CS 7810 Foundations of Cryptography October 16, 2017

Lecture 11: Hash Functions and Random Oracle Model

Lecturer: Daniel Wichs *Scribe: Akshar Varma*

1 Topic Covered

- Definition of Hash Functions
- Merkle-Damgaård Theorem
- Merkle Trees
- Random Oracle Model

2 Collision-Resistant Hash Functions

A collision-resistant hash functions (CRHF) is a function that "shrinks" a long message to a short output called the *digest* of the message. Intuitively, we want to ensure that the even though the digest is short it uniquely identifies the message. This may seem contradictory since many messages must be hashed to the same digest. However, the security property that we will require, called *collision resistance* says that no efficient adversary can come up with a *collision* consisting of two different inputs that both get mapped to the same output.

2.1 Naive definition

A natural way of defining a CRHF is as follows. A family of functions $H : \{0,1\}^{\ell(n)} \to \{0,1\}^n$ with $\ell(n) > n$ are collision-resistant hash functions iff they satisfy the following properties.

- $H(x)$ can be computed in polynomial time.
- For all PPT adversaries $\mathcal{A}$,

$$\mathsf{Pr}[H(x) = H(x') \wedge x \neq x' : (x, x') \leftarrow A(1^n)] = \mathsf{negl}(n).$$

While this definition intuitively captures the idea of a CRHF it is never going to be satisfied if we allow for non-uniform adversaries. When non-uniform adversaries are allowed there is a trivial adversary that simply hard codes a collision for each possible n and always successfullly outputs $x \neq x'$ such that $H(x) = H(x')$ (all without doing any computation).

2.2 Better definition

To remove the possibility of such trivial adversaries, we modify our definition to include a "seed". A family of functions $H(s,x) : \{0,1\}^n \times \{0,1\}^{\ell(n)} \to \{0,1\}^n$ with $\ell(n) > n$ are collision-resistant hash functions iff they satisfy the following properties.

- $H_s(x) := H(s,x)$ can be computed in polynomial time.
- For all PPT adversaries $\mathcal{A}$,

$$\mathsf{Pr}[H_s(x) = H_s(x') \wedge x \neq x' : s \leftarrow \{0,1\}^n, (x,x') \leftarrow A(1^n, s)] = \mathsf{negl}(n).$$

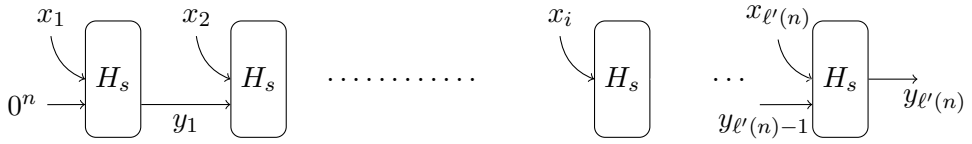
The definition says that when a uniformly random public seed is used, no adversary can find a collision with non-negligible probability. Apart from capturing the intuition of collision-resistant hash functions it also removes the possibility of trivial adversaries. This seed is primarily for having a sound definition and in practice unseeded hash functions are used because no one knows how to come up with the hard coded collisions and hence the trivial adversary.

3 Merkle-Damgaård Theorem

Recall that for PRGs we had a theorem stating that if we have a PRG which extends the input by one bit, then we can use that to create PRGs that extends the input by any polynomial number of bits. Similarly, we now show that if we can compress the input by one bit, then we can compress it by any polynomial number of bits.

Theorem 1 *If H_s is a family of collision-resistant hash functions with $\ell(n) = n+1$, then we can construct a family of collision-resistant hash functions H'_s with $\ell'(n) = \mathsf{poly}(n)$.*

Proof: *We provide a construction for H'_s similar to the construction for the chained PRG construction. Let $X = (x_1, x_2, \ldots, x_n)$ be the input bits to the function H'_s. We define $H'_s(X) = y_{\ell'(n)}$ via the following figure.*



