0368.4162: Introduction to Cryptography Ran Canetti

Lecture 7

15 December 2008 Fall 2008

Scribes: R. Kasher, O. Paneth

Topics for Today

- Collision resistant functions
- Message Authentication Codes (MAC)

1 Collision Resistant Functions

Intuitively, a collision resistant function (CRF) f is a function for which it is hard to find two inputs s.t. their images collide: If $x \neq x'$, then w.h.p. $f(x) \neq f(x')$. It is easy to see that any one-to-one function satisfies this requirement. However, we are interested in functions that are many-to-one, specifically, functions that map long inputs to short inputs, i.e., $|f(x)| < |x|$.
One immediate use for CRFs would be to verify the integrity of a large database D. Instead of storing the entire data in a safe place, we could use a CRF function $f : \{0,1\}^* \to \{0,1\}^n$ and save only $d \leftarrow f(D)$, which is relatively short. For any future suspicious version D' of our database, we can now simply test $d = f(D')$ and see if the version was compromised.

1.1 Formalization

We start with a native attempt:

Definition. A function f is $\epsilon(t)$-collision resistant (CR) if for any adversary A running in time t:

$$\Pr_{A}[A() = (x, x') \text{ s.t. } x \neq x' \wedge f(x) = f(x')] < \epsilon(t)$$

Of course this definition is impossible to achieve: If f has any collision, say for inputs x and x', we can always hard-wire them into A. The algorithm $A_{x,x'}$ is a fast algorithm that produces a collision for f.
Similarly to the definition of trapdoor permutation, we'll use a formulation of family of functions.

Definition 1. A family of functions $F = \{f_i : D_i \to R_i\}_{i \in I}$ is $\epsilon(t)$-CR if there exist algorithms G, E such that $G() \to i \in I$ is an index generation algorithm and $E(i, x \in D_i) \to f_i(x)$, and for any adversary A running in time t:

$$\Pr_{i \leftarrow G()}[A(i) = (x, x') \text{ s.t. } x \neq x' \wedge f(x) = f(x')] < \epsilon(t)$$

Definition 2. (Asymptotic CRF) An ensemble $\mathbb{F} = \{F_n\}_{n\in N}$, $F_n = \{f_i^n : D_i^n \to R_i^n\}_{i\in I_n}$ is CR if there exist algorithms G, E such that $G(1^n) \to i \in I$ is an index generation algorithm and $E(1^n, i, x \in D_i^n) \to f_i(x)$, and for every non-uniform polynomial adversary A:

$$\Pr_{i\leftarrow G(1^n)}[A(1^n, i) = (x, x') \text{ s.t. } x \neq x' \wedge f(x) = f(x')] < \nu(n)$$

where $\nu(n)$ is a negligible function of n.

Note:

- Unlike pseudorandom functions, we allow A to receive i as input, and still we require the hardness of finding collisions.
- If F is CRF and at least two-to-one, i.e. $\forall_{x,i}|f_i^{-1}(x)| \geq 2$, then F is one-way when viewed as a function from $I \times D$. Intuitively, if we can invert F, then for any x, i we can compute $f_i(x)$, find $y \in f_i^{-1}(x)$, and with probability $\geq \frac{1}{2}$ we have $y \neq x$ for which $f_i(y) = f_i(x)$ is a collision.
- There exists two-to-one OWF which is not CR: For example, a OWF that ignores the last bit of its input.
- There is no known construction of a CRF from OWF. Further, we have evidence that this is a hard task. It was proven that no such construction can use the OWF as a black-box.
  In fact, it is not even known that existence of TDPs implies existence of CRF.

We can construct CRF either directly or from known hard problems.

1.2 Construction of a CRF based on hardness of DLOG

We first introduce a new cryptographic primitive, claw free pairs of permutations.

1.2.1 Claw Free Pairs of Permutations

Pairs of permutations p_0, p_1 are claw-free if it is hard to find a "claw" x_0, x_1 s.t. $p_0(x_0) = p_1(x_1)$.

