MIT 6.875

Foundations of Cryptography
Lecture 13

Digital Signatures

We showed:

Theorem: Assuming the existence of one-way functions and collision-resistant hash function families, there are digital signature schemes.

Collision-Resistant Hash Functions

A compressing **family of functions** $\mathcal{H} = \{\mathrm{h}: \{0,1\}^m \to \{0,1\}^n\}$ (where $m > n$) for which it is computationally hard to find collisions.

Def: $\mathcal{H}$ is collision-resistant if for every PPT algorithm A, there is a negligible function μ s.t.

$$\Pr_{h \leftarrow \mathcal{H}}[A(1^n, h) = (x, y): x \neq y, h(x) = h(y)] = \mu(n)$$

Construction of CRHF from Discrete Log

$p = 2q + 1$ is a "safe" prime.

$$\mathcal{H} = \{\mathrm{h}: (\mathbb{Z}_q)^2 \rightarrow QR_p\}$$

Each function $h_{g_1,g_2} \in \mathcal{H}$ is parameterized by two generators g_1 and g_2 of QR_p (a group of order q).

$h_{g_1,g_2}(x_1, x_2) = g_1^{x_1} g_2^{x_2}$ mod p.

This compresses 2 log q bits into log p ≈ log q + 1 bits.

Construction of CRHF from Discrete Log

$p = 2q + 1$ is a "safe" prime.

$$\mathcal{H} = \{\mathrm{h}: (\mathbb{Z}_q)^2 \rightarrow QR_p\}$$

Each function $h_{g_1, g_2} \in \mathcal{H}$ is parameterized by two generators g_1 and g_2 of QR_p (a group of order q).

$h_{g_1, g_2}(x_1, x_2) = g_1^{x_1} g_2^{x_2}$ mod p.

Why is this collision-resistant? Suppose there is an adversary that finds a collision (x_1, x_2) and (y_1, y_2)…

Construction of CRHF from Discrete Log

$h_{g_1, g_2}(x_1, x_2) = g_1^{x_1} g_2^{x_2}$ mod p.

Why is this collision-resistant? Suppose there is an adversary that finds a collision (x_1, x_2) and (y_1, y_2)…

$g_1^{x_1} g_2^{x_2} = g_1^{y_1} g_2^{y_2}$ mod p.

$g_1^{x_1 - y_1} = g_2^{y_2 - x_2}$ mod p.

(assume wlog $x_1 - y_1 \neq 0 \; mod \; q$)

$g_1 = g_2^{(y_2 - x_2)(x_1 - y_1)^{-1}}$ mod p. $\Rightarrow$ $DLOG_{g_2}(g_1)$!

What if I want to compress more?

Solution 1: Modify the Discrete Log construction

$$h_{g_1,g_2,g_3}(x_1,x_2,x_3) = g_1^{x_1} g_2^{x_2}\, g_3^{x_3} \text{ mod p.}$$

Solution 2: Domain-extension Theorems.

"If there exist hash functions compressing $n+1$ bits to n bits, then there are hash functions that compress any $\mathrm{poly}(n)$ bits into n bits."

Digital Signatures

Theorem: Assuming the hardness of the discrete logarithm problem, there are digital signature schemes.

Other Constructions of CRHFs

From the hardness of factoring, lattice problems etc.

Not known to follow from the existence of one-way functions.

"Black-box separations": Certain ways of constructing CRHF from OWF/OWP cannot work.
"Finding collisions on a one-way street", Daniel Simon, Eurocrypt 1998.