We show that if $\exists X \neq X'$ such that $H'_s(X) = H'_s(X')$, then we can use that to find a collision for H_s. Let $y_1, \ldots, y_{\ell'(n)}$ be the values created during the computation of $H_s(X)$ and $y'_1, \ldots, y'_{\ell'(n)}$ be the values created during the computation $H_s(X')$. We know that $y_{\ell'(n)} = y'_{\ell'(n)}$ because $H'_s(X) = H'_s(X')$. Search backwards for the largest i such that $(x_i, y_{i-1}) \neq (x'_i, y'_{i-1})$ (this i exists because $X \neq X'$). This pair implies that $H_s(x_i, y_{i-1}) = H_s(x'_i, y'_{i-1}) \because y_i = y'_i$. This provides a collision for H_s and is a contradiction to H_s being a CRHF. □

A natural question to ask now is whether the same construction is sufficient for variable length inputs? The answer is no. Consider a CRHF for which $H_s(0^{n+1}) = 0^n$, we simply prepend 0^n to our input and get collisions, $H'_s(X) = H'_s(0^n||X)$.

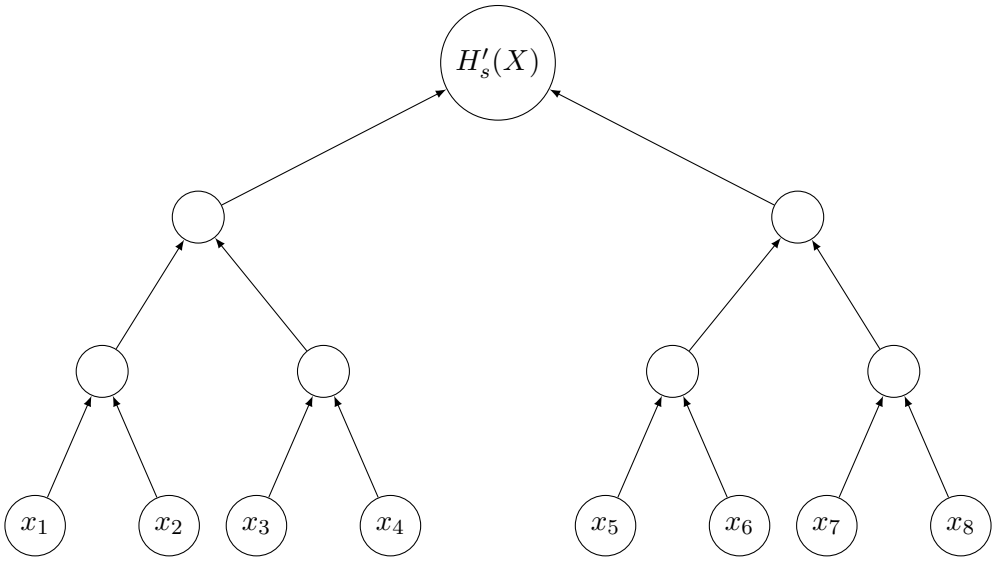
The place where our proof fails is when we search for "the largest i". We simply "fall off" the chain of H_s corresponding to $H'_s(X)$ when searching for this i. We can avoid this problem by defining $H''_s(X) = H'_s(X|| < \ell'(n) >)$ where $< \ell'(n) >$ is the binary representation of the length of the input X. Now the same proof idea will work. Either we find a collision in the first n bits ($\because \ell'(n) \leq 2^n$) from the right or $|X| = |X'|$ and the earlier proof works is sufficient.

4 Merkle Trees

While the Merkle-Damgaård construction allows us to go from 1 bit compression to any sized compression, we now look at a different construction which is more useful in some scenarios.

Theorem 2 *If H_s is a family of collision-resistant hash functions with $\ell(n) = 2n$, then we can construct a family of collision-resistant hash functions H'_s with $\ell'(n) = n \cdot 2^i$ for any $i > 0$.*

Proof: *Let $X = (x_1, x_2, \ldots, x_{2^i})$ be the input bits split into blocks of n bits each. We compute the final digest by creating a binary tree (of depth i) with each n bit block as a leaf, and every internal node being the hash of the concatenation of its two children ($x = H_s(x_l, x_r)$). Thus, the root will be a digest of the required size and would be dependent on all the bits of X. We illustrate the tree structure for $i = 3$.*



The proof that this construction gives us a CRHF is quite similar to the proof of the Merkle-Damgaård construction. Using a downwards search starting at the root in the two

trees, we would see that a parent and the corresponding children would provide a collision for H_s if there is a collision for H'_s. □

4.1 Merkle-Damgaård construction vs. Merkle Trees

We have seen two ways to construct CRHFs with more compression power starting with less powerful CRHFs. Both of these have their advantages and disadvantages and we now look at some of those.

