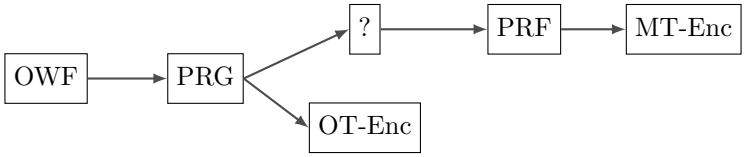
0368.4162 Foundations of Cryptography November 28, 2019

Lecture 5: Constructing and Applying Pseudorandom Functions

Lecturer: Nir Bitansky

1 Previously on Foundations of Crypto

We know that one-way functions (OWF) imply pseudorandom generators (PRG), which gives one-time encryption for messages of arbitrary length. We've seen that to get more advanced forms of encryption, in particular, many-time encryption, we need a more general object called pseudorandom functions (PRFs). Today, we'll see how to construct PRFs and how to use them for more applications.



2 The GGM Pseudorandom Function

Let us start by recalling the definition of PRFs.

Definition 2.1 (Pseudorandom Function (PRF))**.** *A pseudorandom function is a polynomial-time function F that takes as input a seed $s \in \{0,1\}^n$ and input $x \in \{0,1\}^n \cong [2^n]$ and outputs $y \in \{0,1\}^n$, such that for any n.u. PPT $A = \{A_n\}_{n\in\mathbb{N}}$ there exists a negligible μ such that for all $n \in \mathbb{N}$:*

$$\left|\Pr\left[A_n^{F_s(\cdot)} = 1 \;\middle|\; s \leftarrow \{0,1\}^n\right] - \Pr\left[A_n^{R(\cdot)} = 1 \;\middle|\; R \leftarrow \{0,1\}^{n\{0,1\}^n}\right]\right| \leq \mu(n) \ .$$

In '86 Goldreich, Goldwasser, and Micali gave a beautiful construction of a PRF from any PRG.

Theorem 2.2 ([GGM86])**.** *If there exist PRGs, then there exist PRFs.*

Corollary 2.3. *Assuming OWFs, there exist PRFs.*

Proof. The construction is based on a recurring theme in cryptographic constructions — replacing a *chain* by a *tree*, which has the advantage that the number of nodes we can reach is exponential in the depth (instead of linear). Specifically, let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a length-doubling PRG. For a seed s, we'll denote by $G_0(s)$ and $G_1(s)$ the two n-bit parts of the output $G(s)$.

We define:

$$F_s(x) = G_{x_n}(\cdots G_{x_2}(G_{x_1}(s))\cdots) \ .$$

We will now analyze the security of the construction. We will rely on the following claim.

Claim 2.4. *For any polynomial $q = q(n)$,*

$$\left\{U_{2n}^{(1)}, \ldots, U_{2n}^{(q)}\right\}_{n\in\mathbb{N}} \approx_c \left\{G(U_n^{(1)}), \ldots, G(U_n^{(q)})\right\}_{n\in\mathbb{N}} \ .$$

The claim can be proved by a hybrid argument (you proved a more general claim in the homework).

We turn to prove the security of our constructed function F. Let A be an adversary that distinguishes the function F_s (for a random s) from a random function R, with advantage $\delta = \delta(n)$, and assume it makes at

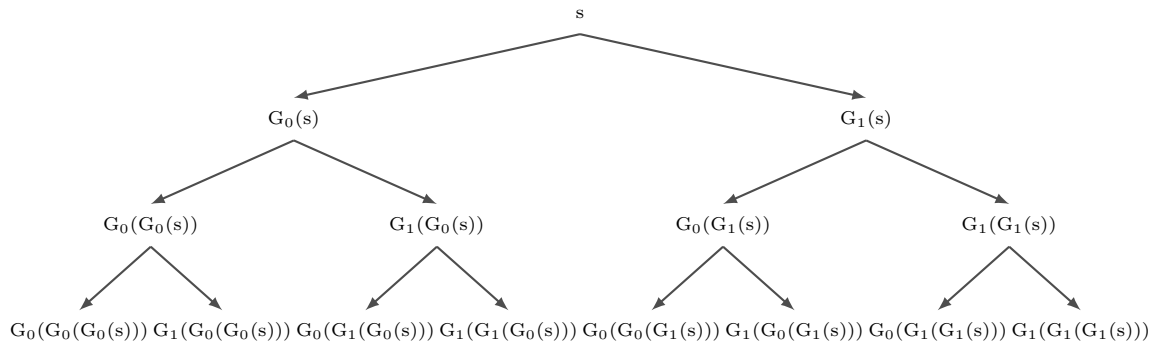


Figure 1: A GGM Tree for inputs of length $n = 3$. To compute $F_s(x_1x_2x_3)$ follow the corresponding path and output the resulting leaf.

most $q = q(n)$ queries to its oracle. We will construct a corresponding distinguisher B for the two distribution ensembles in Claim 2.4.

