1 Cryptographic hash functions

Last time we saw a construction of message authentication codes (MACs) for fixed-length messages. We then showed a way to extend the construction in order to handle variable-length messages. There is another method for constructing variable-length MACs which is quite popular in practice, as it has very efficient implementations. It involves the use of another cryptographic primitive called a *cryptographic hash function*.

Definition 1. A collection of functions $\{h_S \colon \{0,1\}^* \to \{0,1\}^k\}$ is an (s,ε) *cryptographic hash function family* if for every circuit A of size at most s,

$$\Pr_S[A(S) = (x,x'),\ x \neq x',\ \text{and } h_S(x) = h_S(x')] \leq \varepsilon.$$

Since the domain of h_S is infinite and its range is finite, it is guaranteed that for every h_S a pair (x,x') such that $x \neq x'$ and $h_S(x) = h_S(x')$ always exists (in fact, there are infinitely many). The definition requires that finding such a pair is computationally intractable.

The parameter S is called the seed, and it plays a similar role as the key in pseudorandom functions. There is one difference: The key of the hash function is known to the adversary that is trying to break it, while a pseudorandom function key must be kept private.

Since it is intractable to find collisions in a hash, the hash $h_S(M)$ completely "determines" the message M for any computationally bounded party. So it seems reasonable that instead of tagging M to obtain an authentication, we could obtain almost the same effect by tagging $h_S(M)$.

Claim 2. *Suppose (Tag, Ver) is a MAC for message length k that is (s,ε) secure against chosen message attack and computable by a circuit of size t and $\{h_S\}$ as an (s,ε) secure cryptographic hash family where h_S is computable by a circuit of size t. Then the scheme*

$$Tag'((K,S),M) = Tag(K, h_S(M)) \qquad Ver'((K,S),(M,T)) = Ver(K, h_S(M), T)$$

is a variable-length MAC that is $(\Omega(s/t), 2\varepsilon)$ secure against chosen message attack.

The hash key S can in fact be made public in this scheme.

Proof. Suppose (Tag', Ver') is not $(\Omega(s/t), 2\varepsilon)$ secure and let A' be a circuit of size $\Omega(s/t)$ such that

$$\Pr[A'^{Tag'} \text{ produces a forgery}] > 2\varepsilon.$$

We use A' to construct two adversaries: A circuit $A^?$ that tries to forge (Tag, Ver) and a circuit C that tries to find a collision in h_S. Intuitively, every time $A'^{Tag'}$ produces a forgery, either its forged message collides with one of its queries under h_S, or else A^{Tag} produces a forgery.

More formally, consider the following circuits $A^?$ and C:

$A^?$: Choose a random S and simulate $A'^?$. When $A'^?$ wants to query its oracle on M, query your oracle on $h_S(M)$, and when A outputs a potential forgery (M, T), output $(h_S(M), T)$.

$C(S)$: Choose a random K and simulate $A'^{Tag'((K,S),\cdot)}$. Remember all the queries $M_1, \ldots, M_q$ made by A' and its output M. If $h_S(M_i) = h_S(M)$ for some i, output the pair (M_i, M).

Then $A^?$ and C are both of size at most s. We now argue that whenever $A'^{Tag'}$ produces a forgery, either one of the queries of $A^?$ collides with M under h_S in which case C outputs a collision, or A^{Tag} outputs a forgery. So we either obtain an adversary that (s, ε) breaks the collision resistance of $\{h_S\}$ or an adversary that (s, ε) breaks the security of (Tag, Ver), obtaining a contradiction.

So let's assume that $A'^{Tag'}$ makes queries $M_1, \ldots, M_q$ and outputs the forgery (M, T). By the definition of forgery, $M \neq M_i$ for all i. If $h_S(M_i) = h_S(M)$ for some i, then C outputs a collision for h_S. Otherwise, $h_S(M_i) \neq h_S(M)$ for all i. Since (M, T) is a forgery for (Tag', Ver'), we must have $Ver'(M, T) = Ver(h_S(M), T) = 1$, so the output $(h_S(M), T)$ of A^{Tag} will then be a forgery for (Tag, Ver). □

What is the advantage of this variable-length MAC over the one from last time? This is a simpler design, and moreover hash functions often have faster implementations that pseudorandom functions.

