This lecture is dedicated to constructions of digital signature schemes. Assuming the existence of CRHF (or UOWHF), we know that it is sufficient to construct signature schemes for fixed-length messages (at least as long as the security parameter). However, we will see that such constructions are not easy. We start with an educational construction, based on trapdoor permutations. This construction will be insecure. In fact, it’s insecurity will suggest — incorrectly — that it might be impossible to construct a secure signature scheme. Luckily, this conclusion, known as “signature paradox”, will turn out to be wrong. We will then construct one-time signature schemes, which will allow one to sign at most one message. Then, we give Naor-Yung construction, which shows how to extend a one-time signature into a regular-signature. The construction will crucially use the “hash-then-sign” paradigm, but will be somewhat inefficient. Finally, we will move to more practical secure signature schemes. For that, we will introduce the random oracle model, and show how to revive the originally doomed “trapdoor signature” scheme in this model. This scheme is called full domain hash, and is extensively used in practice.

1 Examples and problems

The purpose of this section is twofold. First, we show how signature schemes that are “somewhat secure” can be designed using trapdoor permutations. Second, by showing that these schemes fail to meet the security standards we set for signatures, we give evidence that designing secure signatures is hard, if possible at all. We also present a convincing but thankfully fallacious “proof” of the non-existence of secure signature schemes.

1.1 Examples: trapdoor signature schemes

The subject of this section is a way of building signature schemes from trapdoor permutations.

**RSA Signature.** The idea behind this scheme is the following. We use the inverse of the RSA function to produce the signature $\sigma = \text{RSA}^{-1}(m)$ from the message $m$, and those interested in checking it just compute $\text{RSA}(\sigma)$ and compare it with $m$. More precisely (using the notation in the definition of PKS):

a) $SK = (p, q, d)$ and $VK = (n, e)$ where $p, q$ are random $k$-bit primes, $n = pq$ is the RSA modulus, $e \in \mathbb{Z}^{\ast}_{\varphi(n)}$ is the RSA exponent and $d = e^{-1} \mod \varphi(n)$.

b) $\text{Sign}_{SK}(m) = m^d \mod n$ (where $m \in \mathbb{Z}^{\ast}_{n}$).

c) $\text{Ver}_{VK}(\sigma) = [\sigma^e = m \mod n]$ (where $\sigma \in \mathbb{Z}^{\ast}_{n}$).
A VK-only attack cannot result in universal forgery with non-negligible probability under the RSA assumption though the following reasoning. Because a successful universal forgery would enable one to find RSA^{-1}(m) of any (and, thus, of a random) m with probability ε, contrary to the RSA assumption. This implies that under the RSA assumption this scheme is universally unforgeable against VK-only attack.

Rabin Signature. The idea used in the previous scheme is again used, only substituting the modular squaring function for RSA. That is,

a) $SK = (p, q, g, h)$ and $VK = n$ where $p, q$ are random $k$-bit primes, $g$ generates $\mathbb{Z}_p$, $h$ generates $\mathbb{Z}_q$ and $n = pq$ is the modulus.

b) $Sign_{SK}(m) = m^{\sqrt{n}} \in \mathbb{Z}_n^*$ (where $m \in \mathbb{Z}_n^*$ and any of the four square roots will do).

c) $Ver_{VK}(\sigma) = [\sigma^2 = m]$ ($\sigma \in \mathbb{Z}_n^*$).

The same argument given for RSA proves that this PKS is universally unforgeable against VK-only attacks under the factoring assumption, and therefore even more believable than RSA.

Signature based on any Trapdoor Function. It turns out that the above constructions can be generalized for an arbitrary trapdoor function $f$ with trapdoor information $t$ (technically, a family of trapdoor functions with an efficient generation algorithm for $(f, t)$)

a) $SK = (f, t)$ and $VK = (f)$.

b) $Sign_{SK}(m) = f^{-1}(m)$, computed using $t$.

c) $Ver_{VK}(\sigma) = [f(\sigma) = m]$.

The previous two examples can be easily put into that framework. We now present a general theorem on the unforgeability of trapdoor signatures.

**Theorem 1 ((In)Security of Trapdoor Signatures against VK-only attack)** If $f$ is a trapdoor family, then the corresponding trapdoor signature scheme is universally unforgeable against VK-only attack, but existentially forgeable against VK-only attack.

**Proof:** We prove the first assertion by contradiction. Suppose that $f$ is a trapdoor but the corresponding signature scheme does not have the desired property. That means that there exists some PPT $A$ such that with non-negligible probability $\varepsilon = \varepsilon(k)$

$$\Pr(\text{Sign}_{SK}(m) = \sigma \mid (SK, VK) \leftarrow \text{Gen}(1^k), m \leftarrow \mathcal{M}_k, \sigma \leftarrow A(m)) = \varepsilon$$

Since $\text{Sign}_{SK} = f^{-1}$ and $\text{Ver}_{VK} = f$, we can rewrite this as $\Pr(m = f(\sigma) \mid m \leftarrow \mathcal{M}_k, \sigma \leftarrow A(m)) = \varepsilon$. Moreover, since $f$ is a permutation, if $x \in \mathcal{M}_k$ is random then $m = f(x)$ is random, and therefore $\Pr(f(x) = f(\sigma) \mid x \leftarrow \mathcal{M}_k, \sigma \leftarrow A(f(x))) = \varepsilon$. Therefore, $A$ inverts $f$ with non-negligible probability, which contradicts the fact that $f$ is a trapdoor permutation.

