Lecture 9
February 23, 2004
Last revised March 2, 2004

Lecture 9

9.1 Message authenticity review

Till the last lecture our focus in this course has been to study techniques that achieve *message secrecy*, e.g. techniques that allow Alice to send a cipher-text c representing a message m over an open channel to Bob without letting a malicious adversary Eve learn much about m by eavesdropping on the channel and learning c; and yet allowing Bob to correctly decode c to get back m.

In the last lecture we started talking about *message authenticity*[1], which is concerned not with keeping messages from Alice to Bob secret but with preventing messages sent over an open channel from being tampered with by a malicious adversary: guaranteeing that when Bob receives a message he knows whether it really came unmodified from Alice or was created/modified by a malicious adversary Mallet reading and writing signals passing through the open network.

We motivated the need for authenticity:

- If I send a message to my bank asking it to pay Amazon.com $ 100.00 for a book, I would not like to worry about somebody changing the amount to $ 1000.00.
- A computing system with sophisticated well-designed and well-implemented security systems would be useless against an adversary that can send commands to the system that appear to come from the system administrator.

At this point one might worry about the following kinds of attacks:

- A malicious adversary Mallet simply drops all messages being sent by Alice to Bob. These are called *denial of service* attacks, and while they are an important issue in computer and network security, they are clearly not addressable using cryptography. Hence, in this course we will not consider such attacks.
- Alice sends a message to her bank (Bob) asking it to pay $ 100.00 to the malicious adversary Mallet. Mallet sends this message to the bank over and over again, hoping that the bank will transfer $ 100.00 out of Alice's account and into Mallet's account each time. These are called *replay* attacks, and we will not consider them in this course. It is not hard to see that attacks of this kind can be easily prevented by introducing a sequence number into each message sent over the network, and accepting each valid message received only if it has a new sequence number.

[1] a.k.a. *message integrity*

Let $\mathcal{M}$ be a set of messages, and $\mathcal{C}$ be a set of cipher-texts. Let $S_k : \mathcal{M} \to \mathcal{C}$ be used by Alice to encode messages she wants to send to Bob. Let $R_k : \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ be used by Bob to check authenticity of received transmissions. We defined the advantage of an adversary A in the integrity of plain-text sense to be:

$$\mathrm{Adv}^{\text{INT-PTXT}} A = \Pr_k[A^{S_k, R_k} \text{ forges}]$$

where adversary A has oracle access to S_k and R_k, and A is deemed to have succeeded in forging when it is able to find a cipher-text y such that $R_k(y)$ is a valid message that A never used to query S_k.

We provided a quantitative notion of authenticity in the integrity of plain-text sense as follows: a message authentication scheme (S, R) is (t, q_t, q_v, ϵ)-INT-PTXT-secure if

$$\mathrm{Adv}^{INT-PTXT} A \leq \epsilon$$

for all adversaries A running in time $\leq t$, making $\leq q_t$ queries to S_k, and $\leq q_v$ queries to R_k.

Similarly, we defined the advantage of an attacker A in the integrity of cipher-text sense to be:

$$\mathrm{Adv}^{\text{INT-CTXT}} A = \Pr_k[A^{S_k, R_k} \text{ forges}]$$

where adversary A has oracle access to S and R, and A is deemed to have succeeded in forging if it is able to find a cipher-text y that it never obtained by querying S_k, and such that $R_k(y) \neq \perp$.

We provided a quantitative notion of authenticity in the integrity of cipher-text sense as follows: a message authentication scheme (S, R) is (t, q_t, q_v, ϵ)-INT-CTXT-secure if

$$\mathrm{Adv}^{INT-CTXT} A \leq \epsilon$$

for all adversaries A running in time $\leq t$, making $\leq q_t$ queries to S_k, and $\leq q_v$ queries to R_k.

We observed that integrity in the INT-CTXT sense is stronger than integrity in the INT-PTXT sense. Finally, we observed that previously studied encryption techniques that provide *message secrecy* do not provide authenticity. How can we achieve authenticity ?

9.2 Achieving message authenticity

Let Alice attach a cryptographic checksum to each message that she sends to Bob. Should this checksum be computed by a secret algorithm ? Kerckhoff's Law tells us that "only secrecy of the key provides security".[2] Hence this cryptographic checksum must be computed by an algorithm F_k using a secret key k shared between Alice and Bob. Figure 1 shows the scenario. On receiving a pair (m, T), Bob checks if $F_k(m) = T$. If yes, Bob accepts the message m, else he rejects it. In the following discussion for notational convenience we will use the function $V_k(m, T)$ which evaluates to 1 if $F_k(m) = T$ and 0 otherwise.