Definition 3. (Asymptotic CFP) An ensemble $\mathbb{P} = \{P_n\}_{n\in N}$, $P_n = \{P_{i,n}^0, P_{i,n}^1\}_{i\in I_n}$, $p_{i,n}^b : D_i \to D_i$ is claw-free if there exist algorithms G, E such that $G(1^n) \to i \in I$ is an index generation algorithm and $E(1^n, i, b, x \in D_i^n) \to P_i^b(x)$, and for every non-uniform polynomial adversary A:

$$\Pr_{i\leftarrow G(1^n)}[A(1^n, i) = (x_0, x_1) \text{ s.t. } P_{i,n}^0(x_0) = P_{i,n}^1(x_1)] < \nu(n)$$

where $\nu(n)$ is a negligible function of n.

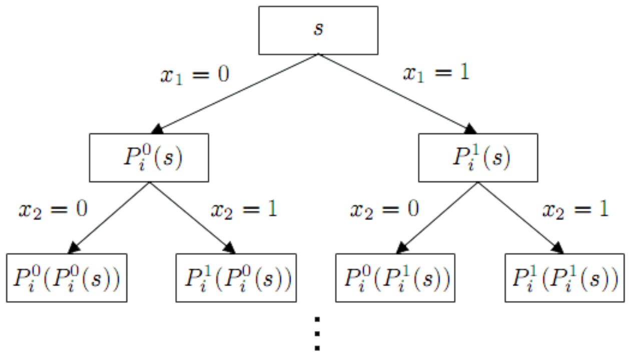

Figure 1: Constructing CRF from CFP

1.2.2 Construction of CFP from DLOG

Let $2^{n-1} \leq p < 2^n$ be an n-bit prime, g a generator of $\mathbb{Z}_p^*$, and $z \neq g$ an element of $\mathbb{Z}_p^*$. There exists a randomized polynomial algorithm $G(1^n) \to (p, g, z)$.
We define for $b \in \{0,1\}$, $P_{p,g,z}^b(x) = z^b \cdot g^x(p)$. The evaluation algorithm E is defined accordingly. Note that z^b is a constant, g is a generator, and so P^0, P^1 are permutations.
Finding a claw x_0, x_1 implies finding the discrete log of z:

$$g^{x_0} = z^0 \cdot g^{x_0} = P^0(x_0) = P^1(x_1) = z^1 \cdot g^{x_1}(p)$$

and so $z = g^{x_0 - x_1}(p)$.
Given an algorithm A that defeats the claw-freeness of P with probability ϵ, we can construct an algorithm A' that produces x on inputs p, g, z s.t. $g^x = z(p)$ w.p. ϵ:
If $g = z$, A' returns 1. Otherwise, runs A on the key (p, g, z) and receives (x_0, x_1). Returns $g^{x_0 - x_1}(p)$.
The above algorithms inverts DLOG with probability at least ϵ, since when $g \neq z$ then w.p. ϵ, (x_0, x_1) is a claw. Hence, finding a claw in P is at least as hard as DLOG.

1.2.3 Construction of CRF from CFP

Fix n. Let $P = \{P_i^0, P_i^1\}_{i \in I}$, $P_i^b : D_i \to D_i$ be a family of claw-free pair of permutations, with index generator G_P. We define a collision resistant family as follows (See figure 1):

- Index generator $G \to i, s$ where $i \leftarrow G_p$ and $s \leftarrow D_i$.
- $F = \{f_{i,s} : \{0,1\}^* \to D_i\}_{i \in I, s \in D_i}$
- $f_{i,s}(x_1, ..., x_m) = P_i^{x_m}(P_i^{x_{m-1}}(...(P_i^{x_1}(s))...))$

Claim. If P is CFP then F is CRF

Proof. Assume a polynomial algorithm A' that finds collisions in F with probability $\geq \epsilon$. We'll build algorithm A to find a claw in P_i^0, P_i^1 w.p. $\frac{\epsilon}{2}$:
We modify G slightly to return $i, s' = P_i^b(s)$ where $s \leftarrow D_i$ and b is a random bit. Since s is chosen uniformly at random and P_i^b is a permutation, then s' is also random.
A runs A' and obtains (x, x'). If $f_{i,s'}(x) \neq f_{i,s'}(x')$, i.e. A' failed to find a collision, abort. Else, A can find a claw as follows:
Assume w.l.o.g. that $m = |x| \geq |x'| = m'$. We have:

$$P_i^{x_m}(P_i^{x_{m-1}}(...(P_i^{x_1}(s'))...)) = P_i^{x'_{m'}}(P_i^{x'_{m'-1}}(...(P_i^{x'_1}(s'))...))$$

