CS276 Cryptography	**Spring 2006**

Lecture 10: 2006-02-16

Lecturer: David Wagner — *Scribe: Karl Chen*

10.1 Message Authentication Codes

We would like to design a secure MAC without using exponentially large keys. Let's consider the straw-man method from the previous lecture, where we created a MAC for a single-bit message by using a separate key for each possible message:

0	k_0
1	k_1

If we want to send 2 bits, which is twice as hard, we need four separate keys:

00	k_{00}
01	k_{01}
10	k_{10}
11	k_{11}

If we send the bit and the key, then this is a perfectly secure MAC, as long as we only use it once, and the keys are chosen uniformly at random. However, this uses an exponential number of bits for the keys. How can we do this efficiently?

10.2 Efficient MACs

We model the problem as Alice sending Bob exactly one message and tag $(m, t = k_m)$

$$\text{Alice} \xrightarrow{(m, k_m)} \text{Bob}$$

The adversary sees that message and sends one more spoofed message (m', t').

$$\text{Alice} \xrightarrow{(m', t')} \text{Bob}$$

Insight: We're only sending the (m, t) for one row – the advesary never sees the other rows.

If we only send a pair once, it suffices that they are pairwise independent. I.e., if we look at any two rows, they are independent. It is okay if there is correlation when we look at three rows at once; the adversary cannot look at three rows at once.

10.3 2-Universal hash functions

Suppose we have a 2-universal hash function $H : \text{keys} \times M \rightarrow \{0,1\}^{80}$. (Here we choose 80 bits without loss of generality.)

Then we can let each MAC key be a hash on the input:

00	$k_{00} = H_k(00)$
01	$k_{01} = H_k(01)$
10	$k_{10} = H_k(10)$
11	$k_{11} = H_k(11)$

A 2-universal hash function has the property that any pair of hashes is pairwise independent, i.e. the probabilities of a hash of random input matching are the same.

Definition: H is a 2-universal hash function if

$$\Pr[K \xleftarrow{\$} keys; H_k(x) = y \text{ and } H_k(x') = y'] = \frac{1}{2^{160}}$$

10.3.1 Example 1

Suppose the key is a $80 \times l$ matrix:

$$\begin{aligned} H_{M,b} :& \{0,1\}^l \times \{0,1\}^{80} \\ M :& 80 \times l \text{ boolean matrix} \\ b :& \{0,1\}^{80} \end{aligned}$$

The input is a column vector of length l. The hash function performs matrix multiplication: $H_{M,b}(x) = Mx + b$.

(In finite field terminology, we are operating over $GF(2)$.)

We can show this is 2-universal. Proof outline: Fix x, y. We're looking for M, b satisfying these two equations:

$$Mx + b = y$$

$$Mx' + b = y'$$

The probability of a random M satisfying the equation $M(x' - x) = y' - y$ is $1/2^{80}$ (since $x' - x \neq 0$), and given this value of M, the probability that a randomly chosen b satisfies both equations above is also $1/2^{80}$. So the overall probability is $1/2^{160}$.

This example uses big keys, but other methods are available. The key size can be as small as linear in output length and logarithmic in input length. 2-universal hash functions are truly secure: there are *no* requirements on the attacker's computational power.

10.3.2 Discussion

Strong 2-universal hash functions only require logarithmic growth in their key length. Contrast this to one-time pads which require linear growth. Also, we can come up with *very* fast 2-universal hashes, faster than any encryption code we know of. There are implementations that use only a few instructions per byte—almost as fast as `memcpy` !

An example use of 2-universal hash functions is in networking stack implementations. Historically, lots of network stack implementations had worst-case performance that was quadratic due to hash collisions, so an attacker could craft packets to bring down your system. One defense against this kind of algorithmic DoS would be to use 2-universal hash functions.

