Digital signatures: for 128 bits of security:

- RSA signatures: 3072 bits $\quad O(\lambda^3)$ bits
- ECDSA signatures: 512 bits $\quad 4\lambda$ bits
- Schnorr signatures: 384 bits $\quad 3\lambda$ bits

Can we construct shorter signatures?

Boneh-Lynn-Shacham (BLS signatures): short signatures from pairings

Starting point: secretly-verifiable signature (i.e., a MAC)

Let $\mathbb{G}$ be a group of prime order p with generator g

KeyGen: sample $k \xleftarrow{R} \mathbb{Z}_p$

Sign(k, m): output $\sigma = H(m)^k$ where $H: \{0,1\}^n \to \mathbb{G}$ is modeled as a random oracle

Verify(k, m, σ): check that $\sigma = H(m)^k$

<u>Observation</u>: signature is single group element (2λ bits)

<u>Security (Sketch)</u>: To forge a signature on the message m^*, adversary needs to compute $H(m^*)^k$.

We can write $H(m^*)$ as g^α for a random α.

Forgery on m^* requires computing $H(m^*)^k = g^{\alpha k}$, which "looks" like a CDH solution.

Question: how will we answer signing queries in the proof?

This will rely on the random oracle (more later).

Aside: $H(m)^k$ is also a PRF (assuming DDH in ROM).

In fact: it is an oblivious PRF.

client (x) $\qquad\qquad\qquad$ server (k)

$$\xrightarrow{\quad y = H(x)^r \quad}$$

$$\xleftarrow{\quad z = y^k \quad}$$

$$z^{r^{-1}} = y^{kr^{-1}} = H(x)^{rkr^{-1}} = H(x)^k$$

implies oblivious transfer (OT)

Question: how to support public verification?

Approach: Publish g^k as the verification key. Use pairing to check signatures.

KeyGen: Sample $k \xleftarrow{R} \mathbb{Z}_p$. Set $sk = k$ and $vk = g^k$

Sign(sk, m): Output $H(m)^{sk}$.

Verify(vk, m, σ): Check $e(g, \sigma) \stackrel{?}{=} e(vk, H(m))$

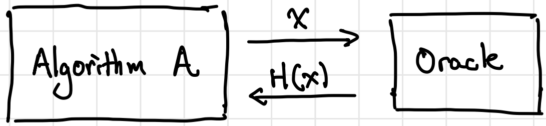
Correctness: $e(g, \sigma) = e(g, H(m)^k) = e(g, H(m))^k = e(g^k, H(m))$

$$= e(vk, H(m))$$

Security: Will rely on CDH assumption in $\mathbb{G}$. Will model H as a random oracle.

Random oracle model (ROM): model H as a uniform random function

An adversary $\mathcal{A}$ in the random oracle model is an adversary that is given oracle access to the random function H:



Formally, algorithm A is an oracle-aided Turing machine. It queries the oracle by writing x to its oracle query tape. The response is provided by writing to its oracle response tape.

Informally: view oracle query as sending a message out. Algorithm A then waits for a response.

Key property: in a reduction in the random oracle model, the reduction algorithm needs to answer the random oracle queries. These answers must be distributed properly.

Theorem. If the CDH assumption holds in $\mathbb{G}$, then BLS signature scheme is secure.

Proof. Suppose there exists efficient A that breaks security of signature scheme. We make the following assumptions about A:

- It makes at most Q queries to the random oracle
- It always queries the random oracle on m before making a signing query on m
- It queries the random oracle on the challenge message m^* at some point in the game

Any adversary A that does not satisfy the latter two properties can be converted into one that does.

We use A to break CDH. The high-level approach is as follows:

- Let (g, g^x, g^y) be the CDH challenge.
- Set g^x as the verification key.
- Set g^y as $H(m^*)$

How to answer signing queries on message m?

- When A queries H on message m, sample $\alpha_m \xleftarrow{R} \mathbb{Z}_p$ and reply with $H(m) := g^{\alpha_m}$.

<u>Observe</u>: g^{α_m} is distributed uniformly over $\mathbb{G}$.

- The signature on m is then $H(m)^x = g^{\alpha_m x} = (g^x)^{\alpha_m}$.

known to reduction!

How do we know which message is m^*?

- We don't! Will guess one at random and hope we get lucky.