If $x_m \neq x'_{m'}$, then $P_i^{x_m}(a) = P_i^{x'_{m'}}(a')$ for some a, a' and we are done. Otherwise, since P_i^b is a permutation, we apply $(P_i^{x_m})^{-1}$ on both sides of the equation, and continue as above on x_{m-1} and $x'_{m'-1}$. If x' is not a suffix of x, then we will necessarily find a claw before "stripping" all P_i's. If it is, then at the end of the process we have found:

$$P_i^{x_{m-m'}}(...(P_i^{x_1}(s')...) = s'$$

Recall $s' = P_i^b(s)$, and with probability $\frac{1}{2}$, $b \neq x_{m-m'}$, and we have a claw $P_i^{x_{m-m'}}(...) = P_i^b(s)$.
In any case, the probability of finding a claw is at least $\frac{1}{2} \cdot \Pr_{A',i,s'}[f_{i,s'}(x) = f_{i,s'}(x')] \geq \frac{\epsilon}{2}$, which is non-negligible when ϵ is non-negligible. □

1.3 Direct Constructions of CRF

There's a number of direct constructions of collision resistant functions, often called "cryptographic hash functions", and these are the predominantly used ones in actual systems today.

1.3.1 Merkle-Damgård Construction

Essentially all of the cryptographic hash functions have the following structure, often called the Merkle-Damgård construction:
The function $H : \{0,1\}^* \rightarrow \{0,1\}^k$, $k = 128, 160, 192, ...$
The main element of H is a compression function $h : \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^k$, where l is called the block-length and is typically 512 bits.
H works as follows: The input message m is partitioned into l-bit blocks, $m_1, m_2, ...$
$C_0 = \text{IV}$ is a fixed value, $C_i = h(C_{i-1}, m_i)$ is the chaining value
H outputs the last C_i value after all the message blocks were processed.
Notes on the construction:

- Finding a collision in H can be done with expected $2^{k/2}$ queries by the birthday paradox, even when we know nothing of its structure.

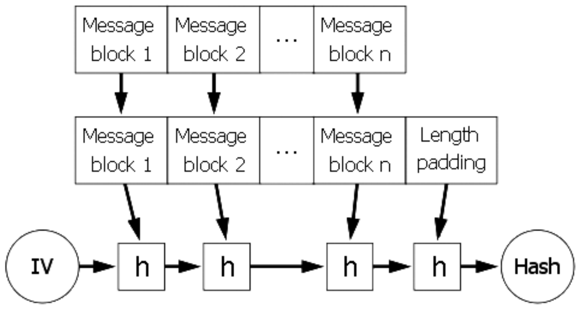

Figure 2: Merkle-Damgård chaining

- Intuitively, the strength of H depends on the collision resistance of h, although we cannot apply our notion of CR here because H, h are specific functions. Even if we regard h as a family of functions with key C_i, the reduction won't work.
- A query on a prefix $m_1, ..., m_i$ reveals the chaining value C_i. This weakens the cryptographic hash. (Although it is not obvious how this information can be used to find collisions in h, cryptographic hash functions have many other interesting properties and uses, some of which will be discussed later.) To prevent this m can be padded with its length (and again padded to fill the last block).

1.3.2 Construction of Compression Functions

Using Block Ciphers

We can think of several ways to construct compression functions using block ciphers:

- Feed the input block m_i as input and the key C_i as key
- Reverse the order of the input and the key
- Perform a XOR of the key (or the input) with the output

We don't have any concrete reason to prefer one model over the other.
Historically, the main drawback of all these constructions was that the block size for DES, 64bit, was too small, making birthday attacks feasible. Since AES has a block size of 128bit, using AES as a basis for such a construction is not unreasonable and has been proposed more than once.