Lemma 10.1 *(Ignoring $H(x')$) $H(x)$ is uniform.*

$$\Pr[k \xleftarrow{\$} keys; H_k(x) = y] = 1/2^{80}\ \forall x, y$$

Lemma 10.2 *Given the value of $H(x)$, $H(x')$ is uniform, if $x' \neq x$.*

$$\Pr[k \xleftarrow{\$} keys; H_k(x') = y' | H_k(x) = y] = 1/2^{80}\ \forall x, x', y, y' (x \neq x')$$

Colloquially: If I tell you what x maps to, and ask you the distribution of what x' maps to, I might as well not have told you about x. I.e., if I evaluate the hash function on one input, it tells you nothing about another input.

10.4 Security of a MAC

Given a $(\infty, 1, 1, 1/2^{80})$-MAC, we actually have a $(\infty, 1, q_v, q_v/2^{80})$-MAC. The restriction that the adversary can only try to forge once is not important, was just an artifact. However, it is essential that the adversary sees only one properly MACed message.

The adversary sees x and y and generates x' and y'. x' and y' can depend on x and y, but cannot depend on k except as y depends on k.

Consider any adversary A trying to break this MAC scheme by forging a message. By assumption, the adversary gets at most one verification query. If the adversary makes no queries then it obviously cannot forge the MAC. Assume without loss of generality that the adversary makes exactly one query, because if the adversary didn't, it could just make a query and ignore the result. Similarly, we can assume without loss of generality that the order is to query the signing oracle then the verification oracle.

In summary, the attacker's steps are:

1. Ask signing oracle: $y \Leftarrow H_k(x)$
2. Generate $x' \neq x$, y'
3. Send to verification oracle

$$\text{A} \xrightarrow{\quad x \quad} \text{Signing oracle} \xrightarrow{y = H_k(x)} \text{A} \xrightarrow{\quad x', y' \quad} \text{verification oracle}$$

The adversary is allowed to be randomized; however, without loss of generality we can determinize the adversary. If there is one that can successfully forge using randomization, then there is also one that does not need randomization. Here is how: consider that the randomized adversary used a series of coin tosses to produce x. The probability of the adversary winning is the average of the probability of winning for each particular coin toss result. So there must be some particular coin toss that was at least the average, and we can simply hard-code that coin toss.

10.5 Using a 2-universal hash for a MAC

Claim: Given a 2-universal hash as a MAC, no matter the adversary's algorithm, his probability of winning is low.

Theorem 10.3 *If H is strongly 2-universal, then it is a $(\infty, 1, 1, 1/2^{80})$-secure MAC.*

Proof: Fix A. Sum over all the values of x', y' the adversary might choose.

$$\begin{aligned}\text{Adv } A &= \sum_{x',y'} \Pr[H_k(x') = y' | H_k(x) = y] \cdot \Pr[A \text{ chooses } x', y' \text{ given } x, y] \\ &\leq \frac{1}{2^{80}}\end{aligned}$$

■

In practice, a probability of $1/2^{80} + \epsilon$ is good enough, so people often use "almost" 2-universal hash functions, which allow more flexibility.

So far we have restricted Alice only sending one message under the key. If we wanted to allow two messages, we would require 3-universal hash functions, and so on. 3-universal hash functions are hard to construct. But we can just use a random function, since that would be N-universal. And we know that we can use a pseudorandom function in place of a random function.

10.6 Using a PRF for a MAC

One of the issues with using a PRF for a MAC is that authentication usually deals with long packets — for example, Ethernet packets are 1500 bytes, but AES only works on 128-bit inputs. The solution is to hash the 1500 bytes to 128 bits using a 2-universal hash function.

