Last Revised: 10.23.97.

ECS 122A — Handout E — Fall 97

Message Authentication

24 October 1997

7 Message Authentication

Though encryption is the goal most commonly associated to cryptography, message authentication is arguably *more* central to building secure systems. As information flows through a network, we may or may not care that it remains private, but we almost certainly do care that it be properly attributable to its source. Indeed this is essential if we are to achieve even the most elementary degree of access control.

Message authentication is used to provide data integrity and proof of origin. These are actually a single goal, a fact which many people do not seem to understand. In ISO parlance message authentication is called "data origin authentication."

Solutions to message authentication allow for a party A to send a message to a party B in such a way that if the message is modified in route to B, B will almost certainly detect this. In the private key case, the tool enabling this is called a *message authentication code*. Here we will examine both the notion of message authentication codes and the question of how to achieve them.

7.1 The basic setup

Parties A and B share a key a and communicate across a channel on which is present an *active* adversary. This adversary can modify data in transit or create new messages of her own choosing. The adversary can read and perhaps even influence the data on the line.

Attacks of the sort sketched above can be quite damaging. For example, say that you have bought something, and in order to be paid the seller sends you his or her bank account number. The adversary modifies this to be her own bank account number, so she gets paid instead. Similarly, say you are buying goods and you send the shop your delivery address electronically. But the adversary modifies your message and substitutes in her own delivery address. In this way she gets the goods for free, and you are denied them.

Let us emphasize from the beginning that we are not concerned here with whether the adversary knows the content of the data. From the point of view of the message authentication goal, she is welcome to know the content of messages. What we want is that any attempt to modify the data is detected. Of course in practice we may want both privacy and authenticity but, if so, we should use both a symmetric encryption scheme and a message authentication code. One has to be a little bit careful as to how to use these two tools in conjunction, but this is not hard.

Again: privacy and authenticity are entirely different things, with no obvious connection. There is quite a widespread fallacy in certain parts of the literature that authenticity is achieved by encryption. (For example, see Dorothy Denning's book, p. 108.) Not only is this false (typically), but it is just the wrong point of view: authenticity, as a goal, has nothing to do with encryption.

Our tool for achieving authenticity of data (in the symmetric setting) is called a *message authentication scheme* (which we will also call a *message authentication code,* or *MAC*). A MAC enables the legitimate parties to detect attempts at forging data (including the tampering of old data). A message authentication scheme includes a *MAC generation algorithm* which, on input of a message x and the key a, generates a short *tag*, or *MAC*. Normally we think of this tag as being transmitted along with the message x, as our way of authenticating x. A message authentication scheme also includes a *MAC verification algorithm* which, on input of a message, a key, and a (purported) tag, answers 0 or 1, with 1 indicating that the message x is deemed authentic, and 0 indicating that it is not. We will denote these two algorithms by MACG and MACV, writing the key as a subscript.

7.2 How NOT to do it

How do we authenticate a message x? One popularly used approach is for A to simply encrypt the data with key a. When B decrypts the message, "if it makes sense" then he assumes it came from A.

The approach just outlined assumes that the adversary cannot compose a string which, when decrypted, "makes sense." As we shall see, this is not something that encryption guarantees.

As a simple example, suppose we encrypt with the "ideal" encryption scheme, a one-time pad. To authenticate a message x, XOR it with your one-time pad a. The resulting message is certainly encrypted, but is it authenticated? Absolutely not. Suppose that the adversary F knows that the message was the string `yes`, as 24 ASCII bits. Then it is trivial for the adversary to change this "authenticated message" to the message `no!`. All the adversary has to do is to XOR the ciphertext with the appropriate constant, $\texttt{yes}\oplus\texttt{no!}$.

A more realistic suggestion one sometimes hears is that the message x be CBC-encrypted with a block cipher such as DES. Let $x = x_1 \cdots x_n$ be comprised by a sequence of n 64-bit blocks. Then to DES CBC encrypt x with DES using key a and "initialization vector" IV, we compute $y = y_1 \cdots y_n$ where $y_i = \mathrm{DES}_a(x_i \oplus y_{i-1})$, for $y_0 = \mathrm{IV}$. The ciphertext is (IV, y).