For the second assertion, notice that a PPT adversary $B$ who on input $VK$ outputs $(f(\sigma), \sigma)$ (for some “signature” $\sigma$ he picks) always succeeds in his attack (i.e., $\sigma$ is a signature of “message” $f(\sigma)$).
Also, it turns out that in some special cases a trapdoor signature may be universally forgeable under the CMA. Consider, for example, trapdoor families for which for all \( k \), \( \mathcal{M}_k \) is a group and \( f: \mathcal{M}_k \to \mathcal{M}_k \) is a group homomorphism. In that case, it is easy for an adversary to find out \( \sigma = \text{Sign}_{SK}(m) \) for any \( m \in \mathcal{M}_k \) without a direct oracle query for \( \text{Sign}_{SK}(m) \). Indeed, for \( m = 1 \) (i.e. the identity element) we have that \( \sigma = 1 \), and for \( m \neq 1 \) it suffices to pick any \( m_1 \in \mathcal{M}_k \) that is not 1 or \( m \), then compute \( m_2 = m^{-1} \) (which also differs from the identity and from \( m \)). The adversary can find out \( \sigma_1 = \text{Sign}_{SK}(m_1) \) and \( \sigma_2 = \text{Sign}_{SK}(m_2) \) by oracle calls and compute \( \sigma = \sigma_1 \sigma_2 \). This shows that such trapdoor families, RSA and Rabin among them, are universally forgeable against chosen-message attacks.

It should be clear by now that the design of “secure” signature schemes is a difficult problem and at this point one might be inclined to believe that it has no solution.

1.2 A “Signature Paradox”

Those who subscribe to the pessimism expressed in the final statement of the previous subsection will not have a hard time accepting the following “theorem”, which would be a far-reaching generalization of the second statement of the theorem on unforgeability of trapdoor signatures, were it only true.

**Theorem 2 (The fallacious “Signature Paradox”)** If \( \text{SIG} = (\text{Gen}, \text{Sign}, \text{Ver}) \) is universally unforgeable against \( VK \)-only attack, then it is existentially forgeable against chosen message attacks.

**Corollary 1 (“No secure signatures”)** There do not exist signature schemes that are existentially unforgeable against CMA.

**Proof:** The corollary follows from the “theorem” because existential unforgeability against CMA implies universal unforgeability against \( VK \)-only attack. Therefore, a scheme cannot satisfy the former property without also satisfying the latter, and the theorem says that universal unforgeability against \( VK \)-only attack implies existential forgery against CMA.

As for the “theorem”, its “proof” goes as follows. The way one proves that some \( \text{SIG} = (\text{Gen}, \text{Sign}, \text{Ver}) \) is universally unforgeable under \( VK \)-only attack is to prove that that property is equivalent to some “hardness” assumption on a problem \( X \) that we believe to be true. On one hand, one shows that the “hard” problem \( X \) is such that its solution yields an algorithm to break \( \text{SIG} \) in polynomial time (for instance, if one can factor \( n \) then one can take modular square roots mod \( n \) and break the Rabin signature scheme with that modulus). On the other hand, one assumes the existence of a PPT \( A \) that on input \( (VK, m) \) provides a valid signature \( \sigma \) for any \( m \) and shows (to get a contradiction) that that would imply that there exists a PPT (denoted by \( B \)) that would use \( A \) as a “black box” (i.e. as an oracle) to break the hardness assumption and solve that given instance of \( X \) (in the Rabin case, if such an \( A \) exists, then one can take square roots, and using that square root algorithm as a black box one can factor the modulus). Thus, the existence of such “universal” \( B \) proves that the presumed \( A \) does not exist. Notice, however, that \( B \) by itself is well-defined: given a good \( A \), \( B \) would break \( X \).
But then, if the adversary can perform CMA, he can use this $B = B(VK)$, only that, in place of using “black box calls” to $A$ to get $\sigma' = A(VK,m)$ (for a given $m$), he uses a CMA instead to get $\sigma = \text{Sign}_{SK}(m)$. Then $B$ can solve $X$ in polynomial time and use that solution to break the signature scheme SIG. Therefore, CMA allows the adversary to substitute attacks to the sender for calls to $A$. For instance, in Rabin’s case, that would mean that with a CMA with some well-chosen messages, one would have enough information to factor the modulus $n$ and then break the signature scheme.

What is wrong with that “proof”? Well, here is one mistake in it. When $B$ uses $A$ as an oracle in the proof by contradiction of the universal unforgeability of SIG, $B$ can make any query he wants to $A$. Among other things, he might make a query of the form $A(VK',m)$, where $VK' \neq VK$. That is, he might try different verification keys, maybe because if one knows $\text{Sign}_{VK'}(m)$ for several different $VK'$ one can make a very good guess of what the solution to $X$ is (who knows?). On the other hand, when we try to turn $B$ into a CMA adversary, we must then commit to a single verification key: the one that is randomly chosen by the sender. That is, whenever $B$ launches a CMA on the sender, he (the sender) always uses the $VK$ that he himself has chosen; equivalently (in a more formal language), all oracle calls that $B$ can make to $\text{Sign}_{SK}$ must use the same $SK$ that is in the output of Gen, and it is quite improbable that a $VK' \neq VK$ will work with $SK$ in the right way. Therefore, the last paragraph of that “proof” is wrong.

2 One-time secure signature schemes

The falsehood of the “signature paradox” in the last section leaves us with some hope that it might be possible to build secure signature schemes after all. But given all the difficulties we have faced so far, we’d better try to do it one step at a time. Our plan, which we begin to put into practice in this section, is: to build a rather simple scheme that is secure as long as the adversary can only make one CMA, and to show how to get a secure scheme out of it.

2.1 Lamport’s scheme for one bit

One-way functions (OWF) provide a nice way of signing one bit, which we present below.

**Definition 1** [Lamport’s scheme for 1-bit messages] Using the notation from the definition of PKS, and letting $f : \{0,1\}^* \rightarrow \{0,1\}^*$ be a fixed OWF, we define Lamport’s scheme for one-bit messages by:

a) $SK = (X_0, X_1)$, where $X_0$ and $X_1$ are drawn randomly and independently from $\{0,1\}^k$, and $VK = (Y_0, Y_1) = (f(X_0), f(X_1))$.

b) $\text{Sign}_{SK}(m) = X_m$ (remember, $m \in \{0,1\}$)

c) $\text{Ver}_{VK}(\sigma, m) = [f(\sigma) = Y_m]$

Lamport’s scheme is “one-time secure” in the sense that if the adversary wants to find out what the signature for 1 (say) with only one oracle query, that query must have the
form $\text{Sign}_{SK}(0) = X_0$, which is just a random string and doesn’t help him at all in finding out what $\text{Sign}_{SK}(1)$ is. The intuition can be easily transformed into a proof.