Dedicated Compression Functions

Commonly used hash functions:

- MD4, MD5 proposed by Ron Rivest
- SHA-1, SHA-2 proposed by NIST and based on MD5
- The European standard RIPEMD

All have similar underlying structure. Following is a rough sketch of the compression function of MD5:

- The 512-bit input block is partitioned into 16 32-bit block $m_1, ..., m_{16}$
- The 128-bit chaining value C is partitioned into 4 32-bit blocks $A_0, ..., A_3$.
- There are four different functions $F_1, ..., F_4$, where each F_i takes 3 32-bit blocks as input and outputs one 32-bit block.
- There are 64 different 32-bit constants $t_1, ..., t_{64}$ determined via an "extremely non-linear" function, $\lfloor \sin(i) \cdot 2^{32} \rfloor$.
- There are 4 different shift values $s_1, ..., s_4$.
- The computation:
  for $i = 1..4$, $j = 1..4$, $k = 0..3$:
  $A_k \leftarrow F_i(A_{k+1}, A_{k+2}, A_{k+3}) \oplus m_{j(k+1)} \oplus t_{ij(k+1)} \ll s_j$
- There are 16 applications for each F_i where in each application a single block is updated as a function of the other 3 blocks.
- The functions F_i are chosen to be "balanced", i.e., they roughly preserve the number of 1s and 0s in the input, in order to prevent bias towards $\overrightarrow{0}$ or $\overrightarrow{1}$ after repeated applications.

There is not much understanding as to why any of these functions should work, and indeed eventually they break. Gaining more understanding on how to construct such functions is an interesting research challenge.

1.4 Cryptographic Hash Functions

We've mentioned several cryptographic hash functions, namely MD5, SHA-1, etc. These constructions are believed to have even stronger properties than collision resistance. In fact, these functions are believed to break any "non-trivial computational relation" (for example, finding the preimage of 0) between the inputs and the outputs.
This property leads to the use of cryptographic hash functions in many applications, such as message authentication codes and digital signatures. It is standard to use trapdoor permutations over hash functions in digital signature schemes. The security of this construction is based on various properties of the hash such as non-homomorphism and difficulty to inverse. We will study these mechanisms in detail later.

Finding rigorous ways to formalize the properties needed for applications, and then to realize them based on known assumptions, is a central goal in cryptography, and an active area of research.

Cryptographic Schemes and Protocols

Up until now in the course we dealt with formalizing and constructing basic cryptographic primitives: OWF, PRG, PRF, PRP, TDP and CRF. These primitives provide essential abstractions from specific underlying hard problems. These problems will be our toolkit in building cryptographic schemes, protocols and applications.
This second part of the course will deal with the cryptographic tasks themselves. Treating these tasks consists of three main steps:

1. Understanding the security requirement for the task and finding a way to formalize these requirements in a way that matches our intuitive idea and makes mathematical sense.
2. Constructing a scheme.
3. Proving the scheme satisfies the definition based on the properties of the underlying tools (under hardness assumptions).

2 Message Authentication Codes (MAC)

The first task we will introduce is that of message authentication codes.
Consider two parties A and B that wish to communicate over an open communication channel. This channel is untrusted: An adversary can inject, delete and modify messages at wish. The parties wish to be able to recognize when a received message is authentic, i.e., it was generated by the second party and not modified en route. To allow the parties to perform this task, we assume they have some shared random secret (key).

2.1 Formalization

A message authentication scheme consists of two interactive algorithms (ITM): *auth* in the sender side and *ver* in the receiver side, both initialized with the same shared (secret) key. Given a message m, $auth(k, m)$ generates an encoded version e. Since no secrecy is required, we can assume w.l.o.g. $e = (m, t)$ where t is called a "tag". Given an encoded message $e = (m, t)$, $ver(k, m, t)$ outputs accept/reject.
We require consistency: *ver* must accept any message encoded by *auth*.