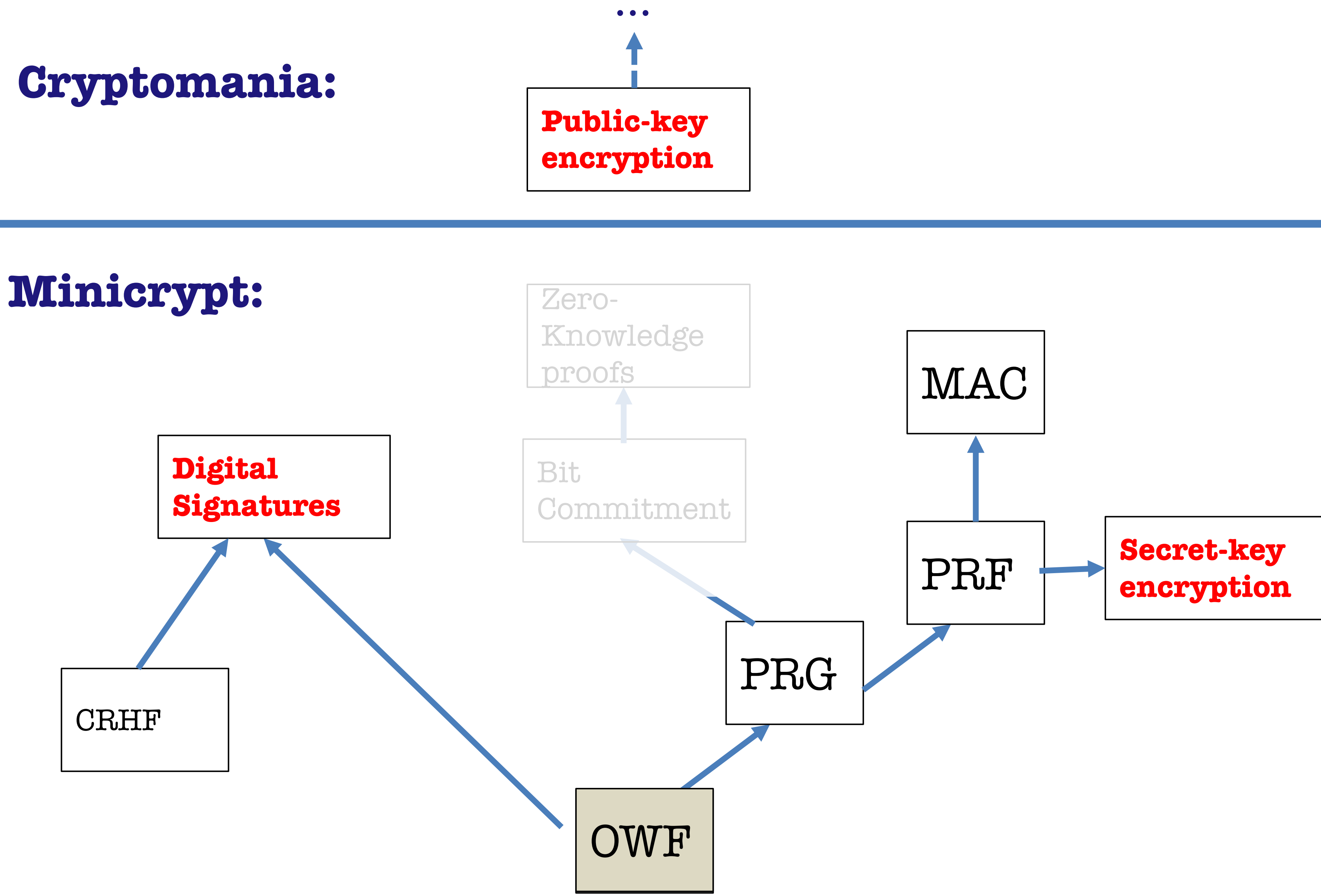
Nevertheless, big open problem: OWF $\overset{?}{\Longrightarrow}$ CRHF?

Digital Signatures

It turns out that collision-resistant hashing is not necessary.

Theorem: Digital Signature schemes exist *if and only if* one-way functions exist.

