1 One-Time Secure Digital Signature Schemes

In the last lecture, we developed a digital signature scheme for message in \( \{0,1\}^n \) and proved it to be one-time secure. We also showed how to construct a family of Collision-Resistant Hash Functions (CRH) and claimed that they would be useful in constructing one-time secure digital signature schemes for messages in \( \{0,1\}^* \). The following theorem makes this claim more formally.

**Theorem 1** If there exists a CRH from \( \{0,1\}^* \rightarrow \{0,1\}^n \) and there exists a one-way function (OWF), then there exists a one-time secure digital signature scheme for \( \{0,1\}^* \).

**Proof.** We will only provide a sketch of the proof here.

Let \( \{h_i\}_{i \in I} \) be a CRH with sampling function \( \text{Gen}_{\text{CRH}}(1^n) \), and let \( (\text{Gen}_{\text{Sig}}, \text{Sign}, \text{Ver}) \) be a one-time secure digital signature scheme for \( \{0,1\}^n \) (as shown in Lecture 19, such a digital signature scheme exists by the assumption that a OWF exists).

We define a new one-time secure digital signature scheme \( (\text{Gen}', \text{Sign}', \text{Ver}') \) for \( \{0,1\}^* \) by

- Generator \( \text{Gen}'(1^n) \) produces its public and private keys from the generator \( \text{Gen}_{\text{Sig}} \) and sampling function \( \text{Gen}_{\text{CRH}} \) by setting \( \text{pk}, \text{sk} \leftarrow \text{Gen}_{\text{Sig}}(1^n) \) and \( i \leftarrow \text{Gen}_{\text{CRH}}(1^n) \) and returning \( \text{pk}' = (\text{pk}, i) \) and \( \text{sk}' = (\text{sk}, i) \).

- Signing function \( \text{Sign}' \) operates on a message \( m \) by calling the original signing function \( \text{Sign} \) on the hash of the message: \( \text{Sign}'_{\text{sk}}(m) = \text{Sign}_{\text{sk}}(h_i(m)) \).

- Verification function \( \text{Ver}' \) then simply verifies the signature on the hash of its message: \( \text{Ver}'_{\text{pk}}(m, \sigma) = \text{Ver}_{\text{pk}}(h_i(m), \sigma) \)

Digital signature schemes that operate on the hash of a message are said to be in the hash-and-sign paradigm.

Now suppose that there is a PPT adversary \( A \) that breaks \( (\text{Gen}', \text{Sign}', \text{Ver}') \) with non-negligible probability \( \epsilon \) after only one oracle call \( m \) to \( \text{Sign}' \). To break this digital signature scheme, \( A \) must output \( m' \neq m \) and \( \sigma' \) such that \( \text{Ver}'_{\text{pk}}(m', \sigma') = \text{accept} \) (so \( \text{Ver}_{\text{pk}}(h_i(m'), \sigma') = \text{accept} \)). There are only two possible cases:
1. $h(m) = h(m')$.
   In this case, $A$ found a collision $(m, m')$ in $h_i$, which is known to be hard, since $h_i$ is a member of a CRH.

2. $A$ never made any oracle calls, or $h(m) \neq h(m')$.
   Either way, in this case, $A$ obtained a signature $\sigma'$ using $(\text{Gen, Sign, Ver})$ to a new message $h(m')$. But obtaining such a signature violates the assumption that $(\text{Gen, Sign, Ver})$ is a one-time secure digital signature scheme.

To make this argument more formal, turn the two cases above into two adversaries $B$ and $C$. Adversary $B$ tries to invert a hash function from the CRH, and $C$ tries to break the digital signature scheme.

$B(1^n, i)$ operates as follows to find a collision for $h_i$.

- Generate keys $pk, sk \leftarrow \text{Gen}_\text{Sig}(1^n)$
- Call $A$ to get $m', \sigma' \leftarrow A^{\text{Sign}_{sk}(h_i(\cdot))}(1^n, (pk, i))$.
- Output $m, m'$ where $m$ is the query made by $A$ (if $A$ made no query, then abort).

$C^{\text{Sign}_{sk}(\cdot)}(1^n, pk)$ operates as follows to break the one-time security of $(\text{Gen, Sign, Ver})$.

- Generate index $i \leftarrow \text{Gen}_\text{CRH}(1^n)$
- Call $A$ to get $m', \sigma' \leftarrow A(1^n, (pk, i))$
  - When $A$ calls $\text{Sign}_{sk, i}'(m)$, query signing oracle $\text{Sign}_{sk}(h_i(m))$
- Output $h_i(m'), \sigma'$.

So, if $A$ succeeds with non-negligible probability, then either $B$ or $C$ must succeed with non-negligible probability.

2 Signing Many Messages

Now that we have extended one-time signatures on $\{0, 1\}^n$ to operate on $\{0, 1\}^*$, we turn to increasing the number of messages that can be signed. The main idea is to generate new keys for each new message to be signed. Then we can still use our one-time secure digital signature scheme $(\text{Gen, Sign, Ver})$. The disadvantage is that the signer must keep state to know which key to use and what to include in a given signature.