**Remark 1** In fact, Lamport’s scheme is “many-time” secure as well for the trivial reason that there are only two messages. So the only non-trivial attack by the adversary is to forge a signature of $b \in \{0, 1\}$ given a signature of $(1 - b)$, i.e. general security is the same as one-time security.

We next generalize this to many bits.

### 2.2 Lamport’s Scheme for many bits

The generalization for long (say, length $n = p(k)$) messages of Lamport’s scheme is:

**Definition 2** [Lamport’s scheme for $n$-bit messages] Using the notation from the definition of $\text{PKS}$, and letting $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a fixed OWF, we define Lamport’s scheme for $\{0, 1\}^n$ by:

a) Let $SK = (X_0^1, X_1^1, X_0^2, X_1^2, \ldots, X_0^n, X_1^n)$, where the $X_i^j$’s are drawn randomly and independently from $\{0, 1\}^k$, and $VK = (Y_0^1, Y_1^1, Y_0^2, Y_1^2, \ldots, Y_0^n, Y_1^n)$ with $Y_i^j = f(X_i^j)$.

b) $\sigma = \text{Sign}_{SK}(m_1 \ldots m_n) = X_{m_1}, \ldots, X_{m_n}$.

c) $\text{Ver}_{VK}(\sigma_1 \ldots \sigma_n, m_1 \ldots m_n) = [\forall i \in \{1, \ldots, n\}, f(\sigma_i) = Y_{m_i}]$.

What Lamport’s scheme does is it builds two tables, one for signing (in which entry $(i, j)$ corresponds to the block that is used at the $j^{th}$ position of $\sigma$ if $m_j = i$, that is $X_i^j$) and one for verification (in which entry $(i, j)$ corresponds to the block that is used at the $j^{th}$ position of $\sigma$ if $m_j = i$, that is $Y_i^j = f(X_i^j)$). See the illustration below for $n = 5$.

- **Table for Signing**

<table>
<thead>
<tr>
<th>bit/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$X_0^1$</td>
<td>$X_0^2$</td>
<td>$X_0^3$</td>
<td>$X_0^4$</td>
<td>$X_0^5$</td>
</tr>
<tr>
<td>1</td>
<td>$X_1^1$</td>
<td>$X_1^2$</td>
<td>$X_1^3$</td>
<td>$X_1^4$</td>
<td>$X_1^5$</td>
</tr>
</tbody>
</table>

- **Table for Verification**

<table>
<thead>
<tr>
<th>bit/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$Y_0^1$</td>
<td>$Y_0^2$</td>
<td>$Y_0^3$</td>
<td>$Y_0^4$</td>
<td>$Y_0^5$</td>
</tr>
<tr>
<td>1</td>
<td>$Y_1^1$</td>
<td>$Y_1^2$</td>
<td>$Y_1^3$</td>
<td>$Y_1^4$</td>
<td>$Y_1^5$</td>
</tr>
</tbody>
</table>

The signature of 01001 is shown in bold.
We notice that an adversary can break the scheme with 2 queries, for if it gets $\text{Sign}(0^n) = X_0^1 \ldots X_0^n$ and $\text{Sign}(1^n) = X_1^1 \ldots X_1^n$, then it knows all $X_i^j$ and can forge a signature for any given message. However, the scheme is one-time secure in the following sense.

**Definition 3** [One-time security for PKS] A PKS $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Ver})$ is said to be one-time secure, that is, existentially unforgeable against CMA with one chosen message query only if for all PPT $A$ we have

$$\Pr(\text{Ver}(m, \sigma) = \text{accept} \mid (SK, VK) \leftarrow \text{Gen}(1^k), (m, \sigma) \leftarrow A^{\text{Sign}_{SK}(VK)}) \leq \text{negl}(k)$$

where $A$ can use the oracle for on at most one query $q$ and cannot output forgery $m = q$.

\*Theorem 3 (One-time security of Lamport’s scheme)

Lamport’s scheme is one-time secure provided $f$ is a OWF.

**Proof:** By contradiction. Suppose that there exists a PPT adversary $A$ that violates the definition of one-time security for Lamport’s scheme with non-negligible probability $\varepsilon$. We can assume without loss of generality that $A$ makes exactly one oracle call, and that the query and the forgery string both the (expected) length $n$.

We consider the following “experiment” $B$. Given input $y$ that $B$ tries to invert, $B$ runs $\text{Gen}(1^k)$ to get $SK$ and $VK$. It then modifies one of the blocks of the verification key: instead of $Y_i^j$ we now have $Y_i'^j = y$ for some random pair $(i, j)$, and all the other blocks stay the same. Let’s call this new verification key $VK'$; note that $VK$ and $VK'$ have the same distribution. Also, $B$ knows the secret key $SK'$ corresponding to $VK'$ except for the value $x \in f^{-1}(y)$ that $B$ tries to extract from $A$. The key observation (that is easy to check) is the following: If $y = f(x)$ where $x \in \{0, 1\}^k$ is random, $VK$ and $VK'$ have the same distribution; moreover, $i$ and $j$ are independent of $VK'$. The new verification table looks like this (for $i = 1, j = 3$ and $n = 5$):

<table>
<thead>
<tr>
<th>bit/position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$Y_0^1$</td>
<td>$Y_0^2$</td>
<td>$Y_0^3$</td>
<td>$Y_0^4$</td>
<td>$Y_0^5$</td>
</tr>
<tr>
<td>1</td>
<td>$Y_1^1$</td>
<td>$Y_1^2$</td>
<td>$y$</td>
<td>$Y_1^4$</td>
<td>$Y_1^5$</td>
</tr>
</tbody>
</table>