Worlds in Crypto



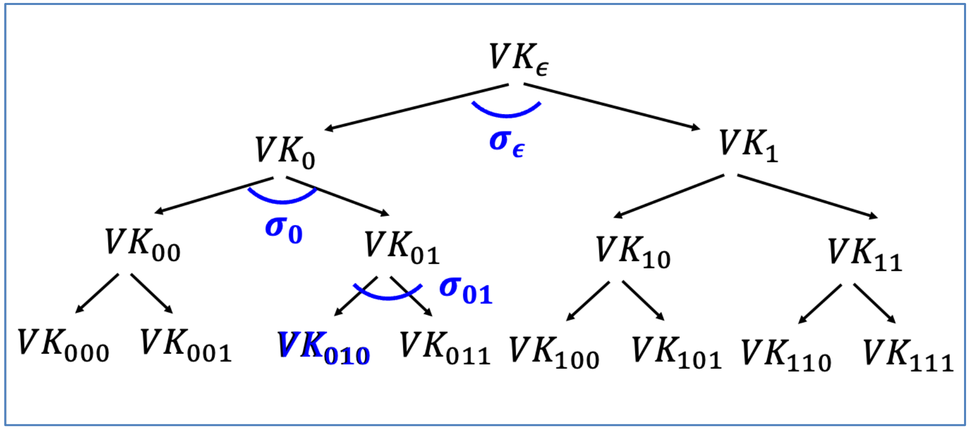
Digital Signature Construction

Start from $(OT.Gen, OT.Sign, OT.Ver)$, a one-time signature scheme that can sign arbitrarily long messages.

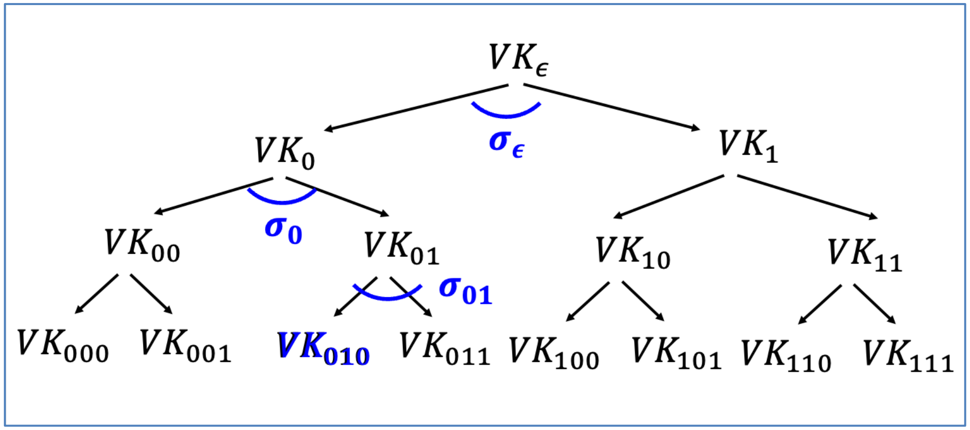
(Lamport + collision-resistant hashing)

Build a (virtual) tree of depth λ = security param.

Let K be a PRF key, $r_i = PRF(K, i)$ for $i \in \{0,1\}^{\leq\lambda}$, and $(VK_i, SK_i) \leftarrow OT.Gen(1^\lambda; r_i)$.



Digital Signature Construction



Signature keys: $SK = K$ and $VK = OTVK_\epsilon$.

Signing Algorithm:

Pick a random leaf $\mathrm{r} \in \{0,1\}^\lambda$,

Generate the authentication path $\sigma_\epsilon, \sigma_{r_1}, \sigma_{r_2}, \ldots, \sigma_r$ & σ^*

$$\sigma_x \leftarrow OT.Sign(SK_x, VK_{x0}||VK_{x1})$$

$$\sigma^* \leftarrow OT.Sign(SK_r, m)$$

The signature is $(\mathrm{r}, \sigma_\epsilon, \sigma_{r_1}, \sigma_{r_2}, \ldots, \sigma_r, \sigma^*)$.