We start with a pair $(pk_0, sk_0) \leftarrow \text{Gen}(1^n)$. To sign the first message $m_1$, we perform the following steps:
• Generate a new key pair for the next message: \( \text{pk}_1, \text{sk}_1 \leftarrow \text{Gen}(1^n) \)

• Create signature \( \sigma_1 = \text{Sign}_{\text{sk}_0}(m_1 \| \text{pk}_1) \) on the concatenation of message \( m_1 \) and new public key \( \text{pk}_1 \).

• Output \( \sigma'_1 = (1, \sigma_1, m_1, \text{pk}_1) \)

Thus, each signature attests to the next public key. Similarly, to sign second message \( m_2 \), we generate \( \text{pk}_2, \text{sk}_2 \leftarrow \text{Gen}(1^n) \), set \( \sigma_2 = \text{Sign}_{\text{sk}_1}(m_2 \| \text{pk}_2) \), and output \( \sigma'_2 = (2, \sigma_2, \sigma'_1, m_2, \text{pk}_2) \). Notice that we need to include \( \sigma'_1 \) (the previous signature) to show that the previous public key is correct. These signatures satisfy many-message security, but the signer must keep state, and signature size grows linearly in the number of signatures ever performed by the signer. Proving that this digital signature scheme is many-message secure is left as an exercise for the student.

### 2.1 Improving the Construction

A simple way to improve this many-message secure digital signature scheme is to attest to two new key pairs instead of one at each step. This new construction builds a balanced binary tree of depth \( n \) of key pairs, where each node and leaf in the tree is associated with one public-private key pair \( \text{pk}, \text{sk} \), and each non-leaf node public key is used to attest to its two child nodes. Each of the \( 2^n \) leaf nodes can be used to attest to a message. Such a digital signature algorithm can perform up to \( 2^n \) signatures with signature size \( n \) (the size follows because a signature using a particular key pair \( \text{pk}_i, \text{sk}_i \) must provide signatures attesting to each key pair on the path from \( \text{pk}_i, \text{sk}_i \) to the root). The tree looks as follows.

![Binary Tree Diagram](image-url)
To sign the first message \( m \), the signer generates and stores \( \text{pk}_0, \text{sk}_0, \text{pk}_{00}, \text{sk}_{00}, \ldots, \text{pk}_{0^n}, \text{sk}_{0^n} \) along with all of their siblings in the tree. Then \( \text{pk}_0 \) and \( \text{pk}_1 \) are signed with \( \text{sk} \), producing signature \( \sigma_0 \), \( \text{pk}_{00} \) and \( \text{pk}_{01} \) are signed with \( \text{sk}_0 \), producing signature \( \sigma_1 \), and so on. Finally, the signer returns \( \sigma = (\text{pk}, \sigma_0, \text{pk}_0, \sigma_1, \text{pk}_{00}, \ldots, \sigma_{n-1}, \text{pk}_{0^n}, \text{Sign}_{\text{sk}_{0^n}}(m) \) as a signature for \( m \). The verification function \( \text{Ver} \) then uses \( \text{pk} \) to check that \( \sigma_0 \) attests to \( \text{pk}_0 \), uses \( \text{pk}_0 \) to check that \( \sigma_1 \) attests to \( \text{pk}_{00} \), and so on up to \( \text{pk}_{0^n} \), which is used to check that \( \text{Sign}_{\text{sk}_{0^n}}(m) \) is a correct signature for \( m \).

For an arbitrary message, the next unused leaf node in the tree is chosen, and any needed signatures attesting to the path from that leaf to the root are generated (some of these signatures will have been generated previously). Then the leaf node key is used to sign the message in the same manner as above.

Proving that this scheme is many-message secure is left as an exercise for the student. The key idea is that fact that \((\text{Gen}, \text{Sign}, \text{Ver})\) is one-time secure, and each signature is only used once. Thus, forging a signature in this scheme requires creating a second signature.

For all its theoretical value, however, this many-message secure digital signature scheme still requires the signer to keep a significant amount of state. The state kept by the signer is

- The number of messages signed

To remove this requirement, we will assume that messages consist of at most \( n \) bits. Then, instead of using the leaf nodes as key pairs in increasing order, use the \( n \)-bit representation of \( m \) to decide which leaf to use. That is, use \( \text{pk}_m, \text{sk}_m \) to sign \( m \).

- All previously generated keys

- All previously generated signatures (for the authentication paths to the root)

We can remove the requirement that the signer remember the previous keys and previous signatures if we have a pseudo-random function to regenerate all of this information on demand. In particular, we generate a public key \( \text{pk} \) and secret key \( \text{sk}' \). The secret key, in addition to containing the secret key \( \text{sk} \) corresponding to \( \text{pk} \), also contains two seeds \( s_1 \) and \( s_2 \) for two pseudo-random functions \( f \) and \( g \). We then generate \( \text{pk}_i \) and \( \text{sk}_i \) for node \( i \) by using \( f_{s_1}(i) \) as the randomness in the generation algorithm \( \text{Gen}(1^n) \). Similarly, we generate any needed randomness for the signing algorithm on message \( m \) with \( g_{s_2}(m) \). Then we can regenerate any path through the tree on demand without maintaining any of the tree as state at the signer.