We consider the following labeling of the nodes $\left\{ x_1 \dots x_i \;\middle|\; \begin{array}{c} 0 \le i \le n \\ x_1 \dots x_i \in \{0,1\}^i \end{array} \right\}$ in the full binary tree. The root of the tree ε (at level zero) is labeled by the random seed $L_\varepsilon := s$. An internal node $x_1 \dots x_i$ (at level i) is recursively labeled by $G_{x_i}(L_{x_1 \dots x_{i-1}})$. Note that each leaf $x_1 \dots x_n$ will be labeled by $L_{x_1 \dots x_n} := F_s(x_1 \dots x_n)$.

We now consider $n + 1$ hybrid experiments $H_0, \dots, H_n$ where we gradually change the labels as follows. In H_i, all the labels $L_{x_1 \dots x_i}$ for nodes at level i are sampled independently at random. Then, the rest of the labels down the tree are chosen according to the same recursive rule as before (the labels at levels $j < i$ are ignored). In each of these hybrid experiments, the adversary's queries are answered according to the leaf labels.

Note that H_0 corresponds exactly to the case that the labels are given by the PRF $L_{x_1 \dots x_n} = F_s(x_1 \dots x_n)$, whereas H_n corresponds to the case that the labels are chosen by a truly random function $L_{x_1 \dots x_n} = R(x_1 \dots x_n)$. Then, there exists an $i \in [n]$ such that A distinguishes H_{i-1} from H_i with advantage δ/n.

The difference between the two hybrids is that:

- In H_{i-1}, all the labels in layer i are chosen pseudorandomly (by applying G to their parent labels).
- In H_i, all the labels in layer i are chosen completely at random.

We now construct the distinguisher B. Recall that B obtains q samples $Y^{(1)}, \dots, Y^{(q)}$ and has to efficiently decide whether they where sampled at random or pseudorandomly. Intuitively, we'd like to use these samples as the labels of ith level in the tree. However, the size of the ith layer could very well be exponential. So we may not have enough samples to do so, and in any case certainly cannot sample all labels efficiently.

The point is that the adversary A doesn't really look at all the leafs, but on at most q of them, so most of the labels are irrelevant. Specifically, we will compute the labels in a lazy fashion.

Algorithm $B(Y^{(1)}, \dots, Y^{(q)})$**:**

- Initialize $\mathcal{L} = \emptyset$ (the set of i-level labels encountered so far).
- Initialize a counter $c = 0$ (the number of samples used so far from the list $Y^{(1)}, \dots, Y^{(q)}$).
- Emulate A, and when it makes a query x act as follows:
  1. If $\mathcal{L}$ contains a pair $(x_1 \dots x_i, L_{x_1 \dots x_i})$, compute the labels down the path from $x_1 \dots x_i$ to $x_1 \dots x_n$ according to the recursive rule, and return the corresponding leaf.