Our reduction proceeds as follows. On input (g, u, v):

1. Sample $i^* \xleftarrow{R} [Q]$.
2. Set $vk = u$. Run A and give vk to A.
3. Algorithm A can now make queries:
   (a) Random oracle query on input m.
   If this is the $(i^*)^{th}$ random oracle query, reply with v.
   Otherwise, sample $\alpha_m \xleftarrow{R} \mathbb{Z}_p$ and reply with g^{α_m}.
   (b) Signing query on input m.
   If m was the (i^*) query to the random oracle, abort.
   Otherwise, let α_m be the exponent associated with m.
   Reply with $\sigma = u^{m}$.
4. After A outputs the forgery (m^*, σ^*), check if m^* was the $(i^*)^{th}$ query to random oracle. If not, abort. Otherwise, output σ^* as the CDH solution.

Suppose A is successful. Then it must query random oracle on m^* at some point. Say it is the i^{th} query. Since the reduction chooses $i^* \xleftarrow{R} [Q]$,

$$\Pr[i^* = i] = \frac{1}{Q}$$

Moreover, random oracle queries + signing queries simulated perfectly. Thus, the reduction algorithm B succeed with advantage

$$\text{CDHAdv}[B] \leq \frac{1}{Q}\,\text{SigAdv}[A]$$

We refer to $1/Q$ as the loss in the security reduction. This is saying that the advantage of the signing adversary can be a factor of Q larger than the CDH advantage.

If we want $\text{SigAdv}[A] < 2^{-128}$ and $Q = 2^{20}$, we will need $\text{CDHAdv}[B] < 2^{-148}$ (i.e., CDH now needs 148-bits of security)

need to choose slightly larger groups (though not done in practice)

Can we obtain a <u>tight</u> reduction? Namely: $\text{CDHAdv} \leq O(1) \cdot \text{SigAdv}$ (independent of Q)

Challenge problem in Exercise Set 1. Hint: Modify BLS to use a randomized signing algorithm. Every message will have two possible signatures and loss will be $1/2$.

useful for aggregating certificate chains

BLS signatures are <u>aggregatable</u>.

Suppose we have verification keys $(vk_1, \dots, vk_n)$, messages $(m_1, \dots, m_n)$, signatures $(\sigma_1, \dots, \sigma_n)$

$$\left.\begin{array}{l} (vk_1, \dots, vk_n) \\ (m_1, \dots, m_n) \\ (\sigma_1, \dots, \sigma_n) \end{array}\right\} e(g, \sigma_i) = e(H(m_i), vk_i)$$

Can we have a single signature σ on $(m_1, \dots, m_n)$ with respect to $(vk_1, \dots, vk_n)$?

Let $\sigma = \prod_{i \in [n]} \sigma_i$ (aggregated signature)

Then:

$$\begin{aligned} e(g, \sigma) &= e\left(g, \textstyle\prod_{i \in [n]} \sigma_i\right) \\ &= \textstyle\prod_{i \in [n]} e(g, \sigma_i) \\ &= \textstyle\prod_{i \in [n]} e(H(m_i), vk_i) \end{aligned}$$

<u>Note:</u> when $m_i = m$ for all $i \in [n]$, then verification just requires <u>two</u> pairings:

"multisignature"
Useful when multiple parties need to sign off on an action

$$e(g, \sigma) = \prod_{i \in [n]} e(H(m), vk_i)$$

$$= e(H(m), \underbrace{\prod_{i \in [n]} vk_i}_{\text{aggregated verification key}})$$

The rogue-key attack: Alice: $vk_A = g^{k_A}$
Bob: $vk_B = g^{k_B}$

Given (vk_A, m_A, σ_A) and (vk_B, m_B, σ_B), we can aggregate $(\sigma_A, \sigma_B) \mapsto \sigma$ on (m_A, m_B) with respect to (vk_A, vk_B)

Suppose adversary wants to make it appear like Bob signed message m_B^*

Adversary picks $\alpha \xleftarrow{R} \mathbb{Z}_p$ and sets $vk' = \frac{g^\alpha}{vk_B} = g^{\alpha - k_B}$

<u>Note:</u> adversary does <u>not</u> know k_B, so adversary cannot produce signatures itself

However, adversary can produce an <u>aggregate</u> signature on (m_B^*, m_B^*) with respect to (vk_B, vk')

aggregate signature: $\sigma_{agg} = H(m_B^*)^{k_B} H(m_B^*)^{\alpha - k_B} = H(m_B^*)^\alpha$

adversary knows α so it can compute this!

