Lecture 10 - MAC's continued, hash & MAC

Boaz Barak

March 3, 2010

Reading: Boneh-Shoup chapters 7,8

The field GF(2^n) . A *field* $\mathbb{F}$ is a set with a multiplication $(\cdot)$ and addition operations that satisfy all the usual rules, and also containing a zero and one element, where for every $a \in F \setminus \{0\}$ there are elements $-a, a^{-1} \in \mathbb{F}$ such that $a + (-a) = 0$ and $a \cdot (a^{-1}) = 1$. A classical example for a field is the real numbers $\mathbb{R}$, but there are other fields (complex, rational numbers), and in particular there are also *finite* fields. One example is the field $\mathbb{F}_p$ of the numbers $\{0, .., p-1\}$ with addition and multiplication done modulo p. (Inverse is found using the GCD algorithm that gives us for numbers a, p two numbers b, c such $ab + pc = gcd(a, p) = 1$ (in the case p is prime and $a \in \{1..p\}$), thus $b = a^{-1} \pmod{p}$.

For every number n, there is also a field of 2^n elements, denoted by GF(2^n). (There is in fact a field of p^n elements for every prime p.) Again we have the same addition and multiplication operations, with addition corresponding to XOR. All you really need to know is that these operations can be easily done by a computer. Still if you care how this field is defined then read on:

For any field $\mathbb{F}$, a *polynomial* of degree at most d over $\mathbb{F}$ is a vector $a = (a_0, a_1, \ldots, a_d) \in \mathbb{F}^{d+1}$ that we think of as the function $a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_d x^d$. We can *multiply* two polynomials a, b of degree at most d to obtain a polynomial $c = a \times b$ of degree at most $2d$ by setting $c(x) = a(x)b(x)$ or equivalently, $c = (c_0, \ldots, c_{2d})$ where $c_i = \sum_{j<i} a_j b_{i-j}$. Similarly we can use a version of the long division algorithm to *divide* a polynomial a by a polynomial b to obtain that $a = x \times b + y$, where x is some polynomial and y, known as the *remainder*, has degree smaller than the degree of a. This is a unique representation, and in this case we write $y = a \pmod{b}$. We say that two polynomials a, b are *co-prime* if there is no polynomial c of degree at least 1 that divides both a and b. One can show again using a variant of the GCD algorithm that for every coprime a, b there exist polynomials x, y such that $ax + by = 1$ or in other words $x = a^{-1} \pmod{b}$ A degree d polynomial p is *irreducible* if its only divisors are multiples of p, or multiples of 1. (In other words it is coprime with any polynomial of degree between 1 and $d - 1$.)

Note that if $\mathbb{F}$ is a finite field then there are exactly $|\mathbb{F}|^n$ polynomials of degree at most $n - 1$. The field GF(2^n) is the field of polynomials of degree at most $n - 1$ in $\mathbb{F}_2$, where addition and multiplication is done modulo some fixed irreducible degree-n polynomial p. Thus an element $a \in$ GF(2^n) is a string $(a_0, \ldots, a_{n-1})$ identified with the polynomial $a(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$. One can easily verify that addition corresponds to XOR'ing the vectors of coefficients.

We will often identify n bit strings with elements of the field GF(2^n), and so assume we can always add and multiply n bit strings.

You can read (much) more about finite fields in Sections 7.2, and chapters 19 and 20 of Victor Shoup's book "A Computational Introduction to Number Theory and Algebra" also available online at `http://www.shoup.net/ntb/`

Pairwise independent hash functions We now discuss a *non-cryptographic* tool (not to be confused with cryptographic hash functions) that is very useful in cryptography and many other areas of computer science. We start with the following definition:

A collection of functions $\{h_k\}$ where $h_k : \{0,1\}^n \to \{0,1\}^m$ is *pairwise independent*, if for every $x \neq x' \in \{0,1\}^n$ and $y, y' \in \{0,1\}^m$,

$$\Pr_k[h_k(x) = y \ \wedge \ h_k(x') = y'] = 2^{-2m} \tag{1}$$

Note that if h_k was completely random function then (1) would hold, but also if you had three distinct inputs x, x', x'' then the probability that h_k evaluated on these inputs is equal to any three fixed outputs would be 2^{-3m}, and generally if you had ℓ distinct inputs the corresponding probability would be $2^{-\ell m}$. Thus the pairwise independence definition just asks for this condition for $\ell = 2$, and so possibly could be satisfied by a non random (and hence potentially efficiently computable) function.