[2] Alternatively known as Shannon's Maxim: "the enemy knows the system".

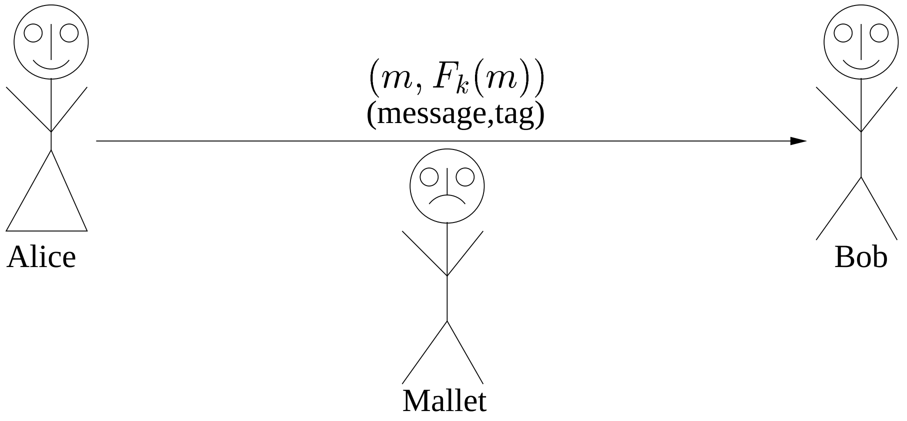

Figure 1: Alice trying to send messages to Bob while Mallet listens in and tampers with the messages

What are the conditions that this scheme must satisfy in order to be a good message authentication scheme ?

Clearly, seeing a number of pairs $(m_i, F_k(m_i))$ should not let Mallet find out the key k. How do we formalize this idea ?

DEFINITION 1 **(Adv^{MAC}A)** *Given a MAC (F_k, V_k), the advantage of an attacker A is defined to be $Adv^{MAC}A = \Pr_k[A^{F_k,V_k}$ forges$]$, where A is deemed to have succeeded in forging if it can find a pair (m, T) such that $V_k(m, T) = 1$ and m was never used by A as an argument to F_k.*

DEFINITION 2 **Security of a MAC** *(F_k, V_k) is a (t, q_t, q_v, ϵ)-secure MAC if $Adv^{MAC}A \leq \epsilon$ for all adversaries A running in time $\leq t$, making $\leq q_t$ queries to F_k, and $\leq q_t$ queries to V_k.*

QUESTION 1 *In Definition 2 why do we allow the attacker to have oracle access to both F_k and V_k ?*

ANSWER 1 *It is easy to see that the attacker must be given access to the verification oracle V_k. Think of the situation where Alice is sending (tagged) messages to Bob over a network on which Mallet can read, modify and write transmissions. Clearly Mallet can send (m, T) pairs to Bob and observe whether they were accepted. The pair (m, T) will be accepted by Bob if and only if $V_k(m, T) = 1$. Thus Mallet can make queries to V_k.*

The other point still remains to be explained: why should Mallet be allowed to make queries to Alice's tag-creation box F_k ?

The answer is not hard to see. Let Alice actually be not a human, but an email server on which Bob's email is stored. Mallet can certainly know Bob's email address, and can send *him emails with contents he can choose as he likes. When Bob accesses his email, the mail server Alice will tag the mail and send it off over the network. Thus Mallet gets to know the tag for any arbitrary message he chooses. Hence, any notion of security that does not consider the situation that Mallet has access to F_k is unrealistic and hence useless.*

Note that the Definition 1 uses forgery in the INT-PTXT sense. We can get an analogous definition of the advantage of an attacker against a MAC in the INT-CTXT sense as follows. This leads to the definition of a Strong-MAC.

DEFINITION 3 (**$\mathbf{Adv}^{Strong-MAC} A$**) *Given a pair (F_k, V_k), the advantage of an attacker A is defined to be $Adv^{Strong-MAC} A = \Pr_k[A^{F_k,V_k}$ forges$]$, where A is deemed to have succeeded in forging if it can find a pair (m, T) such that $V_k(m, T) = 1$ and T was never obtained by A by computing $F_k(m)$.*

DEFINITION 4 **Security of a Strong-MAC** *(F_k, V_k) is a (t, q_t, q_v, ϵ)-secure Strong-MAC if $Adv^{Strong-MAC} A \leq \epsilon$ for all adversaries A running in time $\leq t$, making $\leq q_t$ queries to F_k, and $\leq q_t$ queries to V_k.*