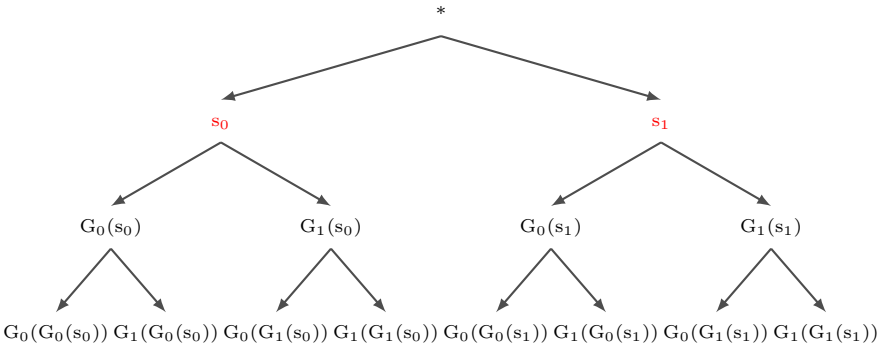


Figure 2: H_i illustrated for $i = 1$

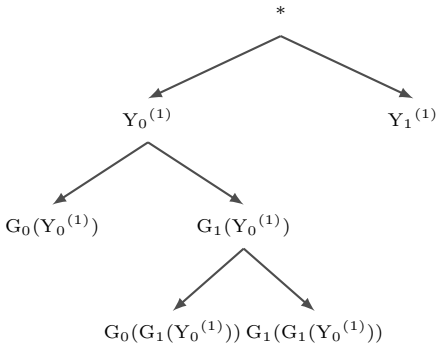


Figure 3: Lazy labeling using the list $Y^{(1)}, ..., Y^{(q)}$ for level i. Paths in the tree are only labeled per A's queries.

2. If $\mathcal{L}$ has no such element:
   - Use the next sample $Y^{(c+1)} = Y_0^{(c+1)} Y_1^{(c+1)}$ to add such an element and a sibling $(x_1 \dots x_{i-1}0, Y_0^{(c+1)}), (x_1 \dots x_{i-1}1, Y_1^{(c+1)})$ to the list $\mathcal{L}$.
   - Increase the counter $c \leftarrow c + 1$.
   - Return to step (1) (now we can compute the corresponding leaf).

- At the end output whatever A outputs.

It is left to see that if the samples $Y^{(1)}, \dots, Y^{(q)}$ are pseudorandom, then A's emulated view is exactly as in H_{i-1}, whereas if they are random, then it's exactly as in H_i. Thus, B manages to distinguish the two cases with advantage δ/n. □

3 From Secrecy to Authenticity

So far, our focus throughout the course was on *privacy* — starting from OWFs, we developed tools like PRGs and PRFs, which in turn led to secret-key encryption schemes for multiple messages. Today, we will change focus and discuss *authenticity*.

The Identification Problem. Let us start by considering a very basic *identification problem*. In this problem, Alice simply wants to prove to Bob that she's indeed Alice and not an impersonator. The adversary

may do more than just listen in on their conversation — it may use the information it gathers in one conversation to try and impersonate Alice in a later conversation. Can we prevent this?

As before, we assume that both parties have met at least once and have agreed on a shared secret. Perhaps the most naïve solution to the problem is that to prove her identity, Alice would simply send Bob the joint secret key, a *password* if you'd like. This clearly fails as once the adversary figures out this password, it can use it from that point on to impersonate. So perhaps Alice should send the password encrypted? A moment of thought reveals that simply sending the encrypted message also fails — it is subject to pretty much the same *replay attack*.

One-Time Puzzles. A simple solution for the problem is the following. Every time that Alice would like to prove her identity, Bob will generate a fresh new *puzzle* that only she could solve. For instance, using a multi-message encryption scheme, Bob can encrypt a random string under their shared secret key and ask that Alice decrypts it. In fact, if Alice and Bob have a shared clock, we can completely avoid interaction. To identify herself at time t, Alice would need to send $F_{sk}(t)$, where F is a PRF with an n-bit output.

Indeed, PRFs essentially give a way of (non-interactively) generating a hard puzzle P_x for any string x in their domain (whose solution is simply $F_{sk}(x)$). Another useful feature is that for any string x, we can delegate the ability to verify the solution for the puzzle for any specific x, without exposing the solution — for instance, we can provide the verifier with $f(F_{sk}(x))$ for a OWF f.

4 Message Authentication Codes

We've sketched how to solve the identification problem, but what if we have a more resourceful adversary, who can hijack the communication line in the middle of the conversation, after Alice had already identified herself. We need a stronger *per-message* authenticity guarantee. This is captured by the notion of *message authentication codes*.