Further, we require security: Any message not encoded by *auth* should be rejected by *ver*. This requirement, however, is not well defined since any encoded message can be potentially generated by an adversary. What we want is a definition that captures the notion of a "session" in the presence of an adversary that can interact with both parties, but still cannot correctly tag messages.

Definition. (First attempt) A scheme (*auth*, *ver*) is secure if for any polynomial adversary C, encoded messages $(m_1, t_1), ..., (m_l, t_l)$ such that $t_i = \text{auth}(k, m_i)$ where $l = \text{poly}(n)$, and message $m^* \neq m_i \forall_i$:

$$\Pr_{k \in \{0,1\}^n}[C(m_1, t_1, ..., m_l, t_l) = (m^*, t^*) \text{ s.t. } \text{ver}(k, m^*, t^*) = \text{acc}] < \nu(n)$$

Note that an adversary C that produces a tag t^* s.t. for some i, (m_i, t^*) is verified even though $t^* \neq t_i$, does not break the scheme. This is acceptable since we don't mind the adversary repeating a legitimate message.
Unfortunately, this definition fails to capture the essence of an interactive adversary, since the messages $m_1, ..., m_l$ are fixed in advance. Since an adversary might infer some security-relevant information from past messages and their tags, and craft his messages accordingly, we wish our definition to hold in this case as well. To this end, we introduce the notion of a game between *auth/ver* and our adversary.

Definition 4. MAC-GAME(auth, ver, C, n):
Initialize shared random key $k \in \{0,1\}^n$.
$C(1^n)$ interacts with sign and ver multiple times, queries $\text{auth}_k(m) \to (m, t)$ for any message m, and queries $\text{ver}_k(m', t')$ for the validity of any pair m', t'.
C wins if it produces any pair m^*, t^* such that $\text{auth}_k(m^*)$ was never queried yet $\text{ver}_k(m^*, t^*) = \text{accept}$.

We use the MAC-GAME to finally define a MAC scheme.

Definition 5. A MAC scheme is a pair of algorithms *auth*, *ver*, such that $\text{auth}_k(m) \to (m, t)$ is a tagging algorithm and $\text{ver}_k(m, t) = \{\text{accept/reject}\}$ is a verification algorithm. The algorithms must satisfy:
Consistency: For every k, m we have: $\text{ver}(k, \text{auth}(k, m)) = \text{accept}$
Security: For any polynomial adversary C,

$$\Pr_{k \in \{0,1\}^n, C}[C \text{ wins MAC-GAME}(\text{auth}, \text{ver}, C, n)] < \nu(n)$$

where $\nu(n)$ is a negligible function of n.

The above definition is a typical example of defining system security by the use of an experiment/interactive game. The game specifies the capabilities or types of interactions of an adversary, and the notion of success or what it means to "break" the security properties of the system.

2.2 Constructions

2.2.1 Secure MAC Scheme for a Single Message

We introduce a scheme that is secure for a single message against computationally unbounded adversaries. First, we need the following definitions:

Definition 6. A family of functions $H = \{h_k : \{0,1\}^n \rightarrow \{0,1\}^m\}_{k\in\{0,1\}^a}$ is called *pairwise independent (or, 2-universal)*[1] if for any $\alpha \neq \beta \in \{0,1\}^n$, $\gamma, \delta \in \{0,1\}^m$:

$$\Pr_{k\leftarrow\{0,1\}^a}[h_k(\alpha) = \gamma \wedge h_k(\beta) = \delta] = \frac{1}{2^{2m}}$$

Note that immediately from the definition it also follows that for any α, γ (and some $\beta \neq \alpha$):

$$\begin{aligned}\Pr_k[h_k(\alpha) = \gamma] &= \sum_\delta \Pr_k[h_k(\alpha) = \gamma \wedge h_k(\beta) = \delta] \\ &= 2^m \cdot 2^{-2m} = 2^{-m}\end{aligned}$$

Definition 7. An ensemble $H = \{H_n\}_{n\in N}$ of families of functions is pairwise independent with and range $m(n)$ and key range $a(n)$ if for each $n \in N$ the family $H_n = \{h_k : \{0,1\}^n \rightarrow \{0,1\}^{m(n)}\}_{k\in\{0,1\}^{a(n)}}$ is pairwise independent.