Parallelizability/Streaming: Merkle Trees naturally provide parallelizability in their construction unlike the Merkle-Damgaård construction. On the other hand, this parallelizability comes with the disadvantage of having to store a lot of intermediate states before reaching the final digest value unlike for Merkle-Damgaård where we only need to store one hashed value at any given time. However, if the data is received in a streaming manner then the Merkle-Damgaård construction is much more efficient in terms of storage space required.

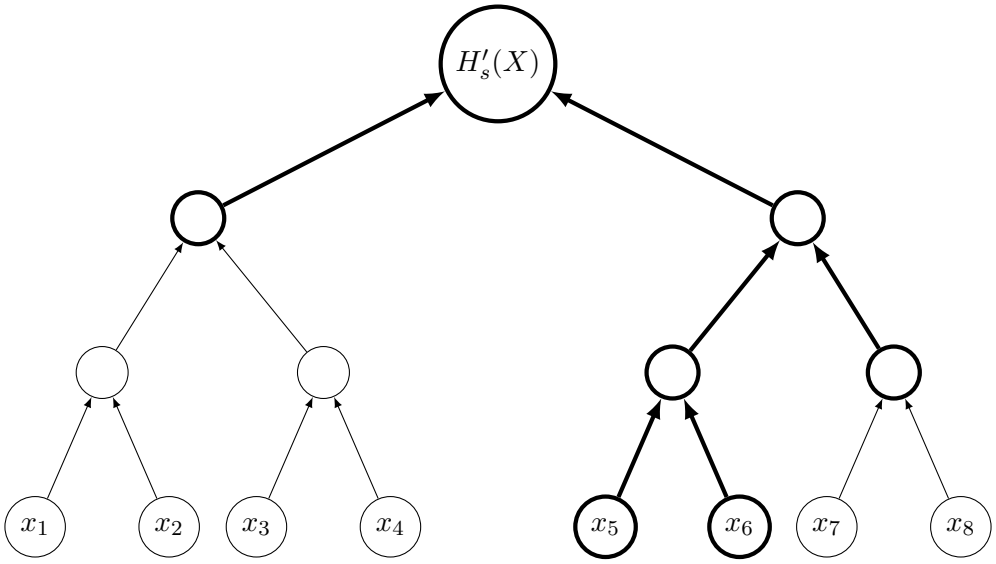
Updation/Appending: If the input value changes slightly then the Merkle Tree construction simply updates the internal nodes that do get changed which needs $O(i)$ computations and hence smaller than $O(\ell'(n))$ computations that the Merkle-Damgaård construction would need. This is a simple storage vs. computation trade-off with the Merkle Tree having to store all intermediate nodes and the Merkle-Damgaård construction having to perform all computations from scratch. Appending to the input is slightly more complicated for the Merkle Tree as new nodes need to be created and balancing of the tree may also come into play if one needs to keep the updation efficient. For the Merkle-Damgaård construction storing the state just before the $< \ell'(n) >$ part might be sufficient.

Small segment of large dataset: A scenario where the Merkle Tree is better equipped compared to the Merkle-Damgaård construction is when a small part of the message needs to be verified. Suppose Alice has stored a very large amount of data with Bob and Charlie wants to see a small segment of this data (say x_i). If Bob sends Charlie the whole data and the corresponding hash, then that results in a large amount of data transfer (especially if Charlie is only concerned with x_i and doesn't care about the rest of x). However, if Bob uses the Merkle Tree method of computing the hash then he can simply send Charlie the requested data segment along with the hashes for all the sibling nodes along the path from the leaf for x_i to the root (as shown in bold in the following figure). This allows us to solve the problem while avoiding high communication overhead which would not be possible with the Merkle-Damgaård construction due to its serial nature.