Definition 4.1 (Message Authentication Code (MAC))**.** *A message authentication code consists of polynomial time algorithms $(Auth, Ver)$ with the following properties.*

- **Syntax:**
  - *$Auth_{sk}(m)$ is a possibly randomized algorithm that takes as input a secret key $sk \in \{0,1\}^n$ and message $m \in \{0,1\}^*$, and outputs a tag t.*
  - *$Ver_{sk}(m,t)$ is a determinisitic algorithm that takes as input the secret key sk, message m, and tag t, and outputs bit (where we shall agree that "1" means accept).*
- **Correctness:** *for any message $m \in \{0,1\}^*$,*

$$\Pr\left[Ver_{sk}(m, Auth_{sk}(m)) = 1\right] = 1 \ .$$

- **Unforgeability against Chosen Plaintext Attack:** *For any oracle aided n.u. PPT $A = \{A_n\}$ there exists a negligible μ such that*

$$\Pr\left[\begin{array}{c}(m^*, t^*) \leftarrow A_n^{Auth_{sk}(\cdot)} \\ m^* \notin Q, Ver_{sk}(m^*, t^*) = 1\end{array}\right] \leq \mu(n) \ ,$$

  where $sk \leftarrow \{0,1\}^n$ and Q is the set of queries that A_n makes to $Auth_{sk}(\cdot)$.

The above definition is a rather strong form of unforgeability, which guarantees that even if the adversary can see tags on polyn-many arbitrary messages of its choice, it cannot generate a valid tag for a new message.

MACs from PRFs. Intuitively, MACs exactly require that each message is associated with a hard-to-solve puzzle that can be solved by anyone that has the secret key, and otherwise is hard to solve (even after the attacker witnessed solutions for other puzzles). PRFs give exactly that.

Claim 4.2. *Let F be a PRF, such that for any seed $sk \in \{0,1\}^n$ F_{sk} maps $\{0,1\}^*$ to $\{0,1\}^n$, then the following is a MAC:*

- $Auth_{sk}(m) := F_{sk}(m)$.
- $Ver_{sk}(m,t) = 1$ *iff* $t = F_{sk}(m)$.

Proof Sketch. The idea behind the proof is the following. First, by the pseudorandomness of F, it is sufficient to prove that the scheme is secure when $Auth$ and Ver are given access to a random function $R(\cdot)$. (This is similar to our proof from last week for multi-message security, and common to many other PRF-based proof. Make sure you understand why it is true.)

For a random function, the tag $R(m^*)$ of any message that has not been previously queried is a random and independent value, and thus can be predicted with probability at most 2^{-n}. □

A Note on the Length of Messages. We've defined PRFs for inputs of length n. As we shall see in the homework, it is possible to extend PRFs for n-bit inputs to inputs of arbitrary length.

5 Signature Schemes

MACs allow us to authenticate peer to peer communication in scenarios where both parties share a secret key. Digital signatures go beyond that allowing to authenticate content in a way that can be publicly verified by anyone. For instance, in *public ledgers* such as bitcoin, users perform transactions of the form "Alice paid \$10 to Bob" and post them to ledger. Here everyone should be able to verify that this transaction was indeed performed by Alice, and thus we'd like Alice to be able to *sign* the transaction in a way that anyone could verify.

Definition 5.1 (Signature Scheme)**.** *A signature scheme consists of polynomial-time algorithms $(Gen, Sign, Ver)$ with the following properties.*

- **Syntax:**
  - *$Gen(1^n)$ is a randomized algorithm takes as input a security parameter n and outputs a secret key sk and a public verification key vk.*
  - *$Sign_{sk}(m)$ is a possibly randomized algorithm that takes as input a secret key $sk \in \{0,1\}^n$ and message $m \in \{0,1\}^*$, and outputs a signature σ.*
  - *$Ver_{vk}(m,\sigma)$ is a deterministic algorithm that takes as input the verification key vk, message m, and signature σ, and outputs bit (where we shall agree that "1" means accept).*
- **Correctness:** *for any message $m \in \{0,1\}^*$,*

$$\Pr\left[Ver_{vk}(m, Sign_{sk}(m)) = 1 \mid (sk, vk) \leftarrow Gen(1^n)\right] = 1 \ .$$

- **Unforgeability against Chosen Plaintext Attack:** *For any oracle aided n.u. PPT $A = \{A_n\}$ there exists a negligible μ such that*

$$\Pr\left[\begin{array}{c}(m^*, \sigma^*) \leftarrow A_n^{Sign_{sk}(\cdot)}(vk) \\ m^* \notin Q, Ver_{vk}(m^*, \sigma^*) = 1\end{array}\right] \leq \mu(n) \ ,$$

where $(sk, vk) \leftarrow Gen(1^n)$ and Q is the set of queries that A_n makes to $Sign_{sk}(\cdot)$.