Denote by $q = q_1 \ldots q_n$ the string whose signature $A(VK')$ asks the signing oracle (simulated by $B$). If it happens that $q_j = i$, then we fail in our attempt to recover $x$. However, since $i$ is random and $VK'$ is independent of $i$, we have that $q_j \neq i$ with probability $1/2$. In this latter case, $B$ can easily “sign” $q$ for $A$ since it does not need $x \in f^{-1}(y)$ for the signature. Thus, with probability at least $\varepsilon/2$ we get that $A$ outputs a valid message/signature pair $(m, \sigma)$, where $m \neq q$, i.e. $m_1 \ldots m_n \neq q_1 \ldots q_n$. Hence, there must be at least one index $\ell$ such that $m_\ell \neq q_\ell$. Since $j$ is chosen at random at the view of $A$ so far was independent from $j$, we get that $\ell = j$ with probability $1/n$. Thus, with overall non-negligible probability $\varepsilon/2n$ we have $m_j = i$, and thus $\sigma_j \in f^{-1}(y)$. Therefore, with non-negligible probability $B$ can output this $\sigma_j$ and invert the OWF $f$. \qed

Lecture 13, page-6
2.3 Long Messages

Although the Lamport scheme allows us to sign arbitrarily long messages (say, of length \( n \)), this comes at the expense of having a public key of length roughly \( 2nk \), which is much larger than the length of the message. As we will see, not only is this inefficient, but it is also insufficient for many practical applications of one-time signatures: for example, those that need to sign verification keys (and there are many of those, stay tuned)! So how do we construct one-time signatures where the length of the verification key is (much) shorter than the length of the message? The answer is to use the hash-then-sign paradigm! Namely, rather than one-time signing a long \( m \) (which might not “fit”), we first hash it down to a short string \( h(m) \), and one-time sign \( h(m) \). What properties are needed from \( h \)? As we saw from last class, collision-resistance is enough. In fact, as was mentioned in the optional material, even universal one-wayness is enough, but we will assume CRHFs for simplicity. Also, although the hash-then-sign was stated for “many-time” signatures, it is easy to see from the proof that it works for one-time signatures as well. Notice, our CRHFs construction had a fixed-length public key \( pk \), and were capable of hashing essentially unbounded-length messages. Thus, using the hash-then-sign with such CRHFs, we get a one-time signature capable of one-time signing essentially unbounded messages, and having a fixed-length verification key \( vk' = (vk, pk) \), where \( vk \) is the verification key capable of handling messages whose length is the output of our hash function. Assuming the latter is proportional to the security parameter \( k \), and that the public key \( pk \) for our CRHF is smaller than \( O(k^2) \), — which is the case for all our constructions, — we get the following corollary by using Lampert’s one-time signature:

**Lemma 2** Given a CRHF whose output size is \( O(k) \) and public key at most \( O(k^2) \), there exists a one-time signature scheme capable of signing arbitrary (polynomial-)length messages, and having a verification key of size \( O(k^2) \).

**Remark 2** Similar lemma holds for UOWHFs as well. The only caveat is that the public key of our UOWHFs (which are not already CRHFs) was proportional to \( \log L \), where \( L \) is the length of the hashed message. Specifically, we knew how to make it roughly \( O(k^2 \log L) \), although better constructions are possible. Thus, using UOWHFs, to sign messages of length \( L \), we get a final verification key of size \( O(k^2 \log L) \). For \( L > k^2 \log k \), this is smaller than the length of the message.

3 From One-time to Full-fledged Security

Let \( OT-SIG = (Gen, Sign, Ver) \) denote any one-time secure signature scheme capable of signing “long-enough” messages. Our question now is: is there a general way of building a secure PKS from \( OT-SIG \)? It is not enough to divide the message into blocks and sign each block separately: Lamport’s scheme did that and yet didn’t meet the security standards that we set for ourselves.

In what follows, we describe several natural approaches, eventually leading to a positive answer to this question.

Lecture 13, page-7
3.1 First Attempt: Use Independent Keys

The first natural idea is to choose \( t \) independent key pairs \( \{(SK_i, VK_i)\}_{i=1}^t \) for our one-time signature scheme, and use the \( i \)-th pair of keys to sign the \( i \)-th message \( m_i \). In particular, after the \( i \)-the message is signed, the signer will remember to never reuse the \( i \)-th secret key again. This “works”, but has the following disadvantages:

- **Signer must keep state.** Indeed, the signer must remember which keys were used, to ensure he will never reuse the old key. Thus, we have what is called a *stateful* signature scheme.

- **A-Priori Bounded Number of Messages.** Clearly, one can only sign \( t \) messages, where \( t \) is the parameter which needs to be decided upon at the beginning. Ideally, we would like to sign an arbitrary (polynomial) number of messages.

- **Long Verification Key.** The length of the verification key \( (VK_1 \ldots VK_t) \) is proportional to the maximum number of signed messages \( t \). Thus, if \( t \) is large, so is the verification key.

- **Long Signing Key.** Same as above, but for the signing key.

3.2 Second Attempt: Merkle Trees

We start with the simplest problem: long verification key \( VK = (VK_1, \ldots, VK_t) \). Instead, let \( h \) is a collision-resistant hash function, and let \( vk = h(VK) \) (plus the description of \( h \)). This works, but, now, the signer must include \( VK \) as part of the signature, since otherwise, the verifier will not know which verification key \( VK_i \) to use. This makes the signature size proportional to \( t \), which is pretty bad.

A natural attempt would be to only provide \( VK_i \) to the verifier as part of the signature. This reduces the length of the signature, but creates another problem: how does the verifier, who only knows \( vk = h(VK_1, \ldots, VK_t) \), check that \( VK_i \) is the correct key, and not some bogus key provided by the attacker? At first, it seems like there is nothing we can do, beside providing the entire \( VK \). But then we can remember the idea of the *Merkle tree*, which solves precisely the problem that we have!

