University of Maryland
CMSC456 — Introduction to Cryptography
Professor Jonathan Katz

Lecture 23

1 Collision-Resistant Hash Functions

We continue with our discussion from last time on the Hash-and-MAC paradigm for authentication of arbitrarily-long messages. Recall the definition of a collision-resistant hash function: Let $H : \{0,1\}^* \to \{0,1\}^n$ be a function taking arbitrary-length inputs and returning n-bit output. A pair (x, x') is a *collision in H* if $H(x) = H(x')$ but $x \neq x'$. We say that H is (t, ϵ)-collision resistant if, for all algorithms A running in time at most t, the probability that A will output a collision is less than ϵ.

We noted last time that collision-resistant hash functions cannot be constructed from arbitrary one-way functions or permutations. However, they can be constructed based on *specific* assumptions including the hardness of factoring, the RSA problem, or the hardness of computing discrete logarithms. Also, there are some practical constructions of functions which seem to be collision resistant; best-known among these are SHA-1 and MD5.

It is interesting to examine the minimum output length n necessary for a hash function to be (t, ϵ)-collision resistant. The following analysis will be informal, but can easily be made more exact. For *any* hash function $H : \{0,1\}^* \to \{0,1\}^n$ we can consider the following algorithm running in time roughly equal to t:

```
Pick random $x$ and compute $y = H(x)$
For $i = 1$ to $t$:
   Pick random $x_i$ different from $x$
   If $y = H(x_i)$ then output $(x, x_i)$ and stop
```

Note that if this algorithm outputs anything, then it outputs a collision. What is the probability that this algorithm finds a collision? Well, if the output of H is spread roughly uniformly in $\{0,1\}^n$, then picking a random x_i and computing $y_i = H(x_i)$ is roughly akin to picking random $y_i \in \{0,1\}^n$. For a particular i, then, the probability that $y_i = y$ is about 2^{-n}. And therefore, roughly speaking, the probability that at least *one* of $y_1, \ldots, y_t$ is equal to y is $O(t/2^n)$. What this means is that if we want H to be (t, ϵ)-collision resistant, we must at least have $t/2^n < \epsilon$, or $2^n > t/\epsilon$.

But in fact we can give an algorithm running in time t which is much better at finding collisions:

```
Pick distinct, random $x_1, \ldots, x_t$
Compute $y_1 = H(x_1), \ldots, y_t = H(x_t)$
If any of the $y_i$'s are equal, output the corresponding $x_i$'s
```

What is the probability of finding a collision now? If the output of H is again assumed to be roughly uniform in $\{0,1\}^n$, then picking random x_i and computing $y_i = H(x_i)$ is akin to picking random $y_i \in \{0,1\}^n$. For fixed i, j (with $i \neq j$), the probability that $y_i = y_j$ is then roughly 2^{-n}. Since there are $\binom{t}{2} = O(t^2)$ pairs $1 \leq i, j \leq t$, the means that the probability that $y_i = y_j$ for at least one pair is roughly $O(t^2/2^n)$. (This is again an application of the "birthday problem" that we have encountered before — we are picking t random elements from $\{0,1\}^n$ and want to find the probability that at least two of these are equal.) This means that if we want H to be (t, ϵ)-collision resistant, we must at least have $t^2/2^n < \epsilon$, or $2^n > t^2/\epsilon$.

Since $t \approx 2^{50}$ is currently considered feasible, this explain why the hash functions MD5 has 128-bit output and SHA-1 has 160-bit output. On the other hand, block ciphers can be secure with much shorter key-sizes (to prevent exhaustive search in time t, we need the key-length k to satisfy $t < 2^k$ or $k > \log t$ — compare this to the case of hash functions which need the output length n to satisfy $t^2 < 2^n$ or $n > 2 \log t$). This explain why the output length of a hash function is about twice the length of a typical key length for a block cipher (at least for the case of DES and other ciphers constructed before a few years ago).

Finally, we stress that these values for n are a necessary, but not sufficient, requirement. That is, it is certainly possible for a function to have "large" output length $n > 128$ and yet still not be collision resistant.

2 Hash-and-MAC

Our motivation for introducing collision-resistant hash functions was to use them to achieve message authentication for arbitrarily-long messages. Let $(\text{MAC}, \mathsf{Vrfy})$ be a (t, ϵ)-secure message authentication code for n-bit messages, and let $H : \{0,1\}^* \to \{0,1\}^n$ be a (t, ϵ')-collision resistant hash function. We can define a new message authentication code $(\text{MAC}', \mathsf{Vrfy}')$ for *arbitrary-length* messages as follows: the sender and receiver share a key s as in the original scheme. To authenticate a message $M \in \{0,1\}^*$, the sender computes $\text{MAC}'_s(M) = \text{MAC}_s(H(M))$. When the receiver gets a message/tag pair (M, tag), the receiver verifies the tag by computing $\mathsf{Vrfy}'_s(M, \mathsf{tag}) = \mathsf{Vrfy}_s(H(M), \mathsf{tag})$. We stress that H is part of the definition of the scheme, and is fixed and known to the adversary attacking the scheme. (Only the key s is unknown to the adversary.)