Signature schemes are the first *public key* primitive we've seen so far. For a long time, it was believed that constructing them inherently requires stronger computational assumptions than those required for secret-key cryptography (OWFs); specifically, assumptions of the type used to construct public-key encryption schemes (we'll hear more about those later on). Surprisingly, it turned out that signatures schemes can be constructed from OWFs afterall.

Theorem 5.2. *[Lam79, GMR84, Gol86, NY89, Rom90] If there exist OWFs then there exist digital signature schemes.*

We'll now prove a weaker version of this theorem that also relies on *collision-resistant hashing*. We will then explain, only at high level how to remove this extra assumption.

Step 1: One-Time Signatures. We'll start by constructing a weaker object called a *one-time signature* where the adversary can ask for a signature only on a single message of her choice and should then try to forge a signature on a different message.

Definition 5.3 (One-Message Unforgeability)**.** *A signature scheme is one-time unforgeable if for any oracle aided n.u. PPT* $A = \{A_n\}$ **that makes a single oracle query** *there exists a negligible* μ *such that*

$$\Pr\left[\begin{array}{c}(m^*,\sigma^*) \leftarrow A_n^{Sign_{sk}(\cdot)}(vk) \\ m^* \neq q, Ver_{vk}(m^*,\sigma^*) = 1\end{array}\right] \leq \mu(n) \ ,$$

where $(sk, vk) \leftarrow Gen(1^n)$ *and* q *is the query that* A_n *makes to* $Sign_{sk}(\cdot)$*.*

Step 1.1: Fixed Length Messages. Let us first construct a one-time signature scheme for messages of fixed length n. The construction is as follows:

- $sk = \{x_{i,b} : i \in [n], b \in \{0,1\}\}$ where each $x_{i,b} \leftarrow \{0,1\}^n$.
- $vk = \{f(x_{i,b}) : i \in [n], b \in \{0,1\}\}$, where f is a OWF.
- $Sign_{sk}(m) = \{x_{i,m_i} : i \in [n]\}$
- To verify, apply the function and compare to the verification key.

Claim 5.4. *The above is a secure one-time signature.*

Proof Sketch. Let A be an adversary that breaks the scheme with probability $\varepsilon(n)$, we'll construct an adversary B (with polynomially related running time) that inverts f with probability $\varepsilon/2n$.

$B(y)$**:**

- Samples keys sk, vk as in the actual scheme.
- Samples i, b, and replaces $f(x_{i,b})$ with y, to obtain a new secret key vk'.
- It now emulates $A(vk')$ as follows:
  - When A makes a query m to be signed, if $m_i \neq b$, then B can sign using the secret key sk. If $m_i = b$ it aborts.
  - When A outputs a forgery m^*, $\left\{x^*_{j,m^*_j} : j \in [n]\right\}$, if $m^*_i = b$, B outputs x^*_{i,m^*_i} as a preimage for y; otherwise, it aborts.

We claim that B inverts with probability at least $\varepsilon/2n$. First, note that whenever A produces a valid forgery, then x^*_{i,m^*_i} is a preimage of y. We next analyze the probability that a valid forgery occurs.

Consider an alternative adversary A' for the signature scheme that like B samples i, b at random, and if $m_i = b$ or $m^*_i \neq b$ aborts. Then A' successfully forges a signature with probability $\varepsilon/2n$. Indeed, fix any transcript of the original experiment with respect to A that results in a forgery (such a transcript is fixed by the choice of vk, sk and coins for A). Then this transcript includes a query m and forgery message m^* such that for some j $m_j \neq m^*_j$. The probability that i, b chosen by A' coincide with j, m^*_j is at least $1/2n$. In this case, A' succeeds. Overall, it forges with probability at least $\varepsilon/2n$.

It is left to note that the view of A when emulated by B is distributed identically as the view of A', and thus a forgery is produced with probability at least $\varepsilon/2n$. □

Step 1.2: One-Time Signatures for Arbitrary Length. A limitation of the scheme we've just constructed is that it supported messages of fixed size n — the size of the keys for the scheme scaled with n. Our next step will be to obtain one-time schemes where the size of the keys is independent of the length of messages. (Jumping forward, short keys will prove crucial in the next step when extending one-time signatures to many-time signatures.)

For this purpose, we will introduce a very useful cryptographic tool — *collision-resistant hashing*. These are compressing functions where it is hard to find collisions (although they certainly exist).