In general, a malicious party can register a "bad" public key and make it appear that a quorum of honest parties all signed off on a message

Secure aggregation:

adversary		challenger
		$(vk, sk) \leftarrow \text{KeyGen}$
	$\xleftarrow{\quad vk \quad}$	
	$\xrightarrow{\quad m \quad}$	
	$\xleftarrow{\text{Sign}(sk,\, m)}$	

$$\downarrow$$

$$(m_1, \dots, m_n),\ (vk_1, \dots, vk_n),\ \sigma_{agg}$$

Adversary wins if $\text{Verify}((vk_1, \dots, vk_n), (m_1, \dots, m_n), \sigma_{agg}) = 1$ and there exists $i \in [n]$ where $vk_i = vk$ and A does not ask for a signature on m_i

Supporting secure aggregation:

1. Instead of signing message m, sign the pair (vk, m) — namely, individual signatures now bind to an associated verification key.

   Previous attack no longer applies:

   $$\sigma_{agg} = \underline{H(vk_B, m_B^*)}^{k_B} \cdot \underline{H(vk', m_B^*)}^{\alpha - k_B}$$

   different bases so no cancellation of k_B

   Can show that this scheme satisfies secure aggregation

2. Include a proof of possession of signing key (i.e., "sign" the public key)

   $$vk = (u, \pi) \quad \text{where} \quad u = g^k \ (\text{BLS verification key}), \quad \pi = H'(u)^k$$

   Here, H' is modeled as a random oracle (independent of H).

   To check that vk is valid, check that $e(g, \pi) = e(u, H'(u))$

$$e(g, \pi) = e(g, H'(u)^k) = e(g^k, H'(u)) = e(u, H'(u))$$

When verifying a signature (normal or aggregate), first check that each provided verification key is <u>valid</u>.

Can show that π "proves knowledge" of secret key k in the random oracle model. Important that H' is independent of H.

Previous attack does not apply since adversary needs to come up with

$$\pi = H'(g^{\alpha - k_B})^{\alpha - k_B}$$ but adversary does not have k_B

Can show this scheme satisfies secure aggregation.

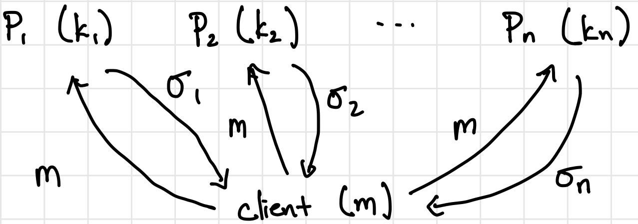
Threshold BLS signatures: To protect the signing key in cryptographic constructions, the secret key is often "secret-shared" across multiple devices / parties

BLS signing: $\sigma = H(m)^k$

n-out-of-n secret sharing: sample $k_1, \ldots, k_n \xleftarrow{R} \mathbb{Z}_p$ and set $k = \sum_{i \in [n]} k_i$

$$vk = g^k \qquad sk_1 = k_1, \ldots, sk_n = k_n$$

To sign, each of n parties needs to sign:



$\sigma_i = H(m)^{k_i}$ — client can verify each "share" σ_i is valid (with respect to $vk_i = g^{k_i}$)

To reconstruct signature on m: compute $\sigma = \prod_{i \in [n]} \sigma_i = \prod H(m)^{k_i}$

$$= H(m)^{\sum_i k_i}$$
$$= H(m)^k$$

All n parties need to sign. Adversary must learn all $k_1, \ldots, k_n$ to forge signatures (subset hides k completely).

What if we want t-out-of-n signing? Use <u>polynomials</u>!

Let p be a large prime. Let $x_1, \ldots, x_d \in \mathbb{F}_p$ be distinct values.

Then for all $y_1, \ldots, y_d \in \mathbb{F}_p$, there exists a <u>unique</u> polynomial f of degree $d-1$ where $f(x_i) = y_i$. The polynomial f is a solution to the following linear system:

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{d-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{d-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_d & x_d^2 & \cdots & x_d^{d-1} \end{bmatrix}}_{\text{Vandermonde matrix } V} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{d-1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix}$$

$$f(x) = \sum_{i=0}^{d-1} f_i x^i$$

Unique solution exists if Vandermonde matrix $V \in \mathbb{F}_p^{d \times d}$ is full-rank.

Can be shown that $\det(V) = \prod_{0 \le i \le j \le d} (x_j - x_i)$

If x_i, x_j are distinct, then $\det(V) \ne 0$ (no zero divisors in a field)

Computing $f_0, \ldots, f_{d-1}$ from $y_1, \ldots, y_d$ is Lagrange interpolation

↳ When $x_1, \ldots, x_d$ are d^{th} roots of unity (and d is power of two), interpolation can be implemented in $O(d \log d)$ time using FFT

<u>Shamir secret sharing</u>: suppose we want to share a message m with n users so that any subset t of them can reconstruct (and any subset with $< t$ users learn nothing about the message).

