CSE 5852: Lecture 23

November 28, 2016

1 Last Class

Last class, we introduced a collision-resistant hash function based on the discrete logarithm problem. We also introduced the random oracle model and showed security of RSA hash-and-sign in this model. In this class, we will show that collision-resistant hash functions are sufficient to build a signature scheme.

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.*
4. ***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 $\mathcal{A}$.

2 Signature Schemes Based on Hashing

Recall the Lamport signature scheme that is secure for a single message. Let $H_k : \{0,1\}^{2n} \to \{0,1\}^n$ be a family of hash functions. Then to sign messages of length ℓ we construct a secret key which is 2ℓ random values $x_{i,j}$ in $\{0,1\}^{2n}$. Our public key is the hash of each of these values. So $y_{i,j} = H_k(x_{i,j})$ along with the key k.

We will show this scheme is secure for a single message. We need to show that an adversary who breaks this scheme is able to break collision resistance of the hash function.

Proof. Our main observation is this if an attacker has a signature on a message m and wishes to forge on some message $m \neq m'$ there is at least one position i such that $m_i \neq m'_i$. Call m'_i the value b. So forging a signature requires finding a preimage of h of the value $y_{i,b}$.

Let $\mathcal{A}$ be a PPT adversary that is trying to break security of the signature scheme. We assume that $\mathcal{A}$ always requests a signature of some message before

outputting a forgery. We denote by m the requested message and the final forgery as m', σ'. Note that whenever $\mathcal{A}$ outputs a forgery it inverts the hash function at some location.

We build the following algorithm $\mathcal{A}'$ that attempts to break collision resistance of $\mathcal{H}$.

1. Receive input k.
2. Choose random values $x_{i,j}$ and compute $y_{i,j} = H_k(x_{i,j})$.
3. Provide $k, y_{i,j}$ to $\mathcal{A}$
4. When $\mathcal{A}$ requests a signature of m provide the corresponding $x_{i,j}$.
5. When $\mathcal{A}$ outputs a forgery m', σ' find the location i where $m_i \neq m'_i$. Denote $m'_i = b$.
6. If $x_{i,b}$ is not equal to the σ_i then output x_i, σ'_i as a forgery.
7. Else output $\perp$.

Each value $x_{i,j}$ is chosen uniformly at random. The adversary has no information about which $x_{i,j}$ was used to produce the $y_{i,j}$. Thus, as long as each $y_{i,j}$ has at least two possible preimages, the preimage σ'_i returned by $\mathcal{A}$ will differ from the known preimage of $\mathcal{A}'$. The argument that most $y_{i,j}$ will have multiple preimages is similar to the proof that collision resistance implies second preimage resistance. The basic idea is that at most $2^n - 1$ values of x have a unique y such that $y = H_k(x)$. Thus, with probability $\frac{2^n-1}{2^{2n}}$ an $x_{i,j}$ is chosen from the set *Good* with multiple preimages. By the union bound the probability that all $x_{i,j}$ are chosen from the set *Good* is at least $2\ell\frac{1}{2^n}$. Furthermore, since this probability is so small the adversaries success in this set cannot drop by too much. Together these facts allow us to conclude that $\mathcal{A}$ has a nonnegligible chance of inverting when there are multiple preimages. In this case, the adversary will choose a different preimage than $\mathcal{A}'$ with probability at least one half representing a collision. Overall this occurs with an inverse polynomial probability. □

So Lamport's scheme allows us to sign a single message at the cost of a long private key. Furthermore, the length of this message can be arbitrary at the cost of increasing the length of the private (and public) keys.

So our next signature scheme is going to require state. The basic idea is that the holder of the secret key will additionally keep some state value s. At the conclusion of each signing operation the state will be updated from s_i to s_{i+1}. Note that the public key is still fixed and does not require updating. We can obviously construct a t secure signature scheme by generating t independent public keys and only signing with each private key a single time. This creates a public key whose length is linear in t.