Theorem 1 *$(\text{MAC}', \mathsf{Vrfy}')$ is a $(t, \epsilon+\epsilon')$-secure message authentication scheme for arbitrary-length messages.*

Proof We sketch the proof here. Assume we have some adversary $\widetilde{A}$ attacking $(\text{MAC}', \mathsf{Vrfy}')$ and running in time t. We want to bound the success probability of this adversary (recall that this is the probability that $\widetilde{A}$ successfully forges a valid message/tag pair on a *new* message that was never explicitly authenticated by the sender). Letting $\widetilde{\mathcal{M}}$ denote the messages that $\widetilde{A}$ submits to its oracle MAC', we then want to bound:

$$\mathsf{Succ}_{\widetilde{A}} \overset{\text{def}}{=} \Pr[s \leftarrow \{0,1\}^k; (M, \mathsf{tag}) \leftarrow \widetilde{A}^{\text{MAC}'_s(\cdot)} : \mathsf{Vrfy}_s(M, \mathsf{tag}) = 1 \wedge M \notin \widetilde{\mathcal{M}}].$$

Let $\mathsf{Collision}$ denote the event that $H(M) = H(M')$ for some $M' \in \widetilde{\mathcal{M}}$. (I.e., this denotes the event that $\widetilde{A}$ was able to find a collision in H.) We then have:

$$\mathsf{Succ}_{\widetilde{A}} \;=\; \Pr[\widetilde{A} \text{ succeeds} \wedge \mathsf{Collision}] + \Pr[\widetilde{A} \text{ succeeds} \wedge \overline{\mathsf{Collision}}]$$

(where $\overline{\mathsf{Collision}}$ means that a collision does not occur).

We may now note the following:

1. When $\widetilde{A}$ succeeds and there *is* a collision, then we know that $H(M) = H(M')$ for some $M' \in \widetilde{\mathcal{M}}$. Furthermore, $M \neq M'$ since $\widetilde{A}$ cannot succeed if this is true (recall that $\widetilde{A}$ can only succeed if $M \notin \widetilde{\mathcal{M}}$). So, this means that $\widetilde{A}$ has found a collision in H (note that event $\mathsf{Collision}$ as defined above was not exactly the same as finding a collision in H — we need to additionally ensure that M and M' are different). Since H is (t, ϵ')-collision resistant, we have $\Pr[\widetilde{A} \text{ succeeds} \wedge \mathsf{Collision}] < \epsilon'$.

2. When $\widetilde{A}$ succeeds and there is *not* a collision, let $\mathcal{M} \overset{\text{def}}{=} \{H(M) \mid M \in \widetilde{\mathcal{M}}\}$ (i.e., this is the set of hashes of all elements in $\widetilde{\mathcal{M}}$). Since $\widetilde{A}$ succeeded, we know that $\mathsf{Vrfy}'_s(M, \mathsf{tag}) = 1$ and hence $\mathsf{Vrfy}_s(H(M), \mathsf{tag}) = 1$. On the other hand, since $\mathsf{Collision}$ did not occur we know that $H(M) \notin \mathcal{M}$. But then $\widetilde{A}$ has essentially forged a valid message/tag pair in scheme $(\text{MAC}, \mathsf{Vrfy})$ on a *new* message $H(M)$. Since we are given that $(\text{MAC}, \mathsf{Vrfy})$ is (t, ϵ)-secure, this means that $\Pr[\widetilde{A} \text{ succeeds} \wedge \overline{\mathsf{Collision}}] < \epsilon$.

Putting everything together shows that $\mathsf{Succ}_{\widetilde{A}} < \epsilon + \epsilon'$, proving the theorem. ■

3 Perfect Message Authentication

Thus far, all our schemes have been secure against a computationally-bounded adversary only, and our schemes only ensure that an adversary has a small (but non-zero) probability of success. In the case of encryption we were able to define (and achieve) a notion of perfect secrecy; can we do the same for message authentication?

What might a definition of perfect message authentication look like? Well, seemingly we would require that a computationally-unbounded algorithm cannot possibly forge a valid tag on a new message; that is: a message authentication code is perfectly secure if for all algorithms A we have:

$$\Pr[s \leftarrow \{0,1\}^k; (M, \mathsf{tag}) \leftarrow A^{\text{MAC}_s(\cdot)} : \mathsf{Vrfy}_s(M, \mathsf{tag}) = 1 \wedge M \notin \mathcal{M}] = 0.$$

However, this definition is *unachievable*. For *any* message authentication scheme, an adversary can always do the following: guess a random key $s' \in \{0,1\}^k$ and output $(M, \text{MAC}_{s'}(M))$ (and ask no queries to the MAC oracle). Note that if the adversary guesses correctly (and $s' = s$) then the adversary succeeds; thus, the adversary's probablity of success is at least $1/2^k$ (and hence not 0).

As a side point, we note that this sort of attack (i.e., guessing the key) does not contradict the fact that the one-time pad encryption scheme is perfectly secure. This is partly due to our definition of perfect secrecy in the case of encryption, but is also due to the fact that the goal of secrecy is fundamentally different from the goal of message authentication. In the case of message authentication, guessing the key incorrectly *does not diminish the adversary's total probability of success*; i.e., the adversary succeeds with probability at least $1/2^k$ regardless of what happens whenever it guesses the incorrect key. So:

$$\begin{aligned}\Pr[\mathsf{Succ}] &= \Pr[\mathsf{Succ} \wedge \text{correct guess}] + \Pr[\mathsf{Succ} \wedge \text{incorrect guess}] \\ &\geq \Pr[\mathsf{Succ} \wedge \text{correct guess}] = 2^{-k}.\end{aligned}$$