Definition 5.5 (Collision-Resistant Hash (CRH))**.** *A collision-resistant hash function is given by a polynomial time algorithm H that given a key $hk \in \{0,1\}^n$ and input $x \in \{0,1\}^*$ outputs a hash $H_{hk}(x) \in \{0,1\}^n$. Furthermore, for any n.u. PPT $A = \{A_n\}_{n\in\mathbb{N}}$ there exists a negligible μ such that for all $n \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{c|c} H_{hk}(x) = H_{hk}(x') & hk \leftarrow \{0,1\}^n \\ x \neq x' & (x,x') \leftarrow A_n(hk) \end{array}\right] \leq \mu(n) \ .$$

Given a collision-resistant function, we can now transform any one-time scheme $(Gen, Sign, Ver)$ for messages of fixed length into a one-time scheme $(Gen', Sign', Ver')$ for messages of arbitrary length:

- $Gen'(1^n)$ samples $(sk, vk) \leftarrow Gen(1^n)$ and $hk \leftarrow \{0,1\}^n$, and outputs $sk' = (sk, hk), vk' = (vk, hk)$.
- $Sign'_{sk'}(m)$ outputs $Sign_{sk}(H_{hk}(m))$.
- $Ver'_{vk'}(m, \sigma)$ output $Ver_{vk}(H_{hk}(m), \sigma)$.

Claim 5.6. *The above is a secure one-time signature.*

Proof Sketch. Let A be an adversary that breaks the scheme with probability $\varepsilon(n)$, we'll construct an adversary B that finds collisions for a random key hk with probability $\varepsilon - n^{-\omega(1)}$. $B(hk)$ will sample sk, vk by herself, append hk to derive sk', vk' and emulate $A(vk')$. We claim that, except with negligible probability, whenever A makes a query m and produces a forgery on $m^* \neq m$, it is the case that $H_{hk}(m) = H_{hk}(m^*)$, namely it finds a collision. Indeed, if this is not the case then A breaks the underlying one-time signature scheme. □

Can Collision-Resistant Hash Functions Be Constructed from OWFs? We do not know the answer to this question, and in fact, it can be shown that *standard proof techniques* cannot establish such a reduction. Still collision-resistance is considered a relatively mild assumption and has many candidates. Later on, we'll mention how in the context of signatures we can actually rely on a weaker object that only guarantees *targeted collision resistance*.

Step 2: A Stateful Scheme for Many Messages. We now wish to extend our one-time scheme into a scheme that can resist an arbitrary number of signing queries. As a first step, we'll construct a stateful scheme where at every point t in time, the signer will posses a state $state_t$. The idea here is to generate

a chain of trust (which is somewhat analogous to the *chain of secrecy* we've seem in previous encryption constructions). At every point in time we will generate a fresh one-time pair of keys, and every time in addition to the message also sign the new verification key. The idea is that if we trust the first verification key in the chain, we'll be able to trust what follows. (Think why to fulfill this strategy it is crucial that the keys in the one-time scheme don't scale with the signed messages.)

We next describe the scheme (a bit less formally than we usually do).

- **State:** After $t \geq 0$ signatures, the state includes:
  - A current one-time signing-key sk_t.
  - A list L_t consisting of triplets $(m_1, vk_1, \sigma_1) \ldots, (m_t, vk_t, \sigma_t)$ including previous messages, verification keys and signatures.
- **Signing:** At time $t+1$, to sign m_{t+1}:
  - Sample a new one-time pair (sk_{t+1}, vk_{t+1}).
  - Compute a signature $\sigma_{t+1} = Sign_{sk_t}(m_{t+1}, vk_{t+1})$.
  - Update L_t to obtain L_{t+1}.
  - Send L_{t+1} as the signature.
  - Store sk_{t+1}.
- **Verifying:** given initial verification key vk_0, message m_{t+1} and signature L_{t+1}, verify the chain starting form vk_0.

Claim 5.7. *The above is a secure (stateful) signature scheme.*

Proof Sketch. Let A be an adversary that breaks the scheme with probability $\varepsilon(n)$ and at most $t(n)$ queries, we'll construct an adversary B that breaks the one-time scheme with probability $\varepsilon/(t+1)$. The basic idea is that to forge a signature on a message outside the chain, A must somewhere create a fork in the chain, or extend it, and that's where it breaks the one-time scheme. Our reduction will guess where this happens and simulate the rest.