2.1 Merkle Trees

We could also condense down the length of the public key using something called a Merkle tree. The idea is to publish a single hash value as the public key. Draw a Merkle tree. This has the effect of creating a constant length public key at the cost of a logarithmic length signature.

Construction 1. *Let* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *be a one-time secure signature scheme with verification keys in* $\{0,1\}^n$*. Let* H *be a family of collision resistant hash functions from* $\{0,1\}^{2n} \to \{0,1\}^n$*. Then the following is a* t *secure signature scheme:*

1. Gen' *run* t *copies of* Gen *to create* vk_i, sk_i*. Create Merkle tree of* vk_i *with root value* h^**. Publish* h^* *as the verification key. Set value* $i = 1$*.*
2. $\mathsf{Sign}'(sk, i, m)$*. Run* $\sigma_i = \mathsf{Sign}(sk_i, m)$*. Publish* σ_i *along with the corresponding Merkle tree nodes as the signature.*
3. $\mathsf{Vfy}'(h^*, h_i, m, \sigma)$*. Verify the Merkle tree then run* $\mathsf{Vfy}(vk_i, m, \sigma)$ *for the one-time scheme.*

This scheme requires knowledge of the number of messages ahead of time. We're now going to remove this restriction.

2.2 Chain Based Signatures

As before assume we have a one-time secure signature scheme. We generate a vk_1, sk_1 and have vk_1 be the public verification key. When we need to sign instead of just signing the message we also create a new pair vk_2, sk_2. So our signature looks like $\mathsf{Sign}'(sk_i, m)$ does the following:

1. run $(sk_{i+1}, vk_{i+1} \leftarrow \mathsf{Gen}(1^n)$.
2. Generate $\sigma_i \leftarrow \mathsf{Sign}(sk_i, vk_{i+1}, m)$.
3. Let $\sigma_i' = \sigma_{i-1}', vk_i, \sigma_i$.

So note the tradeoff that we've made. We have a constant length public key and signing only requires a constant number of operations but it is necessary to store all previously generated public keys. Indeed, verification takes linear time in the number of messages. Note this also requires all signed messages to always be presented when verifying a message this might be a problem but a good encryption scheme can be added as well to take care of this problem.

Second note, we need the one-time secure signature scheme to be capable of signing messages that are longer than its own public key. This was not the case for the Lamport signature scheme. However, before applying our one-time secure scheme we can first use a collision resistant hash function to decrease the length of the input.

2.3 Combining Tree and Chain

The main problem with the Tree based approach was that we needed to known the number of messages to be signed ahead of time. The chain based approach doesn't scale well because it has linear size signatures. Lets try and combine the two approaches. Lets assume we are talking about signing messages of length n. When we need to sign a message m it generates a path down to this message creating new signing and verification keys along the way. If these have already been generated it uses the previously generated ones. Then at the end it uses the signature key for the corresponding message.

Technique	vk length	σ len	# messages bounded
Multi keys	$O(t)$	$O(1)$	Yes
Merkle tree	$O(1)$	$O(\log t)$	Yes
Chain	$O(1)$	$O(t)$	No
Tree Signing	$O(1)$	$O(\log n)$	No

Table 1: Comparison of stateful signature techniques based on collision-resistant hashes.

2.4 Removing state

The reason the previous scheme needed state is that the leaves of the tree had to be consistent with the generated public keys from previous schemes. We couldn't regenerated these internal leaves for each signature. If we did regenerate we would be using a one-time scheme to sign multiple different messages which may not be secure. The way to replace this is to go back to two observations:

1. Pseudorandomness is just as good as randomness.
2. We know a mechanism to consistently generate a large amount of pseudo-random data.

So the idea is to replace all of the randomness needed to generate the signing keys and the signature algorithms with pseudorandom values derived from a pseudorandom function. The key to this function will be the overall secret key and will allow regeneration of all needed signing keys. Some care is needed to ensure that distinct calls to the PRF are used for each node in the tree and each signing algorithm.

3 How does identity actually work?

Explain the CA structure and how things are loaded on to individual computers.

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.