Thus the key is actually a pair: the hash key k_0 and the AES key k_1.

$$\begin{tikzcd} & \text{Msg} \arrow[d] \\ k_0 \arrow[r] & H \arrow[d] \\ k_1 \arrow[r] & \text{AES} \arrow[d] \\ & z_i \end{tikzcd}$$

Let

$$f_{k_0,k_1}(x) = AES_{k_1}(H_{k_0}(x))$$

Define

$$Coll \equiv \exists i \neq j.\ H_{k_0}(x_i) = H_{k_0}(x_j)$$

By union bound,

$$\Pr[Coll] \leq \sum_{i \neq j} \Pr[H_{k_0}(x_i) = H_{k_0}(x_j)]$$

Since $x_i \neq x_j$, the probability they collide under H is at most $1/2^{128}$, so

$$\Pr[Coll] \leq \sum_{i \neq j} \frac{1}{2^{128}} = \binom{q}{2} \frac{1}{2^{128}}$$

Theorem 10.4 *If H is strongly 2-universal, and AES is a (t, q, ϵ)-PRF, then f is a $(t, q, +\binom{q}{2}/2^{128})$-MAC.*

Proof: Fix A; we want to compute the advantage of A. Construct two hybrid functions.

$$\text{Adv } A = \Pr[A^{AES_{k_0} \circ H_{k_1}} = 1] - \Pr[A^{R * H_m} = 1] + \Pr[A^{R \circ H_m} = 1] - \Pr[A^{R'} = 1]$$

$$\begin{aligned} \Pr[A^{R \circ H_{k_0}} = 1] - \Pr[A^{R'} = 1] &= \Pr[A^{R * H_{k_0}} | Coll] \Pr[Coll] + \Pr[A^{R * H_{k_0}} = 1 | \neg coll] \Pr[\neg coll] - \Pr[A^R = 1] \\ &\leq \binom{q}{2}/2^{128} + \Pr[A^{R \circ H_{k_0}} = 1 | \neg coll] - \Pr[A^{R'} = 1] \end{aligned}$$

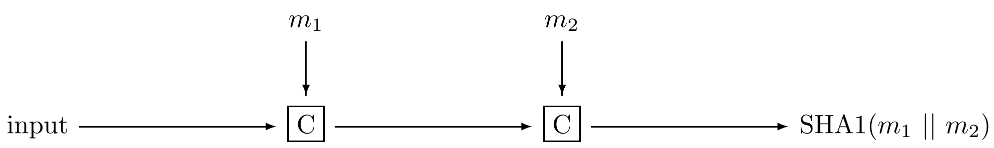
If there is no collision at the input to R, then the outputs of $R \circ H_{k_0}$ are independent uniform values, just like the outputs of R'. Consequently, the latter two probabilities in the equation above are actually equal. ■

10.7 HMACs: Introduction

Hash functions map a large bitstring to smaller number of bits:

$$\{0,1\}^\infty \to \{0,1\}^{160}$$

SHA-1 is a hash function designed to require 2^{80} steps to attack by brute force. Since its design a short-cut attack has been found that requires only 2^{65} steps, which is still barely out of reach. Designing the compression function is a black art.



10.7.1 Using a hash function for a MAC

In 1990, the community started realizing the need for a protocol for message authentication, but it was unclear how to do it. Here were some proposals to the IETF working group:

1. $F_k(x) = \text{SHA1}(k \mid\mid x)$

2. $F_k(x) = \mathrm{SHA1}(x \mid\mid k)$

Scheme 2 is vulnerable to off-line attack. We would like to require the attacker to have to make 2^{80} oracle queries rather than just 2^{80} offline computations.

Scheme 1 is completely broken: a null message becomes simply $\mathrm{SHA1}(k)$, which allows the attacker to get any string; or in general, $F_k(x)$ gives the attacker $F_k(y)$ for any y prefixed by x.

The next suggestion:

3. $F_k(x) = \mathrm{SHA1}(k \mid\mid x \mid\mid k)$

seemed okay at first, but turned out to have a subtle attack. Now the community wanted a rigorous way to analyze hashing MAC techniques. These schemes were all designed ad-hoc without proof of security.

Finally **HMAC** was proposed, which was rigorously designed, and all was well; overnight it became the standard MAC. In the next lecture we discuss HMAC and how it is a 2-universal PRF hashing methodology.