CSE 5852: Lecture 22

November 16, 2016

1 Last Class

Last class, we introduced the concept of a hash function. Roughly a hash function is a compressing function that is difficult to invert (even with all parameters of the function known). We introduced three versions of security for a hash function $H : \{0,1\}^\ell \times \{0,1\}^{2n} \to \{0,1\}^n$. (Recall, we already saw a strongly universal hash function, this only had security if the selection of the hash function was hidden from the adversary, but it provided information-theoretic security.)

Definition 1. *Let K be a set and defined for each $k \in K$ the function $H_k : D_k \to R_k$. Then $\{H_k\}_{k\in K}$ is a collection of hash functions if:*

1. *There is a PPT algorithm* Gen *that on input 1^n outputs $k \in K$.*
2. *$|D_k| > |R_k|$. So each function H_k actually does reduce its domain.*
3. *Given k and $m \in D_k$, $H_k(m)$ is efficiently computable.*

Collision resistance For all PPT $\mathcal{A}$, there exists a negligible function $\epsilon(n)$ such that for $k \leftarrow \mathsf{Gen}(1^n)$,

$$\Pr[(m_1, m_2) \leftarrow \mathcal{A}(1^n, k) \wedge m_1 \neq m_2 \wedge H_k(m_1) = H_k(m_2)] \leq \epsilon(n).$$

this probability is over the choice of k and any randomness used by $\mathcal{A}$.

Second-preimage resistance A hash function is second preimage resistant if given k and a uniform m it is infeasible for a PPT adversary to find some $m' \neq m$ such that $H_k(m') = H_k(m)$. Note: Look at the distinction between this and collision resistance. Here $\mathcal{A}$ doesn't get to choose the point m they are forced to find a preimage on a random point.

Preimage resistance A hash function is preimage resistant if given k and uniform y it is infeasible for a PPT $\mathcal{A}'$ to find a value m such that $H(x) = y$.

Theorem 1. *Collision resistance $\Rightarrow$ Second-preimage resistance $\Rightarrow$ preimage resistance*[1]*.*

[1]The second implication is only true in the setting where the hash function shrinks the input significantly.

2 Building a collision-resistant hash

Recall our definition of a collision-resistant hash function:

Definition 2. *Let K be a set and defined for each $k \in K$ the function $H_k : D_k \to R_k$. Then $\{H_k\}_{k\in K}$ is a collection of collision-resistant hash functions if:*

1. *There is a PPT algorithm* Gen *that on input 1^n outputs $k \in K$.*
2. *$|D_k| > |R_k|$. So each function H_k actually does reduce its domain.*
3. *Given k and $m \in D_k$, $H_k(m)$ is efficiently computable.*
4. *For all PPT $\mathcal{A}$, there exists a negligible function $\epsilon(n)$ such that for $k \leftarrow \mathsf{Gen}(1^n)$,*

$$\Pr[(m_1, m_2) \leftarrow \mathcal{A}(1^n, k) \wedge m_1 \neq m_2 \wedge H_k(m_1) = H_k(m_2)] \leq \epsilon(n).$$

this probability is over the choice of k and any randomness used by $\mathcal{A}$.

We'll now construct a collision resistant hash that is only slightly shrinking. Let p be an n-bit prime,. Let g be a generator of $\mathbb{Z}_p^*$. Construct a collision-resistant hash function (CRHF) as follows:

1. Pick $y \in \mathbb{Z}_p^*$ at random. (This is our key or index of the hash function.)
2. $h_{p,g,y}(x, b) = y^b \cdot g^x \mod p$ where $x \in \mathbb{Z}_p^*$ and $b \in \{0, 1\}$.

Note that the size of the domain of h $2(p-1)$ and the size of its range is $p-1$. So it compresses by one bit.

Theorem 2. *If the discrete logarithm problem is hard, the above hash function is collision-resistant.*

Proof. Suppose $\mathcal{A}$ breaks collision resistance of h. That is, there exists a polynomial $p(n)$ such that

$$\Pr_y[(x, b), (x', b') \leftarrow \mathcal{A}(1^n, g, p, y) \wedge y^b g^x \equiv y^{b'} g^{x'}] > \frac{1}{p(n)}.$$

First note that it is necessary that $b \neq b'$. Since g is a generator if $b = b'$ then $y^b g^x \equiv y^b g^{x'}$ implies that $x \equiv x' \mod p-1$. So this is not a collision. Thus, we know that $b \neq b'$. Assume that $b = 0, b' = 1$. This means that $\mathcal{A}$ finds a pair where

$$g^x = h(x, b) = h(x', b') = g^{x'} y \mod p.$$

Put differently $g^{x-x'} \equiv y \mod p$. This means we can use these values to compute the discrete log of y. Here is our new adversary (for the discrete log problem) $\mathcal{A}'$:

1. Input $1^n, p, g, y$ where $y = g^z$ for some unknown z.
2. Run $((x, b), (x', b')) \leftarrow \mathcal{A}$.
3. Verify that $\mathcal{A}$ has output a collision (if not output a random value $z \in \mathbb{Z}_p^*$)
4. If $b = 0$ output $x - x' \mod p-1$ otherwise output $x' - x \mod p-1$.

□

2.1 Enhancing compression

Like pseudorandom generators we can make collision resistant hash functions more compressing. Suppose we have a family of hash functions $H : \{0,1\}^{\ell} \times \{0,1\}^{n+\alpha} \to \{0,1\}^{n}$. We can then design a collision-resistant hash function for an arbitrary length message $H'\{0,1\}^{\ell} \times \{0,1\}* \to \{0,1\}^{n}$.