The reason this is called pairwise independent is that the condition (1) means that if we define the following 2^n correlated random variables $Y_{0^n}, Y_{0^{n-1}1}, \ldots, Y_{1^n}$ by choosing k at random and letting $Y_x =_k (x)$, then these variables are pairwise independent. That is, for every $x \neq x'$, the random variables Y_x and $Y_{x'}$ are independent.

Construction The following is an efficient construction of pairwise independent hash functions mapping $\{0,1\}^n$ to $\{0,1\}^n$ (i.e. $m = n$): the key is a pair $a, b \in \mathrm{GF}(2^n)$ and the function is $h_{a,b} : \mathrm{GF}(2^n) \to \mathrm{GF}(2^n)$ defined as follows: $h_{a,b}(x) = ax + b$. We now prove that $\{h_{a,b}\}$ is pairwise independent. Indeed, fix $x \neq x'$ and y, y' and consider the probability that

$$h_{a,b}(x) = y$$
$$h_{a,b}(x') = y'$$

or in other words

$$ax + b = y$$
$$ax' + b = y'$$

or equivalently

$$a = (y - y')(x - x')^{-1}$$
$$b = y - (y - y')(x - x')^{-1}x$$

thus clearly there is exactly one pair (a, b) that satisfies these conditions, and so the probability is exactly $\frac{1}{2^{2n}} = 2^{-2n}$.

It's an easy exercise to show that if $\{h_k\}$ is a pairwise independent hash function collection mapping $\{0,1\}^n$ to $\{0,1\}^n$ and $m < n$, then by truncating the output to the first m bits we get a pairwise independent hash function collection mapping $\{0,1\}^n$ to $\{0,1\}^m$.

It's also easy to show that by first padding the input with 1 and sufficient number of zeroes, if we have a pairwise independent hash function collection with input of length n, we can actually transform it into such a collection that takes also strings of length shorter than n as input.

Using pairwise independence for MAC's We now show an alternative approach to use a basic PRF to obtain a MAC of long input length. This simple construction is known as "hash and sign": given a basic PRF $\{f_k\}$ where $f_k : \{0,1\}^m \to \{0,1\}^m$ and a pairwise independent hash function collection $\{h_{k'}\}$ where $h'_k : \{0,1\}^n \to \{0,1\}^m$ (for $n \gg m$) we define the collection $\{g_{k,k'}\}$ of functions mapping $\{0,1\}^n \to \{0,1\}^m$ as follows:

$$g_{k,k'}(x) = f_k(h_{k'}(x))$$

Theorem 1. *$\{g_{k,k'}\}$ is a PRF.*

Proof. Consider an adversary A attacking $g_{k,k'}$. We can assume that A makes at most $T \leq \text{poly}(n)$ queries, and that it never repeats the same query. First, we can replace the call to f_k with a call to a random function F since otherwise we could break the PRF property of $\{f_k\}$. Now our assumption that the adversary never repeats the same query means that he should get new independent random answers for every query. This will happen as long as he doesn't submit a pair of queries $x \neq x'$ such that $h_{k'}(x) = h_{k'}(x')$. Let $i \in \{1, \dots, T\}$, it will suffice to prove the following claim:

Claim 1.1. *The probability that the i^{th} query is the first one in which A succeeds in getting $h_{k'}(x) = h_{k'}(x')$ is at most $T/2^m$.*

(This suffices because if the adversary succeeds at some point then this event must happen for some i between 1 and T, and the union bound shows that this can happen with probability at most $T^2/2^m$ which is negligible.)

