Signatures from trapdoor permutations (the full domain hash):

In order to appeal to security of TDP, we need that the argument to $F^{-1}(td,\cdot)$ to be random

Idea: hash the message first and sign the hash value (often called "hash-and-sign")

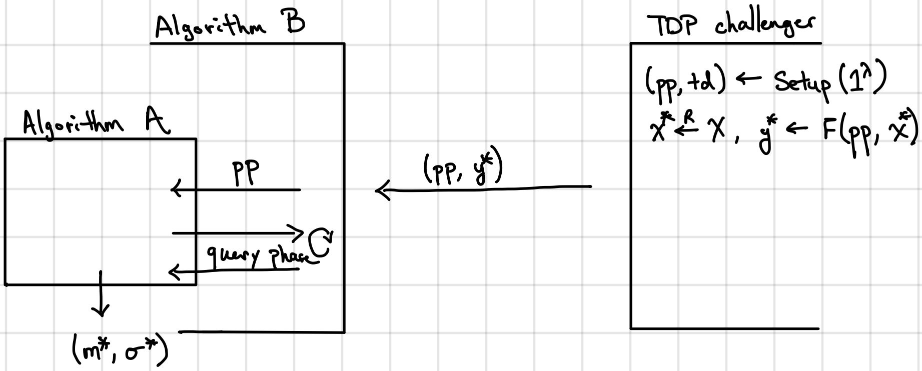
↳ Another benefit: Allows signing long messages (much larger than domain size of TDF)

FDH construction:

- $\mathrm{Setup}(1^\lambda)$: Sample $(pp, td) \leftarrow \mathrm{Setup}(1^\lambda)$ for the TDP and output $vk = pp$, $sk = td$
- $\mathrm{Sign}(sk, m)$: Output $\sigma \leftarrow F^{-1}(td, H(m))$
- $\mathrm{Verify}(vk, m, \sigma)$: Output 1 if $F(pp, \sigma) = H(m)$ and 0 otherwise

Theorem. If F is a trapdoor permutation and H is modeled as a random oracle, then the full domain hash signature scheme defined above is secure.

Proof. Let A be an adversary for the FDH signature We use A to build an adversary B for the trapdoor permutation:



Claim. If A succeeds with advantage ε, then it must query H on m^* with probability $\varepsilon - 1/|X|$.

Proof. Suppose A does not query m^*. Now, (m^*, σ^*) is a valid forgery only if $F(pp, \sigma^*) = H(m^*)$. However, if A does not query m^*, value of $H(m^*)$ uniform and independent of $F(pp, \sigma^*)$. Thus, A succeeds with prob. $1/|X|$.

Key idea: If A succeeds, it will invert the TDP at $H(m^*)$. [Algorithm B will program the challenge y for $H(m^*)$].

But which query is m^*?

Without loss of generality, assume A queries H on message m before making a signing query to m.

Suppose A makes at most Q queries to the random oracle. Algorithm B will guess which random oracle query is m^*.

1. Algorithm B samples $i^* \xleftarrow{R} [Q]$.
2. When A makes a query to H on input m_i
   - Sample $x_i \xleftarrow{R} X$. Let $y_i \leftarrow F(pp, x_i)$
   - Set $H(x_i)$ to y_i and remember the mapping $m_i \mapsto (x_i, y_i)$

   } for all queries other than query i^*

   On query i^* to H for message m_{i^*}
   - Respond with challenge y^*.

   When A makes a signing query for message m:
   - If $m = m_{i^*}$, then algorithm B aborts and outputs $\perp$.

- Otherwise, B looks up mapping $m \mapsto (x, y)$ and replies with x.

3. If B does not abort and A outputs (m^*, σ^*) where $m^* = m_{i^*}$, B outputs σ^*. Otherwise, it outputs $\perp$.

By construction, all queries to H are answered properly (since x is uniform and $F(pp, \cdot)$ is a permutation)
If A does not make signing query on m_{i^*}, then all signing queries answered perfectly
- With probability $\varepsilon - 1/|X|$, algorithm A will query H on m^*, not make a signing query on m^*, and forge a signature on m^*
- With probability $1/Q$, $m_{i^*} = m^*$ in which case B perfectly simulates the signature security game

Algorithm B succeeds with probability at least $1/Q\,(\varepsilon - 1/|X|) = \varepsilon/Q - \mathrm{negl}(\lambda)$.

↳ $\mathrm{TDPAdv}[B] \geq \frac{1}{Q}\, \mathrm{SigAdv}[A] - \mathrm{negl}$.

"loss in security reduction"

Recap: RSA-FDH signatures:

Some (partial) attacks can exploit very small public exponent ($e = 3$)

Setup(1^λ): Sample modulus N, e, d such that $ed = 1 \pmod{\varphi(N)}$ — typically $e = 3$ or $e = 65537$

Output $vk = (N, e)$ and $sk = (N, d)$

Sign(sk, m): $\sigma \leftarrow H(m)^d$ [Here, we are assuming that H maps into $\mathbb{Z}_N^*$]

Verify(vk, m, σ): output 1 if $H(m) = \sigma^e$ and 0 otherwise

Standard: PKCS1 v1.5 (typically used for signing certificates)

↳ Standard cryptographic hash functions hash into a 256-bit space (e.g., SHA-256), but FDH requires full domain

↳ PKCS 1 v1.5 is a way to pad hashed message before signing:

00 01	FF FF ⋯ FF FF 00	DI	H(m)
16 bits	pad	digest info (e.g., which hash function was used)	message hash (e.g., computed using SHA-256)

↳ Padding important to protect against chosen message attacks (e.g., preprocess to find messages m_1, m_2, m_3 where $H(m_1) = H(m_2) \cdot H(m_3)$)
(but this is not a full-domain hash and cannot prove security under RSA — can make stronger assumption...)