The adversary $B(vk)$ guesses $i \in \{0, \ldots, t\}$. It sets $vk_i = vk$, and samples

$$(sk_0, vk_0), \ldots, (sk_{i-1}, vk_{i-1}), (sk_{i+1}, vk_{i+1}), \ldots, (sk_{t-1}, vk_{t-1}), (sk_t, vk_t) \ .$$

It then emulates $A(vk_0)$, it answers queries $1, \ldots, i$ using the sampled secret keys, and on the $i+1$st query m_{i+1}, it queries its own (one-time) oracle on (m_{i+1}, vk_{i+1}). (In the case that $i = t$, there is no $i+1$st query, and no query is made to the one-time oracle.) From there on, it again uses its own sampled keys. Then, when A produces a forgery L^* for m^*, B checks whether L^* first diverges from L_t previously produced in the $i+1$st triplet in the two chains. If not, it aborts. If so, it uses the corresponding message (m^*_{i+1}, vk^*_{i+1}) and one-time signature σ^*_{i+1} as the forgery.

The analysis of the success probability is similar to that in the one-time scheme from OWFs and uses the fact that i is chosen independently of the view of A (until the point it is truncated). □

Step 3: A Stateless Scheme. The previous scheme had two evident problems. First, it was stateful. Second, the length of the signatures L_t grew with the number of signatures. By now, we already know how to solve both problems using *trees and PRFs*. The idea is to look at a full binary tree with nodes $X = \{x_1 \ldots x_i \in \{0,1\}^i, 0 \leq i \in \mathbb{N}\}$ and associate for each node $x \in X$ a one-time pair (sk_x, vk_x). Then the signature on message $m \in \{0,1\}^*$ will include the chain of signatures from the root to the node $m \in X$. Of course that we cannot generate all pairs ahead of time — we will generate them using a PRF (to deal with messages of arbitrary length, we'll use a PRF for inputs of arbitrary length).

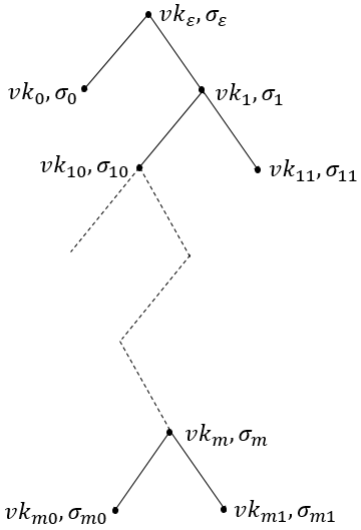


Figure 4: A signature on message m includes verification keys vk_{x0}, vk_{x1} and signatures σ_x on the path from the root to m with intermediate nodes x. The keys (sk_x, vk_x) are computed by $Gen(1^n; F_s(x))$ and signatures σ_x by $sign_{sk_x}(vk_{x0}, vk_{x1})$.

The Construction: In what follows $(Gen', Sign', Ver')$ is a one-time signature for arbitrary length messages and F is a PRF for arbitrary length inputs.

- $Gen(1^n)$: sample a PRF seed $s \in \{0,1\}^n$. Compute $(sk_\varepsilon, vk_\varepsilon) \leftarrow Gen'(1^n; F_s(\varepsilon))$. Save s as the secret key sk and publish vk_ε as the verification key vk.
- $Sign_{sk}(m)$: For every prefix x of m (including the empty one ε):
  - compute $sk_x, vk_x \leftarrow Gen'(1^n; F_s(x))$.
  - compute $\sigma_x = Sign_{sk_x}(vk_{x0}, vk_{x1})$.

  Output $\{\sigma_x, vk_{x0}, vk_{x1}\}_x$.
- $Ver_{vk}(m, \sigma)$: verify all the signatures from the root to the node m.

Note that indeed this scheme is stateless, and the size of the keys is fixed independently of the size of signed messages. The length of the signatures themselves does scale with the length of messages (this can also be dealt with using extra hashing.)

Claim 5.8. *The above scheme is secure assuming the one-time scheme is secure.*

References

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GMR84] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A "paradoxical"'solution to the signature problem (abstract). In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, page 467, 1984.

[Gol86] Oded Goldreich. Two remarks concerning the goldwasser-micali-rivest signature scheme. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 104–110, 1986.

[Lam79] Leslie Lamport. Constructing digital signatures from a one way function. Technical report, October 1979.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 33–43, 1989.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 387–394, 1990.