Proof of claim. Under the assumption that the i^{th} query is the first one, up until that point the adversary only received random values that are completely independent of k'. Thus we can think of k' as only being chosen at random *after* the adversary makes its i^{th} query. Let $x_1, \dots, x_i$ be all the queries the adversary made up until this point. Then for every $j < i$ we have that

$$\Pr_{k'}[h_{k'}(x_j) = h_{k'}(x_i)] = \sum_{y \in \{0,1\}^m} \Pr[h_{k'}(x_j) = y \ \wedge \ h_{k'}(x_i) = y] = 2^m 2^{-2m} = 2^{-m}.$$

hence by taking a union bound over all of the at most T j's in $\{1..i-1\}$ the claim is proven. □

Almost independent hash functions In applications we often think of the input size n as being much larger than the PRF block size m. But in our construction of pairwise independent hash functions, the key was actually of size n which is far too much. We can make the key shorter using a pseudorandom generator, but there is in fact a more elegant solution. It's easy to see that in the proof we could very well use the following weaker definition:

We say that a collection $\{h_k\}$ of functions mapping $\{0,1\}^n$ to $\{0,1\}^m$ is *ϵ-almost pairwise independent* if for every $x \neq x' \in \{0,1\}^n$ and $y, y' \in \{0,1\}^m$,

$$\Pr_k[h_k(x) = y \ \wedge \ h_k(x') = y'] \leq \epsilon \cdot 2^{-m} \tag{2}$$

If $\epsilon = 2^{-m}$ this is the same condition as (1) but if ϵ is negligible (say $2^{-m/2}$) then it's easy to see that this suffices for the proof of Theorem 1.

The "almost" here buys us a lot. While before to construct a hash mapping n bits to m bits we needed $2n$ long key, now we can use a key of size $2m$ to get an $n2^{-m}$-almost PRF. (In fact the factor 2 in both cases is not needed, since one can use a weaker definition called universal hash function— see the Boneh-Shoup book.)

The construction is the following: For $r, a \in \mathrm{GF}(2^m)$ we consider the following function $h_{r,a}$ mapping $p = p_1, \ldots, p_{\ell-1} \in \mathrm{GF}(2^n)$ to $\mathrm{GF}(2^m)$:

$$h_{r,a} = a + p_1 r + \cdots + p_{\ell-1} r^{\ell-1} + r^\ell$$

That is, $h_{r,a}$ treats p as a polynomial (without a constant term) and outputs $p(r) + a$. (The additional leading term r^ℓ corresponds to padding p with 1, and is used to enable hashing different input lengths. Also, using Horner's rule, if you get the message p in reverse order you can evaluate $h_{r,a}$ in the streaming model. See Section 7.2.1 in BS book.)

We show that $h_{r,a}$ is $(\ell+1)/2^m$-almost pairwise independent. Indeed, if $p(r) + a = y$ and $p'(r) + a = y'$ then we get that $p(r) - y = p'(r) - y'$. Now $p - y$ and $p' - y'$ are two distinct polynomials of degree at most ℓ (we use here the fact that p and p' didn't have a constant term), or equivalently, $p - p' - y + y'$ is a nonzero polynomial of degree at most ℓ, and hence can have at most $\ell + 1$ roots. Thus the probability over a random r that $p(r) - y = p'(r) - y'$ is at most $(\ell+1)/2^m$, and since a is chosen independently of r the probability, conditioned on this, that a random a will satisfy $a = y - p(r)$ is at most 2^{-m}.

Collision resistant hash functions Up to now, it was crucial in all our function collections: PRFs, MACs, and even the non cryptographic pairwise and almost pairwise functions, that the adversary never learns the key to the function. We now show a primitive where this is not required.

A collection of functions $\{h_k\}$ where $h_k : \{0,1\}^n \to \{0,1\}^m$ (for $m < n$) is called a *collision resistant hash functions* (CRH) collection, if for every polynomial-time Eve, the probability that Eve wins in the following game is negligible: a key k is chosen at random, Eve gets the key as inputs, and wins if she manages to output a pair $x \neq x'$ such that $h_k(x) = h_k(x')$.

Note that there trivially will *exist* a pair $x \neq x'$ such that $h_k(x) = h_{k'}(x)$.

Collision resistant hash functions are dual to pseudorandom generators, in the sense that here the goal is to *shrink* the input as much as possible. Similarly to pseudorandom generators, if you can shrink the input by even one bit, you can get a CRH collection that shrinks the bit by a polynomial amount.

Practical note: In practice people usually talk about a *single* hash function rather than a collection. You can think of it that someone chose the key k once and for all, and then fixed the function h_k for everyone to use. (Indeed, these constructions often have various parameters, initialization vectors etc...)

One observation on CRH is that you can always break them much faster than 2^m:

Claim 1.2. *For every CRH $\{h_k\}$ mapping $\{0,1\}^n$ to $\{0,1\}^m$, there is an algorithm that can win the game in time* $\mathrm{poly}(n)2^{m/2}$.