NOTE 1 *Note that Strong-MAC is a stronger concept than MAC. In other words, a message authentication scheme that is a (t, q_t, q_v, ϵ)-secure MAC may not be even a $(t, 1, 1, 1)$-secure Strong-MAC. For example, let Alice use a MAC (F_k, V_k) such that for a message m, $T = F_k(m)$ contains a redundant bit (at a position known to Mallet, say at the end with no loss of generality) that Bob's verification algorithm V_k does not look at. It is easy to see that Mallet can easily break this scheme in the INT-CTXT sense with probability 1 by constructing an arbitrary message m, computing $T = F_k(m)$ (this is where it needs to use the one query to F_k it is allowed) and then constructing the tag T' that is identical to T except for the last bit. Now (m, T') is a new pair because T' was not obtained by querying F_k with argument m; and hence Mallet has succeeded in forging in the INT-CTXT sense.*

At this point it is natural to ask why we allow the attacker to make several (q_v) verification queries; i.e. several attempts to forge. It is natural that Mallet should manage to make several queries to Alice's F_k to try to see what the tag-creation process does. But what if we insisted on allowing Mallet to make only one final attempt to forge a message as coming from Alice ? In other words, would it make any difference if we insist that q_v should always be 1 ? The following Lemma shows that it makes no difference.

LEMMA 1
If (F_k, V_k) is a $(t, q_t, 1, \epsilon)$-secure MAC, then (F_k, V_k) is a $(t, q_t, q_v, q_v \cdot \epsilon)$-secure MAC.

PROOF: Let A be an attacker trying to break a $(t, q_t, 1, \epsilon$-secure MAC (F_k, V_k) making q_v queries to the verification oracle V_k. Clearly, A will succeed only if at least one of the queries to V_k returns 1. Thus:

$$\begin{aligned}
\text{Adv}^{MAC} A \text{ making } q_v \text{ queries} &\leq \sum_{i=1}^{q_v} \Pr_k[A^{F_k,V_k} \text{ forges on } i^{th} \text{ query to } V_k] \\
&\leq \sum_{i=1}^{q_v} \epsilon \\
&\leq q_v \cdot \epsilon
\end{aligned}$$

Hence the required result follows. □

As the above Lemma shows, allowing the attacker $q_v > 1$ queries increases his chance of forging by only a factor q_v. What if an attacker Mallet makes a large number of queries and improves his chance of cracking the system ? Can Alice and Bob prevent q_v from getting too large ?

Note that when the attacker makes queries to F_k, Alice and Bob may not get to know about it[3], but whenever Mallet makes unsuccessful queries to V_k, Bob immediately sees a message with a bad tag. Thus, q_v, the number of queries made to V_k can be controlled by Alice and Bob. Indeed, many systems shut down the communication session if too many unauthenticated messages are received.

9.3 Constructing a secure MAC

How do we construct a secure MAC ? Recall the One Time Pad (OTP) which absolutely guaranteed secrecy of a communicated message without any assumptions. Can we get an analogous scheme to absolutely guarantee authenticity of a communicated message ? Say Alice wants to send a single message m to Bob. How can Bob be sure that it has not been tampered with by Mallet ?

Does the One Time Pad idea help ? What if Alice sends the pair $(m, m \oplus k)$ over the network ? Wait a minute... this clearly does not work: Mallet can just intercept the pair $(m, m \oplus k)$, recover the "secret" key k using $k = (m) \oplus (m \oplus k)$, and then use it to "authenticate" any other arbitrary message m' by sending on the pair $(m'm' \oplus k)$ to Bob.

Maybe the scheme does not work because we are sending information about m twice somehow: what if Alice sends only $m \oplus k$? Hmmm... this does not work either: Mallet can intercept the transmitted bit sequence, flip any bits he likes, and send it on to Bob, who will get a message quite different from what Alice had sent and have no reason to suspect it has been tampered with in any manner.

Will it help if Alice uses two independent keys ? What if Alice sends the pair $(m \oplus k, m \oplus k')$? No... this fails for the same reason as the attempt before: all Mallet has to do is to take care to flip the bits in both parts of the transmitted pair in a consistent manner, and Alice an Bob are back to square one.