Cryptographic hashes in practice There are several practical construction of cryptographic hashes with very efficient implementations, which makes them very popular in applications. I don't know much about the rationale behind practical constructions, but if you are interested you can start by looking at the following descriptions of MD5 and SHA0, SHA1, SHA2, SHA3. Collisions in MD5, SHA0, and SHA1 have been found.

SHA3 was chosen a few weeks ago after a three-round competition by the US National Institute of Standards and Technology that was carried out from 2008 until 2012. It was an open competition where every entrant could present their design as well as point out weaknesses in their competitors.

One feature of practical hash functions is that they usually do not have a seed. Instead of a family of hash functions, the constructions give a single hash function. In our theoretical framework, an adversary can trivially find a collision for any specific hash function, as the collision can be hardwired into the adversary circuit. One way to reconcile the practical constructions with the definition is to think of an unseeded hash function as a sample from a family of hash functions to which the seed has been fixed once and for all. If the function is secure enough, then the a collision is hard in any foreseeable future, so it should not hurt to reuse the same seed (which is – remember – public) in all applications.

Attacks There is a fairly fundamental difference between the security of pseudorandom generators and the security of hash functions. Suppose you want to break a pseudorandom generator $G\colon \{0,1\}^k \to \{0,1\}^m$. If you have no additional information about how G works, you can try the following generic attack: Given a string $y \in \{0,1\}^m$, try all possible seeds $x \in B^k$ and output 1 if and only if $G(x) = y$ for some x. This attack can distinguish an output of G from a truly random

string with probability $1/2$, but takes time more than 2^k. However, without additional information on the structure of G, it is essentially the best possible attack.

In contrast, if we want to find a collision in a cryptographic hash family $h_S\colon \{0,1\}^* \to \{0,1\}^k$, the task can always be accomplished with constant probability in time $O(t2^{k/2})$, where t is the time it takes to evaluate h_S. So if we have a cryptographic family with 128 bits of output, it can always be broken in time about 2^{64}, which is large but not prohibitively so.

Do cryptographic hash functions exist? While many researchers believe that indeed they do exist, we do not know the answer to this question. In particular, if the answer is "yes", then it would follow that pseudorandom generators exist, and P $\neq$ NP. Can we then obtain cryptographic hash functions from pseudorandom generators, doing some construction like the one we used for pseudorandom functions? The answer is believed to be "no", and in particular we know for sure that a construction of the type that we used for pseudorandom functions cannot work to construct hash functions from pseudorandom generators. Making sense of this statement is way beyond the scope of this lecture.

There is another object called a *universal one-way hash function* (UOWHF) which provides weaker collision-resistance properties, but can be constructed from a pseudorandom generator. Known transformations of a pseudorandom generator into a UOWHF are, however, very inefficient.

2 The Merkle-Damgård transform

One mystifying aspect of the definition of hash function is that it can be used to hash messages of arbitrary length. We now show that it is in fact sufficient to have a variant that works for fixed-length messages. The construction that turns a fixed-length scheme into a variable-length one is called the Merkle-Damgård transform.

For simplicity let's suppose we have a cryptographic hash function family $h_S\colon \{0,1\}^{2k} \to \{0,1\}^k$ for some k. The Merkle-Damgård transform produces a new family $h'_S\colon \{0,1\}^* \to \{0,1\}^k$ defined by

$$h'_S(M_1 M_2 \dots M_\ell) = h_S(h_S(\dots h_S(h_S(0^k, M_1), M_2) \dots, M_\ell), \ell).$$

(We assume there is some canonical padding that makes the length of every message be a multiple of k.)

Claim 3. *Suppose $\{h_S\}$ is an (s,ε) cryptographic hash family for fixed-length messages, and h_S can be computed by a circuit of size t. Then $\{h'_S\}$ is an $(\Omega(s/t),\varepsilon)$ cryptographic hash family for variable-length messages.*

Proof. Suppose A is a circuit of size $s' = \Omega(st)$ that finds a collision in h'_S with probability ε. Now consider the following circuit B: On input S, B runs $A(S)$, and if $A(S)$ outputs a collision $(M_1 \dots M_\ell, M'_1 \dots M'_{\ell'})$ then B does as follows:

- If $\ell \neq \ell'$, then output the collision $((x,\ell),(x',\ell'))$, where $x = h_S(\dots h_S(h_S(0^k, M_1), M_2) \dots, M_\ell)$ and $x' = h_S(\dots h_S(h_S(0^k, M'_1), M'_2) \dots, M_{\ell'})$.