5 Random Oracle Model

The CRHFs that are used in practice seem to have many other cryptographic properties apart from "collision-resistance". We try to capture this using the Random Oracle model by considering an idealized hash function.

We formally define a Random Oracle model as a model in which all parties (including adversaries) have oracle access to a consistent, uniformly random function $\mathsf{RO} : \{0,1\}^* \to \{0,1\}^n$. This oracle can be thought of as choosing a random output y on being queried with



a value x and remembering its choice. When two people query the function with the same x, they both receive the same y value.

We define cryptographic systems in this model the same as earlier, except both the algorithm and adversary are provided oracle access to $\mathsf{RO}(\cdot)$. The standard security and correctness requirements are carried forward into this setting as we will see in the following examples. We assume that $|X| = n$ while analyzing security of the cryptosystems we define.

5.1 OWFs

We define a OWF in the Random Oracle model as follows:

$$f^{\mathsf{RO}(\cdot)}(X) = \mathsf{RO}(X)$$

Since RO is a truly random function, there is no way for an adversary to invert it except by brute-force queries. The probability that one query of the adversary succeeds is bounded by $\frac{1}{2^n}$. If the adversary $\mathcal{A}$ makes T queries to RO, then we can bound the success probability of $\mathcal{A}$ by $\frac{T}{2^n}$.

5.2 PRGs

We define a PRG in the Random Oracle model as follows:

$$G^{\mathsf{RO}(\cdot)}(X) = \mathsf{RO}(X||0)||\mathsf{RO}(X||1)$$

This is a PRG as it takes in n bits of input and returns $2n$ bits of output. The only way for an adversary to distinguish between $G(X)$ and U_{2n} is to query RO on $X||0$ or $X||1$, which can happen with probability at most $\frac{1}{2^n}$ (adversary needs to guess X). Similar to the OWF case, the success probability is bounded by $\frac{T}{2^n}$ if the adversary is allowed to make T queries.

5.3 PRFs

We define a PRF in the Random Oracle model as follows:

$$F_K^{\mathsf{RO}(\cdot)}(X) = \mathsf{RO}(K||X)$$

This has the same security as the PRG definition if $|K| = n$ (adversary has to guess K).

5.4 CRHF

We define a CRHF in the Random Oracle model as follows:

$$H^{\mathsf{RO}(\cdot)}(X) = \mathsf{RO}(X)$$

We claim that this has security $\frac{T^2}{2^n}$ if the adversary makes T queries. The proof follows by noting that the probability of the i^{th} and j^{th} queries colliding is $\frac{1}{2^n}$ and hence with T queries, the probability of finding a collision is bounded by $\frac{T^2}{2^n}$ (using the union bound). This is also known as the birthday bound that is seen in the birthday paradox.

$$\begin{aligned}
&\mathsf{Pr}[(X \neq X') \wedge (\mathsf{RO}(X) = \mathsf{RO}(X')) : (X, X') \leftarrow \mathcal{A}^{\mathsf{RO}(\cdot)}(1^n)] \\
&\leq \mathsf{Pr}[\exists i, j \in [T], i \neq j, \mathsf{RO}(X_i) = \mathsf{RO}(X_j)] \\
&\leq \sum_{i,j} \mathsf{Pr}[\mathsf{RO}(X_i) = \mathsf{RO}(X_j)] \leq \frac{T^2}{2^n}
\end{aligned}$$

5.5 Random Oracles in real life

The motivation of coming up with the Random Oracle model was to try and capture the extra properties that CRHFs seemed to show in real life, in a formal and rigourous manner. When we try to go back from the theoretical models to practice, we lose this rigour.

In real life there are no random oracles RO, and the cryptographic primitives that we construct in this model cannot be directly used. What is done in practice is to simply replace the RO with a CRHF H_s and use the same constructions as in the Random Oracle model. Of course H_s is not a truly random function, and hence none of the nice properties that we can prove regarding RO necessarily hold when we use H_s. When shifting from theory to practice, mathematical rigour and proofs are let go and one just hopes that it works out. The guarantees regarding security get modified (in a hand-wavy manner), with the number of queries T getting replaced with the running time of H_s. While this is not a rigourous manner of arguing about the security of cryptosystems it seems to work in general and thus gets used.