Let us simplify the situation to be able to analyze the requirements better and see a solution more clearly. Say Alice just has to send a single bit message b to Bob. Now we seem to be getting somewhere: the information that has to be sent across is the choice: first or second, of a pair of two elements. Say Alice and Bob share a secret pair (k_0, k_1). Alice sends the pair (b, k_b) to Bob. On receiving a pair (b, T), Bob checks if $T = k_b$; if it is he accepts the

[3]Consider the situation where Mallet sends email to Bob's email server Alice and observes the tags created by Alice when the contents of the emails are sent to Bob when Bob accesses his email over the network. If Mallet can pretend to be a legitimate email-sender, Alice and Bob will never know that queries are being made to Alice's F_k for some nefarious purpose.

message b, else he rejects it. It is easy to see that if Mallet changes the transmitted pair in any way, unless he gets really lucky and manages to flip k_0 into k_1 or vice versa, Bob will reject his change as an unsuccessful attempt at forgery.

NOTE 2 *It is easy to see that Alice can simply send k_b; b does not need to be sent along at all.*

The above scheme has an obvious problem: for sending a single bit b Alice is forced to also send a (relatively large to minimize probability of forgery by Mallet) tag k_b. Also, each pair k_0, k_1) is good for using only once, and Alice and Bob have to *a priori* agree on a large number of such pairs which must be kept secret. Can we do better than this ?

9.3.1 Pairwise Hashing

Let H be a function $H : \mathcal{K} \times \mathcal{M} \to \{0,1\}^{80}$, where $\mathcal{K}$ is the set of keys and $\mathcal{M}$ is the set of messages.

DEFINITION 5 $H : \mathcal{K} \times \mathcal{M} \to \{0,1\}^{80}$ *is* strongly 2-universal *if:*

$$\Pr_k[H_k(x) = y \textit{ and } H_k(x') = y'] = \frac{1}{2^{160}} \quad \forall x, x' \in \mathcal{M}(x \neq x'), y, y' \in \{0,1\}^{80}.$$

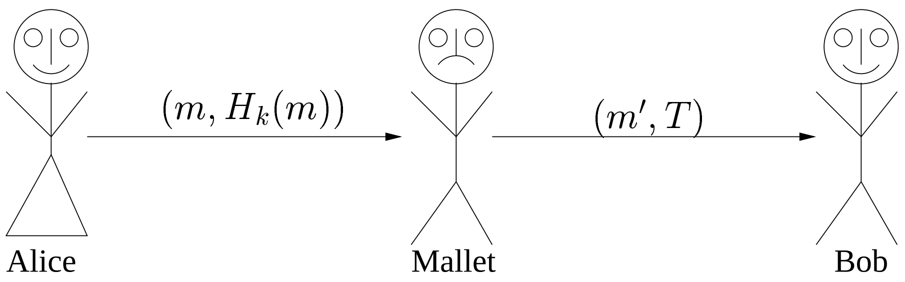

Figure 2: Alice uses pairwise hashing to prevent Mallet from tampering with her messages to Bob

Let Alice tag the message m with $H_k(m)$ and send the pair $(m, H_k(m))$, as shown in Figure 2. On receiving a pair (m', T), Bob needs to check if $H_k(m') = T$; if it is Bob accepts the message m', and rejects it otherwise. The following Theorem tells us that this is a good MAC.

THEOREM 2
If $H : \mathcal{K} \times \mathcal{M} \to \{0,1\}^{80}$ is strongly 2-universal, *then H is a $(\infty, 1, q_v, \frac{q_v}{2^{80}})$-secure MAC.*

Figure 3 shows the situation. Alice sends a pair $(x, H_k(x))$ over the network. Mallet intercepts it, and sends a pair (x', y') $(x \neq x')$ over to Bob. If y' happens to match $H_k(x')$, then Bob will accept the message x' as having been sent by Alice, and Mallet will have succeeded in forging a new plain-text message; thus breaking the MAC (in the INT-PTXT sense). The above Theorem says that if Mallet sends q_v such pairs (x', y') to Bob, he will succeed in forging at least once with probability only at most $\frac{q_v}{2^{80}}$. In order to prove the above Theorem, let us state and prove the following Lemma.

