Authentication II

601.442/642 Modern Cryptography

12th March 2026

Logistics

Logistics

- HW 6 due Today

Logistics

- HW 6 due Today
- Spring break!

Logistics

- HW 6 due Today
- Spring break!
- Midterm on 4/7

Logistics

- HW 6 due Today
- Spring break!
- Midterm on 4/7
  - Will cover everything through the end of Authentication

Logistics

- HW 6 due Today
- Spring break!
- Midterm on 4/7
  - Will cover everything through the end of Authentication
  - (hopefully that means today’s lecture)

Recap: Authentication

k

k

$\mathsf{Enc}(k, m)$

$\mathcal{A}$ sends a message to Bob while *claiming to be Alice.*

How can Bob tell that a message *really did* come from Alice?

Goal: something like a *signature*

- Alice can "sign" a message m to produce a signature σ
- Bob can *verify* that σ is correct for m
- $\mathcal{A}$ cannot *forge* a signature

Recap: MACs

k

m

$\sigma \leftarrow \mathsf{Tag}(k, m)$

(m, σ)

k

If $\mathsf{Ver}(k, m, \sigma) = 1$, accept

else, reject

(m', σ')

Goal: $\mathcal{A}$ can't produce a (m', σ') pair that verifies

Recap: RSA

$$\Pr\left[\mathcal{A}(N, e, x^e \mod N) = x : \begin{array}{c}(p,q) \leftarrow \mathsf{GenRSA}(1^\lambda) \\ N := pq \\ e \xleftarrow{\$} \mathbb{A}^\times_{\varphi(N)} \\ x \xleftarrow{\$} \mathbb{Z}^\times_N\end{array}\right] \leq \mathsf{negl}(\lambda)$$

Recap: RSA

$$\Pr\left[\mathcal{A}(N, e, x^e \mod N) = x : \begin{array}{c}(p,q) \leftarrow \mathsf{GenRSA}(1^\lambda) \\ N := pq \\ e \xleftarrow{\$} \mathbb{A}^\times_{\varphi(N)} \\ x \xleftarrow{\$} \mathbb{Z}^\times_N\end{array}\right] \leq \mathsf{negl}(\lambda)$$

In other words, it is hard to take the e-th root of random elements!

Recap: RSA

$$\Pr\left[\mathcal{A}(N, e, x^e \mod N) = x : \begin{array}{c}(p,q) \leftarrow \mathsf{GenRSA}(1^\lambda) \\ N := pq \\ e \xleftarrow{\$} \mathbb{A}^\times_{\varphi(N)} \\ x \xleftarrow{\$} \mathbb{Z}^\times_N\end{array}\right] \leq \mathsf{negl}(\lambda)$$

In other words, it is hard to take the e-th root of random elements!

Trapdoor: if you have $d \equiv e^{-1}(\mod \varphi(N))$, can take the e-th root of anything!

Recap: RSA

$$\Pr\left[\mathcal{A}(N, e, x^e \mod N) = x : \begin{array}{c}(p,q) \leftarrow \mathsf{GenRSA}(1^\lambda) \\ N := pq \\ e \xleftarrow{\$} \mathbb{A}^\times_{\varphi(N)} \\ x \xleftarrow{\$} \mathbb{Z}^\times_N\end{array}\right] \leq \mathsf{negl}(\lambda)$$

In other words, it is hard to take the e-th root of random elements!

Trapdoor: if you have $d \equiv e^{-1}(\mod \varphi(N))$, can take the e-th root of anything!

RSA public key: (N, e)

Recap: RSA

$$\Pr\left[\mathcal{A}(N, e, x^e \mod N) = x : \begin{array}{c}(p,q) \leftarrow \mathsf{GenRSA}(1^\lambda) \\ N := pq \\ e \xleftarrow{\$} \mathbb{A}^\times_{\varphi(N)} \\ x \xleftarrow{\$} \mathbb{Z}^\times_N\end{array}\right] \leq \mathsf{negl}(\lambda)$$

In other words, it is hard to take the e-th root of random elements!

Trapdoor: if you have $d \equiv e^{-1}(\mod \varphi(N))$, can take the e-th root of anything!

RSA public key: (N, e)

RSA secret key: (N, d)

Recap: RSA

$$\Pr\left[\mathcal{A}(N, e, x^e \mod N) = x : \begin{array}{c}(p,q) \leftarrow \mathsf{GenRSA}(1^\lambda) \\ N := pq \\ e \xleftarrow{\$} \mathbb{A}^\times_{\varphi(N)} \\ x \xleftarrow{\$} \mathbb{Z}^\times_N\end{array}\right] \leq \mathsf{negl}(\lambda)$$

In other words, it is hard to take the e-th root of random elements!