Digital Signature Construction

- Historically regarded as inefficient; therefore, never used in practice.

- However, this signature scheme (or variants thereof) are now called “hash-based signatures” and seeing a re-emergence as a candidate post-quantum secure signature scheme. E.g. https://sphincs.org/

Direct Constructions

"Hash-and-Sign": Secure in the "random oracle model".

"Vanilla" RSA Signatures

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

SK = (N, d) and VK = (N, e)

Sign(SK, m): Output signature $\sigma = m^d \pmod{N}$.

Verify(VK, m, σ): Check if $\sigma^e = m \pmod{N}$.

Problem: Existentially forgeable!

“Vanilla” RSA Signatures

Sign(SK, m): Output signature $\sigma = m^d \pmod{N}$.

Verify(VK, m, σ): Check if $\sigma^e = m \pmod{N}$.

Problem: Existentially forgeable!

Attack: Pick a random σ and output ($m = \sigma^e$, σ) as the forgery.

Problem: Malleable!

Attack: Given a signature of m, you can produce a signature of $2m, 3m, \ldots,$

“Vanilla” RSA Signatures

Sign(SK, m): Output signature $\sigma = m^d \pmod N$.

Verify(VK, m, σ): Check if $\sigma^e = m \pmod N$.

Fundamental Issues:

1. Can ”reverse-engineer” the message starting from the signature (Attack 1)

2. Algebraic structure allows malleability (Attack 2)

How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

SK = (N, d) and VK = $(N, e, \boldsymbol{H})$

Sign(SK, m): Output signature $\sigma = \boldsymbol{H(m)}^d \pmod{N}$.

Verify(VK, m, σ): Check if $\sigma^e = \boldsymbol{H(m)} \pmod{N}$.

So, what is H? Some very complicated "hash" function.

How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

SK = (N, d) and VK = $(N, e, \boldsymbol{H})$

Sign(SK, m): Output signature $\sigma = \boldsymbol{H(m)}^d \pmod{N}$.

Verify(VK, m, σ): Check if $\sigma^e = \boldsymbol{H(m)} \pmod{N}$.

H should be at least one-way to prevent Attack #1.

How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

SK = (N, d) and VK = $(N, e, \boldsymbol{H})$

Sign(SK, m): Output signature $\sigma = \boldsymbol{H(m)}^d \pmod{N}$.

Verify(VK, m, σ): Check if $\sigma^e = \boldsymbol{H(m)} \pmod{N}$.

Hard to "algebraically manipulate" H(m) into H(related m').

(to prevent Attack #2.)

How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

SK = (N, d) and VK = $(N, e, \boldsymbol{H})$

Sign(SK, m): Output signature $\sigma = \boldsymbol{H}(\boldsymbol{m})^d \pmod{N}$.

Verify(VK, m, σ): Check if $\sigma^e = \boldsymbol{H}(\boldsymbol{m}) \pmod{N}$.

Collision-resistance does not seem to be enough. (Given a CRHF h(m), you may be able to produce h(m’) for related m’.)

The Random Oracle Heuristic

Want: A public H that is “non-malleable”.

Given H(m), it is hard to produce H(m’)  any *non-trivially related* m’.

For every PPT adv A and “every non-trivial relation” R,
$$\Pr\left[A\left(h(m)\right) = h(m'): R(m, m') = 1\right] = \mathrm{negl}(\lambda)$$

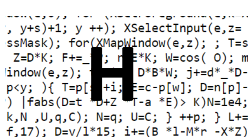
How about the relation R where
$R(x, y) = 1$ if and only if $y = \mathrm{H}(\mathrm{x})$?

The Random Oracle Heuristic

Proxy: A public H that "behaves like a random function"

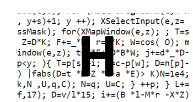
(A PRF also behaves like a random function, but PRF_K is ***not*** publicly computable.)