Indeed, by using a “bottom-up” complete binary tree to iteratively hash \( VK_1, \ldots, VK_t \) (using the same \( h \)), we know that we can prove any particular \( VK_i \) by opening only \( \log t \) values on the path from \( VK_i \) to the root: namely, \( VK_i \) itself and all the siblings of the nodes on the path up from \( VK_i \) to the root \( vk \). This is called *Merkle Signature*. As before, the scheme is stateful, allows to sign an a-priori bounded number of messages \( t \), has the secret key proportional to \( t \), but now has a constant (i.e., \( O(k) \)) verification key, and the total signature of size \( O(k \log t) \).

3.3 Third Attempt: Top-Down Approach

We try a different idea here, which will allow us to get rid of the main problem we faced so far — an a-priori bound on the number \( t \) of signed messages. Instead of going bottom-up,
like with Merkle signatures, we will ensure a constant size verification key by going “top-down”. We start with using a simple path, latter extending it to a complete binary tree, and even later taking care of other inefficiencies. The basic idea is due to Naor-Yung.

What we do is the following. Suppose one wants to sign the messages $m_0, m_1 \ldots$ (of appropriate length) in that given order, where there is no bound of $t$. The scheme $\text{SIG}$ that we propose proceeds as follows.

a) First, it gets $(SK_0, VK_0) \leftarrow \text{Gen}(1^k)$. $VK_0$ is the (public) verification key of our new scheme $\text{SIG}$.

b) To sign $m_0$, $\text{SIG}$ gets $(SK_1, VK_1) \leftarrow \text{Gen}(1^k)$, then computes $\sigma_1 = \text{Sign}_{SK_0}(m_0, VK_1)$ (namely, it signs a tuple $[m_0, VK_1]$, where $\sigma_1$ is some special character that doesn’t show up in other places so that we can separate $m_0$ from $VK_1$) and outputs $(\sigma_1, VK_1, m_0)$ as the signature of $m_0$ (for notational convenience, we include the message inside the signature). It also remembers $(\sigma_1, VK_1, m_0)$ for future use.

c) To check whether $(\sigma_1, VK_1)$ is a valid signature for $m_0$ under $\text{SIG}$, the receiver checks if $\text{Ver}_{VK_0}([m_0, VK_1], \sigma_1) = \text{accept}$ (i.e. whether $\sigma_1$ is a valid signature for $[m_0, VK_1]$ in the OT-SIG scheme).

d) Inductively, to sign $m_i$ (for $i \geq 2$), $\text{SIG}$ gets $(SK_{i+1}, VK_{i+1}) \leftarrow \text{Gen}(1^k)$, computes $\sigma_{i+1} = \text{Sign}_{SK_i}(m_i, VK_{i+1})$ and outputs $(\sigma_{i+1}, VK_{i+1}, m_i, \ldots, \sigma_1, VK_1, m_0)$ (i.e. the entire history so far!) as the signature of $m_i$.

e) Finally, to check whether $(\sigma_{i+1}, VK_{i+1}, m_i, \ldots, \sigma_1, VK_1, m_0)$ is a good signature of $m_i$, one successively checks all the signatures by $\text{Ver}_{VK_1}([\sigma_{j+1}, VK_{j+1}], m_j)$, and accepts only if the entire chain is valid ($0 \leq j \leq i$), where the last key $VK_0$ is taken from the public file.

On an intuitive level, this scheme is secure because no key is used more than once, and therefore the “one-timeness” of OT-SIG is enough. Indeed, if an adversary $B$ attacks the sender with successive chosen messages, each answer it will get will correspond to a different secret key and that will circumvent the original limitations of OT-SIG. As we sketch below, this is indeed correct. However, notice a crucial requirement for our OT-SIG: it has to sign messages which are longer than its verification key! Indeed, already at the first level one needs to sign the key $VK_1$ plus the first message using $SK_0$. Luckily, due to Lemma 2 (and remark Remark 2 for UOWHFs), this is no problem!

**Theorem 4** Provided OT-SIG can sign messages longer that the length of its verification key, the above (stateful and inefficient) construction is existentially unforgeable against chosen message attack (for messages of corresponding length as explained above).

**Proof:** The proof is simple, but a bit tedious, so we just sketch the idea (the sketch below can be easily transofrmed into a formal proof).

Say some $A$ asks to sign messages $m_0 \ldots m_t$, gets a chain $(\sigma_{i+1}, VK_{i+1}, m_i, \ldots, \sigma_1, VK_1, m_0)$ from the oracle, and forges the signature $(\sigma'_{i+1}, VK'_{i+1}, m'_i, \ldots, \sigma'_1, VK'_1, m'_0)$ of some $m'_i \not\in \{m_1 \ldots m_t\}$. We claim that there exists and index $j \leq \max(i, t)$ such that “along the way”, $A$ produced a forgery $\sigma'_j$ of a “new message” $[m'_j, VK_{j+1}]$ under the key $SK_j$, which contradicts one-time security of the $j$-th one-time signature. The proof is a bit boring:
1. If \([m'_0, VK'_1] \neq [m_0, VK_1]\), then \([m'_0, VK'_1]\) is a new message w.r.t. \(VK_0\), and \(\sigma_1\) is the forgery.

2. Otherwise (equality so far), if \([m'_1, VK'_2] \neq [m_1, VK_2]\), then \([m'_1, VK'_2]\) is a new message w.r.t. \(VK_1 = VK'_1\), and \(\sigma_2\) is the forgery.

3. Otherwise (equality so far), if \([m'_2, VK'_3] \neq [m_2, VK_3]\), then \([m'_2, VK'_3]\) is a new message w.r.t. \(VK_2 = VK'_2\), and \(\sigma_3\) is the forgery.

4. And so on. The point is that since we have \(m'_i \not\in \{m_1 \ldots m_t\}\), at some point \(j\) we must have inequality: at the worst case, if \(i > t\), \([m'_{t+1}, VK_{t+2}]\) is a new message w.r.t. \(VK_{t+1}\), since no signatures w.r.t. \(VK_{t+1}\) were given to \(A\) by its oracle.