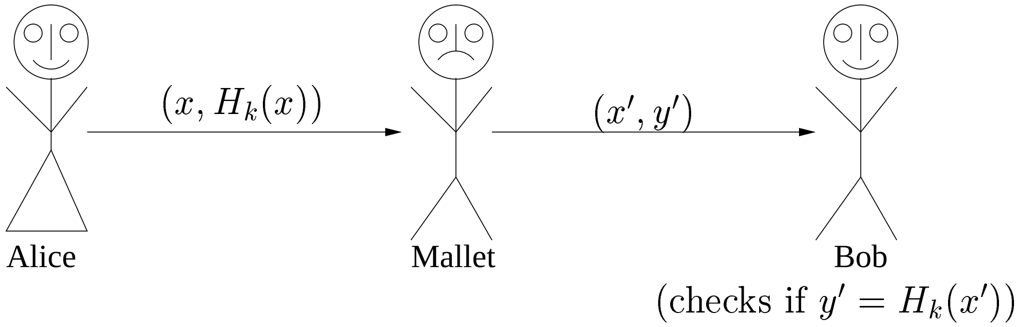

Figure 3: Mallet tries to forge Alice's message to Bob

LEMMA 3
If $H : \mathcal{K} \times \mathcal{M} \to \{0,1\}^{80}$ is strongly 2-universal, then

$$\Pr_k[H_k(x') = y' \mid H_k(x) = y] = \frac{1}{2^{80}} \quad \forall x, x' \in \mathcal{M}(x \neq x'), \forall y, y' \in \{0,1\}^{80}.$$

PROOF:**(For Lemma 3)** Consider arbitrary $x, x' \in \mathcal{M}$, $x \neq x'$; and arbitrary $y, y' \in \{0,1\}^{80}$.

$$\begin{aligned} \Pr_k[H_k(x') = y' \mid H_k(x) = y] &= \Pr_k[H_k(x') = y'] \quad \text{(independence; } x \neq x'\text{)} \\ &= \tfrac{1}{2^{80}} \end{aligned}$$

□

PROOF:**(For Theorem 2)** The probability that an attacker A forges a new message using just one query each to the encoding oracle H_k and the verification oracle is given by:

$$\begin{aligned} \Pr_k[A \text{ forges}] &\leq \textstyle\sum_{x,x',y'} \Pr_k[H_k(x') = y' \mid H_k(x) = y] \cdot \Pr[x, x', y'] \\ &\leq \textstyle\sum_{x,x',y'} \frac{1}{2^{80}} \cdot \Pr[x, x', y'], \qquad \text{by Lemma 3} \\ &\leq \tfrac{1}{2^{80}}, \qquad \text{since } \textstyle\sum_{x,x',y'} \Pr[x, x', y'] = 1 \end{aligned}$$

Clearly, we put no bound on the running time t available to the attacker A. However, we allowed the attacker only one query to the encoding oracle H_k. Thus, from the above probability we conclude that H is an $(\infty, 1, 1, \frac{1}{2^{80}})$-secure MAC. Now, from Lemma 1 it is easy to conclude that H is an $(\infty, 1, q_v, \frac{q_v}{2^{80}})$-secure MAC. □

How hard is it to find 2-universal hashes ? It turns out that it is easy. It can be proved that for the set of messages $\mathcal{M} = \{0,1\}^a$, and the set of tags $\mathcal{T} = \{0,1\}^b$, the hash function $H_{M,B} = M \cdot x \oplus B$ where M is an $a \times b$ binary matrix, $x \in \mathcal{M}$, and b is a $b \times 1$ binary matrix, is 2-universal. There are more efficient ones as well.

9.4 Closing comments

Thus, we have seen how to get unconditional message authentication; making no assumptions about amount of computational resources available to the attacker, or the inequality of certain complexity classes like P and NP, as we often have to do in this and many other areas of theoretical computer science.

For message secrecy, we find that The One Time Pad is the best that one can do: any other scheme that guarantees message secrecy unconditionally ends up needing a key of length equal to the length of the message that needs to be transmitted in secrecy.

Note that by saying that secrecy or authenticity of the message is guaranteed unconditionally we do not mean that the probability of an attacker discovering the contents of a "secret" message or forging a new message is zero. There is clearly a (very low) probability that an attacker might get lucky and guess the One Time Pad key and thus gain access to the message, which no longer remains secret. Similarly, there is clearly a (very low) probability that an attacker can make a lucky guess, find the secret key for the tagging scheme and forge a message.

It turns out that for unconditionally ensuring message authenticity we can do much better. It all depends on what probability of attacker success we are willing to live with. For most practical situations, one might find that a chance of 1 in 2^{80} that an attacker might be able to violate the authenticity of message transmission is small enough to be ignored. To ensure this, we need $80 + O((\log(|m|)^n))$ bits of key, where $|m|$ is the length of the message we need to send and $n \in \mathbb{N}$. Thus, while for message secrecy we would have needed $|m|$ bits of key, for authenticity we need much less.

This however is still a non-trivial amount of key material, because one is needed for each message that needs to be sent. In subsequent lectures we will discuss ways to address this issue. Stay tuned !