Proof. Use the "birthday paradox" □

One can obtain CRH that map $\{0,1\}^*$ to $\{0,1\}^m$. These in turn imply a construction of PRF with large input by first hashing down the input using the CRH and then applying the

PRF. The same proof goes through. The advantage of the new construction is that the key to the CRH does not have to be part of the secret shared key and can be a public parameter of the system. (We will see a more significant advantage of CRH in the context of *signature schemes*.

Collision resistant hash functions are constructed by composing a basic compression function many times, where the compression function can be based on tools similar to block ciphers.

I would like to say that there is also a construction of CRH based on the PRG Axiom, but this is a longstanding open problem in cryptography. Thus we will have to make another conjecture and assume that CRH exist. (We do know how to construct them based on the hardness of factoring large integers, though this construction is rarely if ever used in practice since it's not as efficient as the other candidates.)

The Random oracle model. Why do we think CRH exist? One reason is that it seems that candidate constructions like SHA-256 and AES are so "crazy" that they have no structure and behave like random functions. Collision resistance is one property that random functions will have, but you can also think of other useful properties. lare and Rogaway (building on work by Fiat and Shamir) suggested that we can model such hash functions as *random functions* (more commonly known as a *random oracle*) and use this in the security proofs.

The idea was to start with a protocol that uses a hash function h, and then analyze the scheme as if h was a completely random function, that is given to the adversary as a black-box, and to see if it is secure in this setting. If it is, we say that it is secure in the *random oracle model*. Since intuitively, we think of crazy hash functions as having random-like behavior, if the scheme is not secure even if h is a random function then it's most likely insecure (and we can find an attack) for any instantiation of that with a particular hash function.

The question is what happens if the scheme *is* secure when h is a random function. The first idea that comes to mind is that then we can make it secure by using a function from a *pseudorandom function collection* instead of h. (Indeed, in the original paper of Fiat and Shamir suggesting a use of this paradigm they (mistakenly) claimed that this will work.) However, there's a big problem here: the definition of pseudorandom functions says that an adversary can not tell apart a black-box computing a random function from a black-box computing a random function from the collection. It says *nothing* about what happens if the adversary is actually given the key for the pseudorandom function. In fact it is clear that if the adversary is given the key k then it can trivially distinguish between a black box computing $f_k(\cdot)$ to a black-box computing a random function.

We'd like to define something like "public-key/publicly evaluatable pseudorandom functions" that remain pseudorandom even to someone that knows the key, but it's not known how to make such a definition that is both useful and not impossible to achieve.

Nevertheless, Bellare and Rogaway argued that if one proves that a scheme is secure in the random oracle model then it says something positive about the security of the real scheme, where h is replaced by a cryptographic hash function such as, say, $SHA - 256$.[1] Their argument was that even if we can't pinpoint what's exactly the security properties of the hash functions that we need, the existence of a proof in the random oracle model implies that sufficiently crazy hash functions will be good enough, and that any attack on the scheme will

[1]Actually that will not be a good idea as SHA-256 can be distinguished from a random oracle, see Sections 8.8.3 and 8.8.4 in Boneh-Shoup.

necessarily have to find some weakness in the design of the cryptographic flaw. Thus, roughly speaking, they put forward the following conjecture/thesis (this is my phrasing of the thesis in their paper):

The Random-Oracle Thesis: If a protocol has a proof of security in the random-oracle model, then it will be secure when instantiated with a “sufficiently crazy” hash function.

In the 17 years that passed since their paper, this thesis has been experimentally verified and mathematically refuted.

It was experimentally verified in the sense that no one has yet found an attack against the schemes suggested in their paper. There are also many other schemes that were proven secure in the random oracle model and so far have not been broken.

It was mathematically refuted by Canetti, Goldreich and Halevi in 1998, showing that there in fact exist protocols that are secure in the random oracle model, but can be attacked *no matter what* hash function collection is used. There were further results on this topic, see for example `http://www.cs.ut.ee/~lipmaa/crypto/link/rom/` for more info.

In class we will not make the random oracle thesis as part of our conjectures, since it’s not a precise mathematical conjecture. But if you use constructions based on random oracle in your homework you will get at least partial credit.