Of course, how do we find this \(j\), and how do we simulate the run of \(A\) with this \(j\). Well, we pick \(j\) at random from \(\{0 \ldots T\}\) (where \(T\) is the upper bound of \(A\)'s running time). We generate all the keys on our own, except for the \(j\)-th key, where we use the given verification key \(VK\) whose one-time security we want to compromise. The formal proof follows quite easily from the above.

\[\square\]

**Remark 3** Notice, however, that this construction (known as the Naor-Yung construction) shows that our explanation of the flaws of the “proof of the signature paradox” is not at all artificial. The scheme \(SIG\) above uses \(\text{Sign}\) (from OT-SIG) as a black box, but each time it does so the secret key is different.

### 3.4 Improvements: removing state and fixing signature size

The signature scheme above has at least two undesirable features. First, it is stateful. Second, the size of the signature is proportional to the number of messages. It turns out, we can remove these negative features.

We start with the second problem. Notice, our verification procedure can be thought as a long path: first signature authenticates the second, the second — the third, and so on until the last \(i\)-th signature authenticates the actual \(i\)-th message signed. Thus, we have a growing path \(VK_0 \rightarrow VK_1 \rightarrow \ldots\). It seems much more economical to use a complete binary tree. Namely, the original \(VK = VK_e\) authenticates two new key: \(VK_0\) and \(VK_1\). \(VK_0\) in turn authenticates \(VK_{00}\) and \(VK_{01}\), while \(VK_1 = VK_{10}\) and \(VK_{11}\). And so on until some level \(k\) (say, our security parameter; we only need the level to withstand the birthday attack, as we shall see). More specifically, imagine the following exponential collection of signatures (kept implicitly): \(\text{Sign}_{SK_e}(VK_0, VK_1), \ldots, \text{Sign}_{SK_e}(VK_{x0}, VK_{x1}), \ldots\) until we authenticate all \(2k\) nodes \(VK_x\), where \(|x| = k\). Now, to sign every message \(m\) we will use a different root-leaf path \(x\) down the tree. Assume we decided on \(x = x_1 \ldots x_k\) based on \(m\) (see how later), the fixed size signature of \(m\) is

---

Lecture 13, page-10
We need to address a couple of questions to make this work. We sketch the answers.

- Can we fit two keys inside our one-time signature? The answer is yes if we use a slightly more shrinking CRHF (or UOWHF). So this is not a problem.

- How do we remember these exponentially many keys? The answer is to use a PRF! Namely, we pick a random seed $s$ for a PRF, and use $f_s(x)$ to get the randomness needed to produce the key pair $(SK_x, VK_x)$ for node $x$. Since $s$ is kept secret, these indeed look like properly generated independent signing/verification keys for our one-time signature scheme.

- How do we choose the path $x$ based on $m$, so that they are all distinct? There are several ways to achieve this. The simplest is to choose $x$ at random. Since $k$ is large enough to withstand birthday attack, the probability of reusing the path is negligible. Alternatively, we can use our PRF to extract $k$ pseudorandom bits out of $m$, and let them define $x$ (this makes the signature deterministic!). Yet alternatively, if the message space is $\{0, 1\}^k$, we can actually use $x = m$ to achieve uniqueness. Finally, for larger message spaces we can use CRHF’s or UOWHF’s (the latter should be fresh for every use) to hash our message space into $\{0, 1\}^k$.

The above semi-formal sketch gives a construction, which: (1) can be based on OWFs and CRHFs (or even UOWHFs); (2) is stateless; (3) can be even made deterministic; (4) has fixed signature size; (5) can sign arbitrarily large messages (if needed, using hash-then-sign method); (6) is existentially unforgeable under the chosen message attack. In particular, if we use the UOWHFs instead of more powerful CRHFs, since it is known that UOWHFs can be built from OWFs, in principle we get the following (very inefficient) result:

**Corollary 3** Secure signature schemes exist iff OWF’s exist.

### 3.5 Efficient Signatures?

Unfortunately, the above construction is extremely inefficient, and can never be used in practice. Thus, the next question is what do we do in practice? Half of the answer we already
know: using the hash-then-sign paradigm, it suffices to design efficient fixed-message-size signature schemes. But how to design the latter ones efficiently? We give two answers to that:

- **Using Specific Number-Theory Assumptions.** For example, the Cramer-Shoup signature scheme is provably secure under a stronger version of the RSA assumption: given composite $n$ and random $y \in \mathbb{Z}_n^*$, it is hard to extract any non-trivial root of $y$. Namely, it is hard to come up with $x \in \mathbb{Z}_n^*$ and $e > 1$ so that $x^e = y \mod n$. The usual RSA assumption fixes $e$ as a (random) input to the problem. We do not have time to present this signature scheme, since its proof is somewhat complicated. Interestingly, this is pretty much the only provable and efficient number-theoretic signature scheme up to date.

- **Using Random Oracle Model.** This is the topic of the next section. It turns out that in this model there is a huge variety of very simple-to-state signature schemes (and other cryptographic primitives, like encryption, etc.) which are: (1) extremely efficient; (2) provably secure. What’s the catch? This model assumes something which does not exist... Curious? Read the next section.