We split m into x bit blocks $m_1, ..., m_t$ where m_t contains the length of m, define $H'(m) = H(m_t \circ H(m_{t-1} \cdots H(m_2 \circ H(m_1 \circ IV))))$ where IV is some fixed value, for example 0^n.

3 The Random Oracle Model

The basics of the model are this: assume that H is a public oracle and that anyone is capable of asking for an evaluation of the hash function. If H has not been queried on that point before it provides a random output otherwise it looks up the previous returned value. Importantly, no one has access to the internals of this function, they can only call it. It is important to note that no function H has the chance of actually providing this behavior. One of the main reasons is that in order to evaluate a hash function parties need access to it and can understand how it works. As a silly example, the hash function can be run with its own code as input, this is not possible for the random oracle. There are numerous results showing it is not possible for a hash function to actually fulfill what is needed of a random oracle [CGH04].

Under this model, security is proved using the following two steps:

1. First, a scheme is designed and proved secure in the random-oracle model. Other cryptographic assumptions can also be used in the scheme.

2. When we want to implement the scheme the random oracle is *instantiated* with a cryptographic hash function. At each point when a part is supposed to query $H(x)$ it instead queries $H_i(x)$.

The model is used because it provides a way of reasoning about security.

Properties of the random oracle model 1) If x has not been queried to H, the value of $H(x)$ is a uniform random value.

When we do proofs using the random oracle it allows us to cheat a little bit. When we are engaging in a reduction with $\mathcal{A}$ that uses the random oracle we get to simulate the random oracle because it doesn't exist. This is where its important that $\mathcal{A}$ doesn't have access to the code and had to use an outside algorithm. If the actual description of H was known to $\mathcal{A}$ we Our algorithm $\mathcal{A}'$ is able to create and fake answers to the random oracle.

2) If $\mathcal{A}$ queries x to H, the reduction *sees the query* and learns the value x. (This does not contradict the fact that queries to the oracle are private. Queries are private in the model but we are using $\mathcal{A}$ as a subroutine within a reduction that is simulating the random oracle for $\mathcal{A}$. (Extractability)

3) The reduction can *set* the value of $H(x)$ to a value of its it choice as long as the value is uniform distributed. (Programmability)

Reproducing Cryptographic Objects in the Random Oracle Model Some basic constructions in the random oracle model:

1. Can be used in place of a pseudorandom generator. Its outputs are pseudorandom as long as the input point is not known. That is,

$$|\Pr[\mathcal{A}^{H(\cdot)}(y) = 1] - \Pr[\mathcal{A}^{H(\cdot)}(H(x)) = 1]| \leq \epsilon(n).$$

   Note that x doesn't even have to be fully random, it just needs to be difficult to guess.

2. Can be used as a collision resistant hash function. The probability that any two points will collide is the size of the output domain. So the probability of finding a collision is roughly $q^2/2^n$ where q is the number of queries to the random oracle.

3. Can be used to construct a pseudorandom function. Define $F_k(x) \stackrel{def}{=} H(k||x)$.

Security of RSA signatures in the random oracle model For our purposes we will use the model to show security of the RSA construction [BR96].

$\mathsf{Gen}(1^n)$:

1. Choose two n-bit prime integers p, q using a special algorithm.
2. Compute $N = p \cdot q$ also compute e, d as in Theorem 1.
3. Output $vk = e, N$, $sk = d$.

$\mathsf{Sign}(d, N, m)$:

1. Query $y \leftarrow H(m)$.
2. Compute $\sigma = y^d \mod N$.

$\mathsf{Vfy}(e, N, m, \sigma)$:

1. Compute $y' = \sigma^e \mod N$.
2. Query $y \leftarrow H(m)$.
3. Output 1 if and only if $y' = y$.

Theorem 3. *If the RSA problem is hard relative to the choice of p, q and H is modeled as a random oracle, then the above construction is secure.*

Proof. We will go through the basic structure of the proof. Let $\mathcal{A}$ be an adversary that is able to forge with probability $1/p(n)$. We can assume that if $\mathcal{A}$ requests a signature on a message m or outputs a forgery m, σ then it has queried m to H. (Why can we assume this?)

Recall what our goal is (as an adversary trying to break the RSA problem). We are trying to build $\mathcal{A}'$ that receives the following inputs:

1. N, e, y as input.

2. Choose a uniform $j \in \{1, ..., q\}$ where q is the maximum number of queries to the random oracle.

3. Run $\mathcal{A}$ on input $pk = (N, e)$.

4. Initialize an empty table. An entry in the table indicate (m_i, σ_i, y_i). This entry indicates that $\mathcal{A}'$ has set $H(m_i) = y_i$ and $\sigma_i^e = y_i \mod N$.

5. When $\mathcal{A}$ makes its i-th random-oracle query $H(m_i)$, answer as follows:

   - If $i = y$ return y as the answer to the query.
   - Else choose uniform $\sigma_i \in \mathbb{Z}_N^*$, compute $y_i = \sigma_i^e \mod N$. return y_i as the answer and store (m_i, σ_i, y_i).

6. $\mathcal{A}$ when requests a signature on m, let i be such that $m = m_i$ (recall by assumption when they query m they have previously asked for a random oracle query on it. Answer as follows:

   - If $i = j$ then quit.
   - Otherwise, there is an entry in the table, return σ_i as the answer.

7. When $\mathcal{A}$ finishes it outputs (m, σ). If $m = m_j$ and $\sigma^e = y \mod N$ then output σ.

□

References

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 399–416. Springer, 1996.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.