There are many constructions of pairwise independent function families with short keys (polynomial in n), such as the one we've seen in Problem Set 2.
Let $H = H_n$ be an ensemble of pairwise independent families. Consider the following scheme for n-bit messages:
Initialize random key $k \in \{0,1\}^{a(n)}$. Define $\text{auth}(k, m) = m, h_k(m)$, $\text{ver}(k, m, t) = \text{I}[t = h_k(m)]$, where $h_k \in H_n$.

Claim. The above scheme is secure for a single message.

Proof. Consistency is trivial.
Security follows directly from the definition of pairwise independence: An adversary C can query auth once for some α: $(\alpha, \gamma) \leftarrow \text{auth}_k(\alpha)$. To win C needs to find a pair β, δ s.t. $\text{ver}_k(\beta, \delta) = \text{accept}$. By the definition of auth, ver this implies:$h_k(\alpha) = \gamma \wedge h_k(\beta) = \delta$. Since k is chosen at random, the probability of this event is:

$$\begin{aligned}&\Pr_k[h_k(\beta) = \delta \mid h_k(\alpha) = \gamma] \\ =\ &\Pr_k[h_k(\beta) = \delta \wedge h_k(\alpha) = \gamma] / \Pr_k[h_k(\alpha) = \gamma] \\ =\ &2^{-2m} \cdot 2^m = 2^{-m}\end{aligned}$$

□

[1]Specifically, when $\gamma = \delta$, this implies $\Pr_{k\leftarrow\{0,1\}^a}[h_k(\alpha) = h_k(\beta)] = \frac{1}{2^m}$. This corresponds to the definition we have seen in Exercise 2, which is weaker than the one given here.

2.2.2 Secure MAC Scheme for an Unbounded Number of Messages

Fix n, the message bit length. Let h be a random function from $\{0,1\}^*$ to $\{0,1\}^n$, and define $\mathrm{auth}(h,m) = m, h(m)$, $\mathrm{ver}(h,m,t) = \mathrm{I}[t = h(m)]$.
It is clear the probability that any adversary C wins a MAC-GAME is $\frac{\mathrm{poly}(n)}{2^n}$, as for any m^*, t^*, $\Pr_h[h(m^*) = t^*] = \frac{1}{2^n}$, independently of any previous queries ($\neq m^*$) to h.
Unfortunately, such a scheme is not practical, since the representation of h (i.e., the key) is exponential in n. The next step would be to try a family of pseudorandom functions.
Let $F = \{f_k : \{0,1\}^* \to \{0,1\}^n\}_{k \in \{0,1\}^{a(n)}}$ be a family of PRF. We define the following MAC scheme:
Initialize random key $k \in \{0,1\}^{a(n)}$. Define $\mathrm{auth}(k,m) = m, f_k(m)$, $\mathrm{ver}(k,m,t) = \mathrm{I}[t = f_k(m)]$

Claim. The above scheme is a secure n-bit MAC scheme, assuming that the underlying function family is a PRF.

Proof. Consistency is trivial. Security: Assume towards contraction there exists a polynomial adversary C, s.t. C wins MAC-GAME(auth, ver, C, n) with non-negligible probability ϵ. We'll use C to build a polynomial adversary C' that distinguishes F from $\mathbb{F}_{*,n}$ w.p. ϵ:
C' has access to oracle f. C' simulates C. Every auth-query m_i made by C is answered with $m_i, f(m_i)$. A verify-query (m^*, t^*) is answered by $\mathrm{I}[f(m^*) = t^*]$. C' returns 1 if any verify-query is satisfied, that is, if C was able to produce a forgery.
When $f \in \mathbb{F}_{*,n}$, then for any m^*, t^* we have: $\Pr_f[C'^f = 1] = \frac{1}{2^n}$. When $f \in F$, then $\Pr_k[C'^{f_k} = 1] = \Pr_k[C \text{ wins MAC-GAME}] = \epsilon$
It follows immediately that F cannot be PRF, in contrary to the assumption. □