Does the above provide for message authenticity? Again the answer is no. One trivial way to forge messages is to excise pieces of known messages. For example, given the "authenticated" message $(\mathrm{IV}, y_1 \cdots y_n)$ for $x_1 \cdots x_n$, we can form authenticated messages for any $x_i \cdots x_j$. We can also generate authenticated messages of the same length as the original messages: for example, given $(\mathrm{IV}, y_1 y_2)$, a DES-CBC "authenticated" versions of $x_1 x_2$; and given $(\mathrm{IV}', y_1' y_2')$, a DES-CBC "authenticated" versions of $x_1' x_2'$; it is easy to produce a DES-CBC "authenticated" versions of $x_1.(x_2' \oplus y_1' \oplus y_1)$ — namely, (IV, y_1, y_2'). Such "splicing" attacks on CBC encryption make it useless for providing message authentication.

Some people think that if you "add redundancy" and the CBC encrypt that this will fix the problems above. But it does not.

Let us now move on developing the notion of a message authentication scheme more formally.

7.3 The notion of a secure MAC

7.3.1 Syntax

As is our custom, we describe the syntax before discussing the security.

A message authentication scheme is a 3-tuple $\mathcal{M} = (\mathrm{MACG}, \mathrm{MACV}, \mathcal{K})$ called the "MAC generation" (or signing) algorithm, the "MAC verification" algorithm, and the "key generator."

MACG takes a four-tuple (keys, message, state, coins) to a two-tuple (tag, new-state):

$$\mathrm{MACG} : \underbrace{\Sigma^*}_{\text{key}} \times \underbrace{\Sigma^*}_{\text{message}} \times \underbrace{\Sigma^*}_{\text{state}} \times \underbrace{\Sigma^\infty}_{\text{coins}} \to \underbrace{\Sigma^*}_{\text{tag}} \times \underbrace{\Sigma^*}_{\text{new state}}$$

The coins are used for probabilistic MACs and the state/new state for stateful MACs. That is, we allow a MAC to flip coins, or to remember state. We often just write

$$t \leftarrow \mathrm{MAC}_a(x)$$

where a is the key, x is the message, t is the tag, and the state, if needed, is implicit, just like the coins.

MACV is a mapping from a three tuple (keys, messages, purported-tag) to the binary decision, 1 or 0:

$$\text{MAC-verify} : \underbrace{\Sigma^*}_{\text{key}} \times \underbrace{\Sigma^*}_{\text{message}} \times \underbrace{\Sigma^*}_{\text{purported tag}} \to \underbrace{\{0,1\}}_{\text{decision}}$$

I know of no example where it is useful or desirable for MACV to be probabilistic or stateful.

The key generator $\mathcal{K}$ is a mapping from coins to the key space.

$$\mathcal{K} : \underbrace{\Sigma^\infty}_{\text{coins}} \to \underbrace{\Sigma^*}_{\text{key}}$$

Usually the key is just a random string of some particular length.

We demand of any MAC scheme $\mathcal{M}$ that if $t \leftarrow \mathrm{MACG}_a(x)$ is a tag output with nonzero probability, then $\mathrm{MACV}_a(x) = 1$.

7.3.2 Security

An adversary succeeds in *existential forgery* under an *adaptive chosen message attack* if, after asking for the MACs of messages, $x_1, \ldots, x_q$, the adversary can come up with some *new* message x^* together with a valid MAC t^* for it. The messages the adversary asks may depend on MACs observed in response to previous messages. Indeed that's why the attack is called *adaptive*. The message the adversary creates at the end may be "meaningless" — just so long as it is new. That's why we call the adversary's goal *existential forgery.*

7.4 Achieving a secure MAC

We now describe a method for achieving a secure message authentication code. And we will prove its correctness.

7.5 Wegman-Carter MACs

The final type of MAC which we consider today are the so-called *Wegman-Carter MACs.* This is one of the original applications for Carter and Wegman's idea of universal hash function families. Wegman-Carter MACs have many pleasant properties. They are the most promising avenue for software-efficient MACs. I predict that they are the "way of the future"— that the next generation of practical MACs will be of the Wegman-Carter variety.