- If $\ell = \ell'$, recursively find the largest t for which

  $$h_S(\ldots h_S(h_S(0^k, M_1), M_2) \ldots, M_{t-1}) \neq h_S(\ldots h_S(h_S(0^k, M'_1), M'_2) \ldots, M'_{t-1}) \quad \text{or} \quad M_t \neq M'_t$$

  and output the collision

  $$(h_S(\ldots h_S(h_S(0^k, M_1), M_2) \ldots, M_{t-1}), M_t), (h_S(\ldots h_S(h_S(0^k, M'_1), M'_2) \ldots, M'_{t-1}), M'_t).$$

  Such a t must exist, for otherwise the output of A is not a collision.

B can be implemented by a circuit of size $O(s't) \leq s$, contradicting the assumption that h_S is a (s, ε) cryptographic hash family. □

3 One-way functions and one-way permutations

One-way functions are the most fundamental cryptographic primitive. Without them, most cryptographic tasks, including encryption and authentication, are impossible to achieve. With them, private key cryptography is possible – at least in principle. We already saw that private-key encryption and authentication can be realized from a pseudorandom generator. What is the advantage of using a one-way function?

The advantage is mainly theoretical: A one-way function is a much simpler object than a pseudorandom generator, and there are many examples of functions that we believe are one-way. In contrast, constructing a pseudorandom generator is quite a bit harder, and because the definition is stringent we may not have as much confidence in the construction. However there is a price to pay for this confidence: Current constructions of pseudorandom generators based on (general) one-way functions are extremely inefficient, so they have no practical value. However if we start with a one-way function with extra properties – for example the function is a permutation – then the resulting pseudorandom generators can be quite simple and useful. These techniques will also come up when we talk about public-key encryption in a couple of lectures.

Intuitively, a one-way function $f\colon \{0,1\}^n \to \{0,1\}^m$ is a function that is easy to compute, but hard to invert. "Easy to compute" means that on input x, $f(x)$ can be calculated fast. "Hard to invert" should mean that given $f(x)$, it is hard to recover x. But recovering x from $f(x)$ could be hard for trivial information-theoretic reasons. For example, if $f(x) = 0$ for all x, then certainly recovering x from $f(x)$ will be hard because there are too many choices for x, so we are unlikely to guess the correct one. A more reasonable inversion criterion is to find some x' so that $f(x) = f(x')$.

Definition 4. A function $f\colon \{0,1\}^n \to \{0,1\}^m$ is (s, ε) *one-way* if for every circuit A of size s,

$$\Pr_{x \sim \{0,1\}^n}[A(f(x)) = x' \text{ so that } f(x') = f(x)] \leq \varepsilon.$$

The definition of one-way function makes sense even when ε is very large, say $\varepsilon = 0.99$. In fact, it is possible to take a one-way function where ε is large and turn it into a new one-way function where ε is small.

We can also give an asymptotic definition: A family of functions $\{f_n\colon \{0,1\}^n \to \{0,1\}^{m(n)}\}$ is *one-way* if f_n is computable in time polynomial in n and for every polynomial p and sufficiently large n, f_n is $(p(n), 1/p(n))$ one-way.

There are quite a few examples of functions that are believed to be one-way. In the subset sum function, the inputs are integers $x_1, \ldots, x_k$ (represented by bit strings of length k) and a subset $S \subseteq \{1, \ldots, k\}$ and the function is defined as

$$SS_k(x_1, \ldots, x_k, S) = (x_1, \ldots, x_k, \sum_{i \in S} x_i).$$

Clearly this function is efficiently computable; however the inversion problem: "Given k (random) integers and a sum of some (random) subset of them, can you find the subset?" is believed to be computationally intractable.

Here is another example. Let M be a $3n \times n$ matrix and $f_M \colon \{0,1\}^n \to \{0,1\}^{3n}$ be the function $f_M(x, e) = Mx + e$, where $x \in \{0,1\}^n$ is a random (column) vector, $e \in \{0,1\}^{3n}$ is a random vector that has exactly $0.1n$ entries equal to one, and the multiplication and addition are modulo two. It is believed that when M is chosen at random, with high probability the function f_M is one-way. Conjectures have been made even for specific choices of M.

A *one-way permutation* is a one-way function $f \colon \{0,1\}^n \to \{0,1\}^n$ which is also a permutation of $\{0,1\}^n$. In a one-way permutation, for every y there is a unique x with $f(x) = y$, so the definition is simpler: The adversary now has to output $A(f(x)) = x$ with probability at least ε.