## 4 Random Oracle Model and Full Domain Hash

Remember the hash-then-sign methodology. Informally, it said that if SIG is a “secure” signature scheme, and $h$ is a “well-behaved” hash function, then SIG’ inherits the security of SIG, where $\text{Sign}(m) = \text{Sign}(h(m))$. Thus, a “good” hash function may succeed in preserving the security of our original signature scheme. Let’s ask a more ambitious question. Can a “really good” hash function improve the security of the signature scheme we started from? To be more specific, consider the first signature scheme that comes to mind and that we originally rejected — trapdoor signature. Here $f$ is a TDP and $\text{Sign}(m) = f^{-1}(m)$ while $\text{Ver}(\sigma) = [f(\sigma) \equiv m]$. We saw that this scheme is existentially forgeable under key-only attack. However, let us try to apply the hash-then-sign method hoping that good enough $h$ can make the scheme secure. We get $\text{Sign}(m) = f^{-1}(h(m))$ and $\text{Ver}(\sigma) = [f(\sigma) \equiv h(m)]$. Rephrasing our question above, what properties of $h$ (if any) would make the trapdoor signature above existentially unforgeable against the chosen message attack? From a practical point of view, it seems like having some really good function $h$ indeed improves the security of the trapdoor signature: for example, existential forgery no longer seems possible. But can we prove it?

Before answering this question, let’s try some of the candidates for $h$ which we already know about. First, assume $h$ is chosen at random from the family of CRHF’s, and made part of the public key. Unfortunately, this does not seem to be sufficient. Intuitively, to prove the security of a construction based on a trapdoor permutation, the permutation (and thus its inverse) has to be always applied to a (pseudo)random input. In our case, we compute $f^{-1}(h(m))$, which is (pseudo)random only if $h(m)$ is (pseudo)random. The definition of CRHF’s says nothing about pseudorandomness of $h(m)$. In fact, one can design very non-random CRHF families. Thus, CRHF’s do not seem to suffice. The next attempt will be to choose $h$ from a family of PRF’s. This seems to solve the pseudorandomness of
$h(m)$ issue... but not quite. The problem is, should the seed $h$ to our family be public or private? If it is private, the family is indeed pseudorandom, but then we cannot verify the signatures, since we cannot compute the values $h(m)$. On the other hand, if it is made public, the values $h(m)$ no longer seem pseudorandom! More precisely, keeping $h$ public will never allow us to contradict the security of PRF’s (which rely on the seed being secret). Thus, PRF’s do not quite work as well.

In fact, no conventional function $h$ (even taken from some function family) can work! Indeed, $h$ has to be public for verification, and pseudorandom for security. Do there exist public pseudorandom functions? The answer is no. Once something is public (and efficient), it is never pseudorandom: indeed, predicting $h(m)$ is trivial. So it seems like we did not achieve anything. But what if we assumed that public truly random functions exist? And then, assuming $h$ is such a function, can we prove that the modified trapdoor signature is secure?

4.1 Random Oracle Model

We will demonstrate in a second that the answer to the above question is indeed positive. But first, let us examine our new model more closely. In this model, called the Random Oracle Model, one assumes the existence of a public truly random function $h$. This $h$ is called a random oracle.

The model seems contradictory at first. A skeptic might say: “No wonder we can great things in this model. Since a public function is no longer random, we are assuming something which does not exist. From a false statement, anything can be proven.” And indeed, many people appall the RO model. However, it is not as meaningless as one might imagine. First, we are not necessarily proving the existence of some objects. Usually, we are basing some specific construction on a “ideal function” $h$, and try to argue if this construction is secure. Second, one can “implement” our function in the following way. Imagine a true “random oracle” $O$ sitting in the sky. Whenever we need some value $h(m)$, we give $O$ a query $m$, and $O$ returns $h(m)$. The oracle $O$ is assumed to be completely trusted: (1) all the values returned are random and independent from each other; (2) the same query $m$ will return the same answer $h(m)$, and (3) $O$ does not cooperate with an adversary, so every “new” value $h(m)$ indeed looks random to everyone (including the adversary). In this sense, the model is actually “implementable”. Of course, in real life, the role of $O$ be be played by some publicly known function $h$. But what the proof of security in the RO model really says is the following: “If you believe that the only way the adversary uses the knowledge of $h$ is by computing $h$ at points of its choice, and if the function $h$ indeed looks pseudorandom to such a restricted adversary, then the adversary indeed cannot break the security of the system.” To put it differently, it rules out at least a certain class of “black-box” attacks. Namely, if the adversary wants to break the system, it really has to look at the details of $h$’s implementation and try to exploit them. If it treats $h$ like a “black-box”, it cannot be successful.

To summarize the above discussion, RO model is a very strong assumption, in fact, non-existent. However, the proof in the RO model are not meaningless: they at least show

---

Footnote:

3In fact, there are papers showing it is very unlikely to base the security of the signature above in the “standard model”.

Lecture 13, page-13
that there is something secure about the system designed, and to find a possible flaw, the adversary must utilize the weaknesses of the public function \( h \). In practice, functions like SHA do not seem to have to weaknesses which are easy to exploit, at least up to date.\(^4\) Thus, in practice, scheme secure in the RO model, are “currently secure” in the real life as well. Having said that, one should be very careful when using the random oracle. In particular, to justify a scheme in the RO model, this scheme should satisfy be either significantly simpler/efficient than current schemes without random oracle, or there should be no provable schemes without random oracle that achieve the task at hand. In practice, both cases happen: many things (including signatures and encryption) are very easy to do in the RO model, and for certain advanced cryptographic concepts (i.e., “identity based cryptography” among other), only RO-based solutions are currently known. Below we give the simplest example of using the RO model.

4.2 Full Domain Hash

We now come back to the modified trapdoor signature scheme — called the full domain hash — and show its security in the RO model. The proof will show the power of the model.

Recall, \( f \) is a TDP, \( \text{Sign}(m) = f^{-1}(h(m)) \) and \( \text{Ver}(\sigma) = [f(\sigma) \neq h(m)] \). Intuitively, seeing a signature of some message \( m \) corresponds to seeing a random \( (x, f(x)) \) pair (where \( x = f^{-1}(h(m)) \) is indeed random). Such pair the adversary can generate by itself. On the other hand, forging a signature of a “new” \( m \) corresponds to inverting \( f \) at a random point \( h(m) \), which should be hard since \( f \) is one-way. We translate this intuition into a formal proof.

**Theorem 5** Full domain hash is existentially unforgeable against the chosen message attack, provided \( f \) is a TDP and \( h \) is modeled as a RO.

