1 Lecture 7: Message Authentication Codes (MACs)

In a similar vein to signatures, what if we want to send smaller messages, but still want some sort of verification, i.e. to prove that the message being read has not been tampered with. This is the idea of MAC's.

Each message is sent with some corresponding MAC, generated from a secret key s and the message m itself (denoted $MAC_s(m)$. Then, when the receiver gets the message, they can use the secret to verify that the MAC corresponds to the message. This makes it so that eavesdropper would not be able to tamper with the message, since the receiver would be able to tell that the tampered message does not have a correct MAC.

1.1 Single-use MAC

If we are only sending one message, one potential candidate for the secret and MAC function is: $s = (a, b)$, $MAC_s(m) = a \cdot m + b$, where all calculations are done in some finite field F. Then, each message-MAC pair specifies a point on the line $y = ax + b$ in the finite field F.

So if only one message is sent, then only one point on the line is specified, so an eavesdropper does not have enough information to generate a proper MAC for a tampered message (unless, perhaps, the tampered message satisfies $m' = k|F| + m$). So this protocol is information-theoretically secure for one message being sent.

However, note that if we send two messages, this completely determines the line, so it is very important that this is used *only once* for any given secret key s.

1.1.1 Pairwise Independence

We can generalize this notion to something called "**Pairwise Independence**". The idea is that one point should not pin down any others within the distribution. This is precisely what we have going on with the single-use MACs.

Another way to think of this is in terms of a family of functions: given the output of one function on one input, the distribution of possible outputs for a different input on this (unknown) function should be uniform. For example, with the single-use MAC, the first output does not give away anything about the line, and since all possible lines give a uniform distribution of possible outputs for all other inputs, we have pairwise independence.

Thinking about this in terms of families, we may also formulate this for families of hash functions, similarly to how we defined Collision-Resistance.

For all of the following, "negligible probability" refers to any probability $\leq \frac{1}{|F|}$, where we are in the finite field F.

Recall that a family of hash functions satisfies collision resistance if, given one hash from the family, a **PPT** adversary cannot select two inputs u, v such that $h(u) = h(v)$, except with negligible probability.

In a similar vein, we will define a family of hash functions to be pairwise independent if, before seeing a randomly selected hash from the family, a **PPT** adversary cannot select two inputs u, v such that $h(u) = h(v)$, except with negligible probability.

It seems clear that the collision resistant game is an easier one to win, and as such, would be a stronger requirement for security of a family of hash functions. We'll now define something that sits between pairwise independence and collision resistance in terms of security: **universal one-way hash functions** (UWH's).

This is sort of a hybrid between the previous two: the adversary picks u first, then the random hash h is revealed, and then the adversary must try to pick a second input v such that $h(u) = h(v)$, and the adversary wins if it can do this with non-negligible probability.

Note that the difficulties fall in the order: pairwise independent < UWH < collision-resistant.

1.1.2 Digital Signatures with UWH's

Recall that the purpose of using collision-resistant hashes in our signature schemes was so that forging a signature of a hashed document would be just as difficult as forging a signature of the document itself. But do we actually need collision-resistant-level security here? Let's think about it.

When we made the linked list of keys, we started with our own given key PK_0, and we used it to sign a new key PK_1. Let's say an adversary wanted to forge our signature within the bigger picture of this scheme. Since the adversary does not know any of our secret keys, they'll have to come up with a private-key public-key pair of their own.

What they would essentially have to do is take the PK_0 signature of PK_1, find a new key PK^* such that $h(PK_1) = h(PK^*)$. So it's not good enough for the adversary to simply come up with two arbitrary values that collide in the hash. The adversary has to specifically find some string that is a hash collision with PK_0.

So, this is as if the adversary picked PK_1, had the hash revealed, and then had to find a collision with PK_1, which is precisely what happens in the UWH game. Thus, the level of security we need to make our digital signature scheme work is in fact, only UWH-level, and not collision-resistant-level!

1.1.3 UWH's from 1WP's

Consider a 1WP g which permutes the set of $n+1$-bit strings. Then, define our "chop" function $c_{a,b}(y)$ to be $ay+b$, with the last bit deleted. We now assert that $U = c_{a,b} \circ g : \{0,1\}^{n+1} \to \{0,1\}^n$ is a UWH.

Note that since c is 2-to-1 and g is 1-to-1, the resulting U is also 2-to-1.

Suppose that we have some adversary $A \in \mathbf{PPT}$ that, when given x, a, b, g, is able to find x' such that $U(x) = U(x')$, with non-negligible probability. We then construct the following adversary A':

0. A' is given a 1WP g and an output y.
1. A' selects a random x' and uses g to compute $y' = g(x')$. (If somehow $y' = y$, we are done. Otherwise, proceed with the following steps)
2. A' selects a random z, and uses the system of equations $ay+b = z||0$, $ay'+b = z||1$ to solve for a, b and create a "chop" function using a, b.
3. A' uses $A(x', a, b, g)$ to find an $x'' \neq x'$ such that $c_{a,b}(g(x'')) = c_{a,b}(g(x'))$.

Note that since g is a 1WP and $x'' \neq x'$, we must have $y'' \neq y'$. And since $y' \neq y$ and $c_{a,b}$ is 2-to-1, we must have $g(x'') = y'' = y = g(x)$ (in fact, we have $x = x''$), so our adversary A' has inverted the 1WP.

Thus, the function U is a UWH, as long as the g we use properly fits all specified properties of a 1WP.

1.1.4 Using UWH's

As a final footnote to all of this, we note that we have only defined a UWH which shrinks the domain by 1 bit. It can be shown that if we want to shrink from m bits to n bits we need to use $m-n$ different 1WP's if we are applying them in this way. So really, we have done all we can, and if we want to do more shrinking, we simply need to chain UWH's of this form.

We will note that chaining them gives a similar effect to Merkel Hash Trees, where the final composition is as strong as any of the intermediates used to make it, since a collision in the full composition implies a collision somewhere within the tree/composition.

In practice, this would be impractical, so other number theoretic tricks are used to speed up the process.