You may have seen in your algorithms class the notion of an *almost universal-2* family of hash functions. Let $\epsilon \in \mathsf{R}^+$ be a positive real numbers. Then a *set* of hash functions $\mathcal{H} = \{h : D \to \{0,1\}^k\}$ is said to be ϵ-AU2 (ϵ-almost universal 2) if for every $x, y \in D$ where $x \neq y$, $\Pr_h[h(x) = h(y)] \leq \epsilon$. That is, for any two points in the domain of the hash function, the chance that they collide under a randomly chosen hash from the family is very small.

We emphasize that the above is not a "cryptographic" property at all. One can *prove* —no assumptions— that various natural families of hash functions are ϵ-AU2, for small values of ϵ.

We will need something just slightly stronger than $\mathcal{H}$ being ϵ-AU2:

Definition 7.5.1 *Let $\epsilon \in \mathsf{R}^+$ be a positive number. A set of hash functions $\mathcal{H} = \{h : D \to \{0,1\}^k\}$ is said to be ϵ-AXU2 ("ϵ-almost XOR universal-2") if for every $x, y \in D$ where $x \neq y$, and every $c \in \{0,1\}^k$, $\Pr_h[h(x) \oplus h(y) = c] \leq \epsilon$.*

Notice that this is a strict generalization of $\mathcal{H}$ be ϵ-AU2; an ϵ-AU2 family need have the small collision probability only for $c = 0$.

So: how is an ϵ-AXU2 family of hash functions going to help us authenticate messages? Here is the method. Assume, first of all, that the sender and the receiver share an infinite random string a and a randomly chosen hash function $h \in \mathcal{H}$ from an ϵ-AXU2 family of hash functions. Then think of a as being divided into k-bit strings, $a_1, a_2, \ldots$. The scheme which we call WC[$\mathcal{H}$] is stateful. MAC generation uses a counter, i, which is initially -1. To MAC a message x, MACG increments i and then returns a tag of $(i, a_i \oplus h(x))$. The MAC verification procedure, given x and a purported tag for it (i, t), accepts if and only if $t = a_i \oplus h(x)$. As always, MAC verification is stateless.

Theorem 7.5.1 *Suppose $\mathcal{H} = \{h : D \to \{0,1\}^k\}$ is ϵ-AXU2. Then* WC*[$\mathcal{H}$] is ϵ-secure.*

Proof. Suppose that adversary A attacks scheme WC[$\mathcal{H}$]. Adversary A asks some sequence of queries, $x_1, \ldots, x_q$, obtaining for each query a MAC, $T_1, \ldots, T_q$. Then A outputs a pair (x^*, T^*). The tag T^* is (i^*, t^*) for some number $i^* \in \mathsf{N}$ and some string t^*.

Let E be the event that $i^* > q$. Suppose event E holds when A executes. Then (x^*, T^*) is a valid forgery if and only if $t^* = a_{i^*} \oplus h(x^*)$ for some a_{i^*} which A has received absolutely no information about. That is, a_{i^*} is a uniformly distributed random variable uncorrelated to the information which A has received. Thus that chance that t^* has the desired form is exactly 2^{-k}, which is $\leq \epsilon$.

Suppose that E does not hold, so $1 \leq i^* \leq q$. Again (x^*, T^*) is a valid forgery if and only if $t^* = a_{i^*} \oplus h(x^*)$, but now A has one and only one piece of information correlated to $a_{i^*} \oplus h(x^*)$; namely, A knows $a_{i^*} \oplus h(x_{i^*})$. So to produce a valid t^* adversary A must select x^* and t^* such that $h(x^*) \oplus t^* = h(x_{i^*}) \oplus t_{i^*}$. This equality holds if and only if $t^* \oplus t_{i^*} = h(x^*) \oplus h(x_{i^*})$. Let $c = t^* \oplus t_{i^*}$. We are saying that adversary A wins if and only if $h(x^*) \oplus h(x_{i^*}) = c$. By the definition of ϵ-AXU2, this event happens with probability $\leq \epsilon$.