**Proof:** Assume some \( A \) produces existentially forgery of the full domain hash with probability \( \varepsilon \). Say, \( A \) asks signatures of \( m_1 \ldots m_t \), and forges a signature of \( m \not\in \{m_1 \ldots m_t\} \). In order for \( A \) to work, \( A \) requires oracle access to RO \( h \) (we denote this by writing \( A^h \)). Without loss of generality, we assume that before asking a signature of \( m_i \), \( A^h \) asked the oracle the value \( h(m_i) \). Also, before producing the forgery for \( m \), \( A^h \) checked \( h(m) \). Clearly, \( A^h \) might as well do these things without much loss in efficiency.

Using \( A^h \), we construct an adversary \( B \) which breaks the one-wayness property of \( f \) with non-negligible probability \( \varepsilon/q \), where \( q \) is the maximum number of questions \( A \) asks the random oracle.\(^5\) Naturally, \( B \) should somehow simulate the run of \( A^h \). But here is a “complication”, whose “resolution” really shows the power of the RO model: \( B \) does not have any random oracles! Indeed, \( B \) is supposed to invert a TDP \( f \) in the standard model we were using so far. On the other hand, \( A^h \) expects access to RO \( h \). Where can \( B \) provide \( A^h \) with \( h \)? The answer is, \( B \) simulates the RO by itself. Namely, whenever \( A^h \) wants to get \( h(z) \) for some \( z \), \( A^h \) now “really asks \( B \) for it”, and \( B \) returns \( h(z) \) in place of the oracle. One way for \( B \) to do it is to indeed return a random string for every new query of \( A^h \).

\(^4\)Recently, some attacks on SHA were found, but they are more or less very clever brute-force attacks, which are easily solved by making the output slightly longer and/or increasing the number of rounds.

\(^5\)In turns out that a more careful analysis can improve this pessimistic, but sufficient bound. For clarity, we will not present such better analysis here.
(and return consistent strings for old queries; since the number of queries is polynomial, \( B \) can remember all the queries so far). As we will see in a second, however, \( B \) can be more creative, provided all the answers are indeed random (indeed, \( A \) we do not know if \( A \) works well with “non-random” oracles). In other words, \( B \) can pick a random string in a possibly more complicated way, than by “simply picking a random string”. We will see this in a second.

Let’s now describe our \( B \). On input \( y = f(x) \) for unknown \( x \), \( B \) sets the public key of \( \text{SIG} \) to the description of \( f \), picks a random \( j \in \{1 \ldots q\} \) (recall, \( q \) is the number of question \( A^h \) asks to RO) and starts running \( A^h \) with this verification key. Whenever \( A^h \) asks his \( i \)-th query \( h(z_i) \), \( B \) does the following. If \( z_i \) was already asked, \( B \) answers with the same value as before (i.e., answers consistently). If the query \( z \) is “new” and \( i \neq j \), \( B \) picks a random \( x_i \), computes \( y_i = f(x_i) \), and claims that \( h(z_i) = y_i \). Notice, \( y_i \) is indeed random, since \( f \) is a permutation, and \( x_i \) is random. \( B \) then remembers \( (x_i, y_i, z_i) \), so that in addition to \( y_i \), \( B \) also knows the “signature” \( x_i \) of \( z_i \) (indeed, \( f(x_i) = y_i = h(z_i)! \)). Finally, if the query is new and \( i = j \), \( B \) returns its own input \( y \) and the value \( h(z_j) \). Notice, in the case the “signature” of \( z_j \) is the value \( x = f^{-1}(y) \) that \( B \) is trying to find. Jumping ahead, \( B \) hopes \( A^h \) will forge a signature \( x \) of \( m = z_j \).

Next, we have to tell how \( A^h \) answers the signing queries of \( A^h \). Assume \( A^h \) wants to sign the message \( m_s \). By our assumption, we assumed that \( A^h \) would first ask \( h(m_s) \), so \( m_s = z_i \) for some \( i \). But unless \( i = j \) (in which case \( B \) halts with failure), \( B \) already knows the signature \( x_i \) of \( z_i = m_s \)!

Thus, unless \( i = j \) \( B \) can successfully answers the the signing queries. Finally, it’s \( A^h \)’s turn to output the forgery \( (m, \sigma) \). We assumed that \( A^h \) asked the value \( h(m) \) earlier, so \( m = z_i \) for some \( i \). Notice, if \( \sigma \) is valid and \( i = j \), \( \sigma = f^{-1}(h(z_j)) = f^{-1}(y) = x \). Thus, \( B \) succeeds in inverting \( y = f(x) \) provided \( A^h \) forged the signature of \( z_j \). Since \( j \) was chosen at random, and all the random oracle answers were independent of \( j \) (since \( y \) and all the \( y_i \)’s were random), the probability that \( A^h \) finds a signature of \( z_j \) is at least \( \varepsilon/q \), which is non-negligible. This completes the proof.

We remark on the crucial point of the proof: \( B \) has control over how to simulate the random oracle, so that it can later answer the signing queries of \( A \). This is exactly the power of the RO model. We assume that the adversary can only access \( h \) in a “black-box” way, so it is legal for \( B \) to know \( A \)’s questions, and to prepare “convenient” answers.

**Remark 4** We also notice that our reduction lost a pretty significant factor \( q \) in the security, where \( q \) is the number of hash queries made by the attacker. While the loss of this factor \( q \) is unavoidable with general TDPs, it turns out that a more clever reductions for specific TDPs, including RSA and Rabin. In fact, and TDP \( f \) “induced” from a family of CFPs \((f, g)\) turns out to be sufficient. For such TDPs, the security loss goes from \( \varepsilon/q \) to roughly \( \varepsilon/q_s \), where \( q_s \) is the number of signing queries issued by the attacker (as opposed to \( q \), which is the number of hash queries, and which could be much higher in practice).

We conclude the lecture by pointing out that there are many other simple (and practically important) signature scheme designed and analyzed in the RO model.