Reality:

$\mathcal{A}($ 

$)$

Random Oracle Heuristic:

$\mathcal{A}$ 

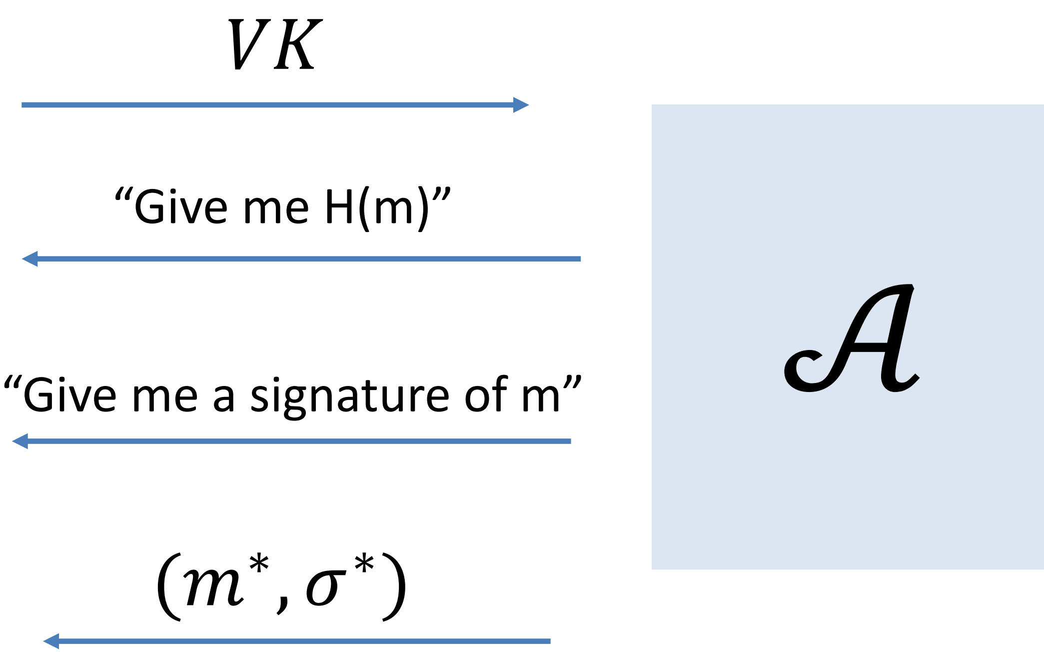
(1^λ)

The only way to compute H is by calling the oracle.

H is virtually a black box.

Proof

Assume there is a PPT adversary $\mathcal{A}$ that breaks the EUF-CMA security of hashed RSA in the random oracle model.



Then, there is an algorithm $\mathcal{B}$ that solves the RSA problem.

Proof

Assume there is a (Q-query) PPT adversary $\mathcal{A}$ that breaks the EUF-CMA security of hashed RSA in the random oracle model.

(N, e, y)