Let p be the probability that event E holds. Our argument then gives that A forges with probability $\leq p2^{-k} + (1-p)\epsilon \leq \epsilon$. $\diamondsuit$

To make a practical Wegman-Carter MAC we use use a finite pseudorandom function to make the a_i-values. Let $F : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^k$. Then WC[$\mathcal{H}, F$] is defined as follows. Let the secret key be a random value $a \in \{0,1\}^\kappa$ and a random $h \in \mathcal{H}$ drawn for some ϵ-AXU2 family of hash functions $\mathcal{H} = \{h : D \to \{0,1\}^k\}$. Scheme WC[$\mathcal{H}, F$] is stateful. MAC generation uses a counter, i, which is initially -1. To MAC a message, MACG increments i and then, if the counter value is less than 2^n, MACG returns a tag of $\langle i, F_a(\langle i \rangle_n) \oplus h(x) \rangle$. The MAC verification procedure, given x and a purported tag for it $\langle i, t \rangle$, accepts if and only if $t = F_a(\langle i \rangle_n) \oplus h(x)$. Using standard arguments we get:

Theorem 7.5.2 *Suppose $\mathcal{H} = \{h : D \to \{0,1\}^k\}$ is ϵ-AXU2, and let $F : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^k$ be a finite pseudorandom function. Suppose F is*

(t, q, δ)-secure (i.e., no adversary running in time $\leq t$ and making $\leq q$ queries can distinguish F-output from truly random output). Then $\mathrm{WC}[\mathcal{H}, F]$ *is a (t', q', μ', e')-secure MAC, where*

$$\begin{aligned} t' &= t + O(\mu) \\ q' &= q + 1 \\ \mu' &= \textit{arbitrary} \\ \epsilon' &= \epsilon - \delta \end{aligned}$$

Now we have an important piece of remaining business: how do we construct an ϵ-AXU2 family? Here is a simple recipe.

Suppose the message space is $D = \{0,1\}^m$. Fix a security parameter k. Define the hash family $\mathcal{H}_k$ as follows: a hash function $h \in \mathcal{H}_k$ is represented by a $k \times m$ matrix H, and the definition of this function is $h(x) = Hx$, where the right hand side indicates multiplication of the matrix H by the m-bit column vector x. A random element of this family is selected by choosing a random $k \times m$ matrix H.

Proposition 7.5.1 *$\mathcal{H}_k$ is 2^{-k}–AXU2.*

Proof. We must show that for all distinct $x, y \in \{0,1\}^m$, and all $c \in \{0,1\}^k$, $\Pr_H[Hx \oplus Hy = c] \leq 2^{-k}$. By the linearity of matrix multiplication,

$$\max_{x \neq y} \Pr_H[Hx \oplus Hy = c] = \max_{x \neq 0} \Pr[Hx = c]$$

Since x is non-zero, let i be an location such that $x[i] = 1$. Then Hx can be regarded as column i of H XORed with a random variable uncorrelated to column i of H. Column i of H is itself uniformly distributed. We conclude that Hx is uniformly distributed. Therefore the chance that Hx equals any particular value, c, is exactly 2^{-k}. $\diamondsuit$

How practical is $\mathrm{WC}[\mathcal{H}, F]$? Well, that depends largely on the function $\mathcal{H}$. (It doesn't depend much on F since, presumably, it doesn't cost much to apply F *once* to its short-length argument.) Using the matrix multiplication scheme, though, $\mathcal{H}$ is not terribly fast to compute when messages are long, and $h \in \mathcal{H}$ takes quite a bit of storage as well.

Here's a trick to make the scheme completely practical. Let *hash* be a collision-intractable hash function which returns ℓ bits. For example, one might take *hash* to be SHA-1, in which case $\ell = 160$. Let $h \in \mathcal{H}$ a randomly selected hash from an ϵ-AXU2 family of hash functions. The domain of the hash functions is ℓ-bit strings, and the range is k-bit strings. Let F be be a finite pseudorandom function which outputs k-bit strings. Then $\mathrm{MACG}_{a,h}(x)$ is the stateful scheme that increments counter i and then returns $\langle i, F_a(i) \oplus h(\mathit{hash}(x)) \rangle$.