One-way permutations are harder to construct than one-way functions. The only construction of a candidate one-way permutation with domain and range $\{0,1\}^n$ that I know of is complicated. Here is a candidate construction of a one-way permutation over the set $\{1, \ldots, p-1\}$, where p is a suitable (large) prime number.

To introduce this function, we need a bit of algebra. For a prime number p, the multiplicative group $\mathbb{Z}_p^*$ is the set $\{1, \ldots, p-1\}$ together with the operation multiplication modulo p. This group is known to be cyclic, which means that there is a *generator* $g \in \mathbb{Z}_p^*$ so that $\mathbb{Z}_p^* = \{g, g^2, \ldots, g^{p-1} = 1\}$. Then the function

$$EXP_{g,p} \colon \mathbb{Z}_p^* \to \mathbb{Z}_p^* \qquad EXP_{g,p}(x) = g^x$$

is a permutation of $\mathbb{Z}_p^*$. If p is a random prime chosen from the range $\{1, \ldots, 2^n\}$, then a random $g \in \mathbb{Z}_p^*$ is a generator with high probability (as n grows), and it is believed that $EXP_{g,p}$ is one-way with high probability over the choice of a prime p and generator g.[1]

The fastest known algorithm for inverting $EXP_{g,p}$ runs in time $2^{n^{1/3}}$.

4 One-way functions and cryptography

While we have plenty of candidates for one-way functions, we do not know how to prove that they exist. If one-way functions exist, then P cannot be equal to NP, so the problem of proving the existence of one-way functions is considered very difficult.

We now make the case that one-way functions are necessary to have the kinds of cryptography we have seen so far. Let us begin by arguing that one-way functions are a prerequisite for pseudorandom generators – if pseudorandom generators exist, then so do one-way functions.

Claim 5. *If $G \colon \{0,1\}^n \to \{0,1\}^{2n}$ is an (s, ε) pseudorandom generator, then G is an $(s - t, \varepsilon + 2^{-n})$ one-way function, where t is the circuit size of G.*

[1] There are, however, bad choices of p that make $EXP_{g,p}$ easy to invert, for example primes of the form $2^m + 1$.

Proof. If G is not a one-way function, then there is a circuit A of size $s - t$ so that

$$\Pr_{x\sim\{0,1\}^n}[A(G(x)) = x' \text{ so that } G(x') = G(x)] > \varepsilon + 2^{-n}.$$

However, for a uniformly chosen $y \sim \{0,1\}^{2n}$

$$\Pr_{y\sim\{0,1\}^{2n}}[A(y) = x' \text{ so that } G(x') = y] \leq \Pr_{y\sim\{0,1\}^{2n}}[G(x') = y \text{ for some } x'] = \frac{2^n}{2^{2n}} = 2^{-n}.$$

Then the circuit $D(z) = G(A(z))$ can ε-distinguish the output of G from a random string. □

Similarly, if cryptographic hash functions exist, so do one-way functions: If $\{h_S \colon \{0,1\}^n \to \{0,1\}^{n/2}\}$ is an cryptographic hash family, then for most S the function h_S is a one-way function. We won't prove this, but here is some intuition: If $h_S(x)$ can be inverted, then one can produce a collision by choosing a random x and running the inverter on $h_S(x)$. This has probability at least $1/2$ of producing an input $x' \neq x$ that maps to $h_S(x)$.

There are arguments of a similar nature that even more complicated cryptographic objects, like message indistinguishable encryption and MACs secure against chosen message attack imply the existence of one-way functions.

This gives overwhelming evidence that one-way functions are necessary in order to have private-key cryptography. Are they also sufficient? There is a famous theorem of Håstad, Impagliazzo, Levin, and Luby, which gives a "yes, but..." answer to this question:

Theorem 6. *If (asymptotically secure) one-way functions exist, then (asymptotically secure) pseudorandom generators exist.*

The proof of this theorem gives a construction that allows us to turn any one-way function into a pseudorandom generator. This construction is, however, very inefficient: If the input to the one-way function is n-bits long, the corresponding pseudorandom generator takes inputs of size about n^{10} and produces only about n additional bits of randomness. Recently the construction has been improved by Haitner, Reingold, and Vadhan, and Vadhan and Zheng, but it still quite inefficient for practical purposes.

If we are given a *one-way permutation*, there is a much simpler and more efficient construction of pseudorandom generators, which we will show in the next lecture.