$\mathcal{B}$

$VK = (N, e)$

Hash Query: m

Pick random $\widetilde{m}$ "trap" $H(\widetilde{m}) = y$

"normal" o.w., $H(m) = x^e$

Sign Query: m

$m = \widetilde{m}$? **!abort!**

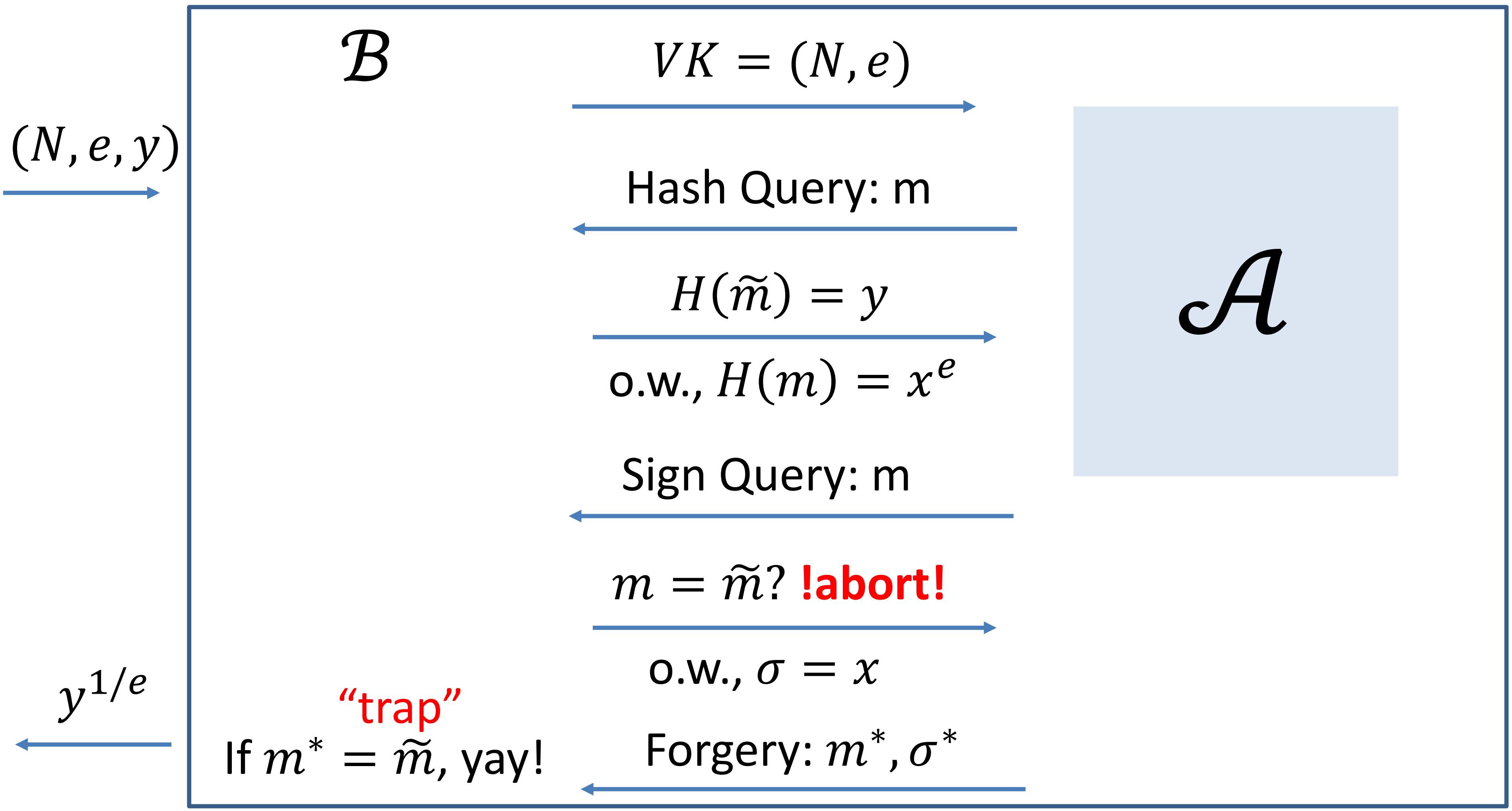
o.w., $\sigma = x$

Forgery: m^*, σ^*

$\mathcal{A}$

Proof

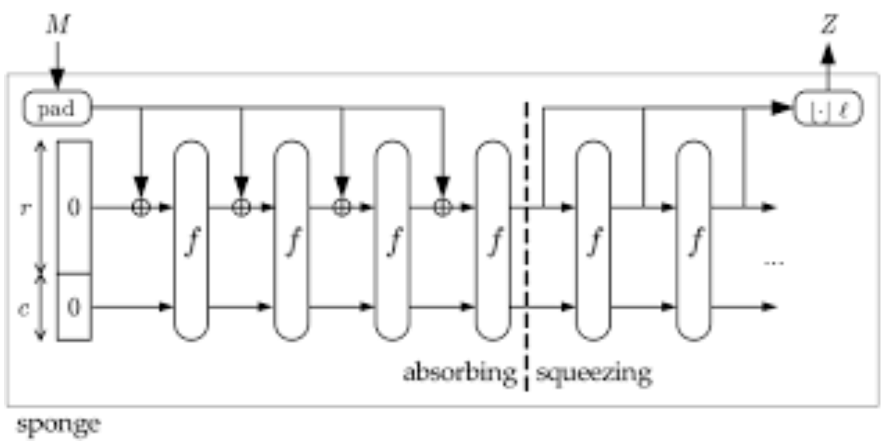
Claim: To produce a successful forgery, $\mathcal{A}$ must have queried the hash oracle on m^*. W.p. $1/Q$, m^* is the trap.



Bottomline: Hashed RSA

(PKCS Standard, used everywhere)

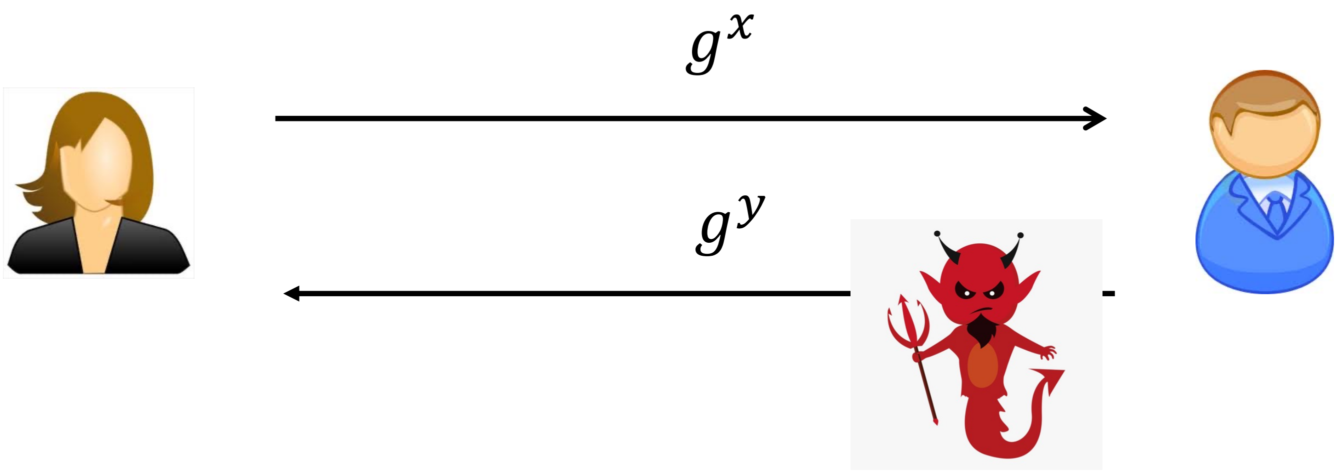
In practice, we let H be the SHA-3 hash function.



… and believe that SHA-3 ”acts like a random function”. That’s the heuristic. On the one hand, it doesn’t make any sense, but on the other, it has served us well so far. No attacks against RSA + SHA-3, for example.

An Application:

Authenticated Key Exchange



An Application:

Authenticated Key Exchange

Bob	vk_B
Alice	vk_A

$$g^x, \mathrm{Sign}(sk_A, g^x) \longrightarrow$$

$$\longleftarrow g^y, \mathrm{Sign}(sk_B, g^y)$$

Many Variants of Signatures (on the board)

Aggregate Signatures: Compressing many signatures into one

Ring Signatures: Protection for Whistleblowers

Threshold Signatures: Protecting against loss of secret key