<u>Approach</u>: Suppose we have n users, and we work over $\mathbb{F}_p$ where $p > n$. Let $s \in \mathbb{F}_p$ be the secret value.

To share s, sample $\alpha_1, \dots, \alpha_{t-1} \xleftarrow{R} \mathbb{F}_p$ and let

$$f(x) = s + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_{t-1} x^{t-1}$$

The share for user $i \in [n]$ is $f(i)$

To reconstruct, a set of t users $T \subseteq [n]$ can interpolate f from $(i, f(i))_{i \in T}$ and evaluate $f(0) = s$

<u>Observe</u>: $\deg(f) = t - 1$ so t shares perfectly determine f

<u>Security</u>: given $t-1$ shares, value of s is <u>perfectly</u> hidden

To show this, we argue that the distribution of any set of $(t-1)$ shares is independent and uniform over $\mathbb{F}_p$.

Fix any set of distinct <u>non-zero</u> points $z_1, \dots, z_{t-1} \in \mathbb{F}_p$. Consider the mapping

$$(\alpha_1, \dots, \alpha_{t-1}) \longmapsto [g(z_1), \dots, g(z_{t-1})]$$

where $g(x) = s + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_{t-1} x^{t-1}$

We claim that this function is a bijection

Suppose there exists $(\alpha_1, \ldots, \alpha_{t-1}) \neq (\alpha'_1, \ldots, \alpha'_{t-1})$ such that the associated polynomials g, g' satisfy

$$g(z_i) = g'(z_i) \quad \forall i \in [t-1]$$

In addition, $g(0) = s = g'(0)$. Thus, the polynomial $g - g'$ has roots at $0, z_1, \ldots, z_{t-1}$ (recall that $z_i \neq 0$). This is a collection of t distinct roots.

However, $\deg(g) = t-1 = \deg(g')$. Thus, $\deg(g-g') = t-1$. This means $g - g' = 0$ and $g = g'$, which is a contradiction.

Thus, the mapping $(\alpha_1, \ldots, \alpha_{t-1}) \mapsto [g(z_1), \ldots, g(z_{t-1})]$ is injective, and thus, a bijection.

If $(\alpha_1, \ldots, \alpha_{t-1}) \xleftarrow{R} \mathbb{F}_p^{t-1}$, then the distribution of $[g(z_1), \ldots, g(z_{t-1})]$ is correspondingly uniform. This holds for all distinct non-zero $z_1, \ldots, z_{t-1}$ and the claim holds.

Shamir secret sharing has <u>linear</u> reconstruction: given shares $(x_1, y_1), \ldots, (x_t, y_t)$, define the Vandermonde matrix associated with $x_1, \ldots, x_t$:

$$V = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{t-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_t & x_t^2 & \cdots & x_t^{t-1} \end{bmatrix}$$

Then coefficients of $f = (s, \alpha_1, \ldots, \alpha_{t-1})^T$ satisfy

$$V \cdot f = y \quad \text{where } y = (y_1, \ldots, y_t)^T$$

Since $s = e_1^T f$, we can write reconstruction as computing

basis vector

$$s = e_1^T f = e_1^T V^{-1} y$$

This is a linear function of the shares y. We usually write $\lambda^T = e_1^T V^{-1}$ to denote the vector of Lagrange interpolation coefficients associated with $x_1, \ldots, x_t$.

Threshold BLS: To construct a t-out-of-n secret share of the BLS signing key k, apply Shamir secret sharing to k.

Let $k_1, \ldots, k_n$ be the shares of k.

The i^{th} party signs a message m by outputting $(\underbrace{i}_{\text{share index}}, \underbrace{H(m)^{k_i}}_{\text{signature share}})$

Given t shares $(x_1, \sigma_1), \ldots, (x_t, \sigma_t)$, we reconstruct in the exponent. Let λ^T be the Lagrange interpolation coefficients associated with $x_1, \ldots, x_t$. Output

$$\prod_{i \in [t]} \sigma_i^{\lambda_i} = \prod_{i \in [t]} H(m)^{\lambda_i k_{x_i}} = H(m)^{\sum_{i \in [t]} \lambda_i \underbrace{k_{x_i}}_{\text{shares of key } k}}$$

$$= H(m)^k$$