Trapdoor: if you have $d \equiv e^{-1}(\mod \varphi(N))$, can take the e-th root of anything!

RSA public key: (N, e)

RSA secret key: (N, d)

I'm going to refer to the algorithm that computes the values N, e, d as **RSASetup**

Digital Signatures

Digital Signatures

Digital Signatures

(sk, pk)

Digital Signatures

(sk, pk)

pk

Digital Signatures

(sk, pk)

m

pk

Digital Signatures

(sk, pk)

m

$\sigma \leftarrow \mathsf{Sign}(sk, m)$

pk

Digital Signatures

(sk, pk)

m

$\sigma \leftarrow \mathsf{Sign}(sk, m)$

(m, σ)

pk

Digital Signatures

(sk, pk)

m

$\sigma \leftarrow \mathsf{Sign}(sk, m)$

(m, σ)

pk

If $\mathsf{Ver}(pk, m, \sigma) = 1$, accept

Digital Signatures

(sk, pk)

m

$\sigma \leftarrow \mathsf{Sign}(sk, m)$

(m, σ)

pk

If $\mathsf{Ver}(pk, m, \sigma) = 1$, accept

else, reject

Digital Signatures

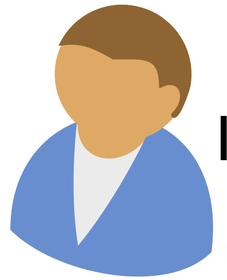
(sk, pk)

m

$\sigma \leftarrow \mathsf{Sign}(sk, m)$

(m, σ)

pk



If $\mathsf{Ver}(pk, m, \sigma) = 1$, accept

else, reject

pk

Digital Signatures

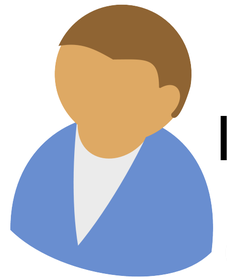
(sk, pk)

m

$\sigma \leftarrow \mathsf{Sign}(sk, m)$

(m, σ)

pk



If $\mathsf{Ver}(pk, m, \sigma) = 1$, accept

else, reject

(m', σ')

pk

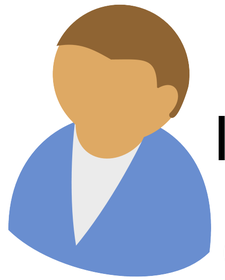
Digital Signatures

(sk, pk)

m

$\sigma \leftarrow \mathsf{Sign}(sk, m)$

(m, σ)



pk

If $\mathsf{Ver}(pk, m, \sigma) = 1$, accept

else, reject

(m', σ')

pk

Goal: $\mathcal{A}$ can't produce a (m', σ') pair that verifies: even after seeing a bunch of valid signatures (m, σ)

Digital Signatures

Digital Signature Scheme Syntax

A *digital signature scheme* consists of three (possibly probabilistic) algorithms:

Digital Signatures

Digital Signature Scheme Syntax

A *digital signature scheme* consists of three (possibly probabilistic) algorithms:

- KeyGen$(1^\lambda) \rightarrow (sk, pk)$ outputs a secret key and a public key

Digital Signatures

Digital Signature Scheme Syntax

A *digital signature scheme* consists of three (possibly probabilistic) algorithms:

- KeyGen$(1^\lambda) \rightarrow (sk, pk)$ outputs a secret key and a public key
- Sign$(sk, m) \rightarrow \sigma$ takes as input a secret key and a message and outputs a signature

Digital Signatures

Digital Signature Scheme Syntax

A *digital signature scheme* consists of three (possibly probabilistic) algorithms:

- KeyGen$(1^\lambda) \rightarrow (sk, pk)$ outputs a secret key and a public key
- Sign$(sk, m) \rightarrow \sigma$ takes as input a secret key and a message and outputs a signature
- Ver$(pk, m, \sigma) \rightarrow b$ takes as input a public key, a message, and a signature and outputs a bit

Digital Signatures

Digital Signature Scheme Syntax

A *digital signature scheme* consists of three (possibly probabilistic) algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to (sk, pk)$ outputs a secret key and a public key
- $\mathsf{Sign}(sk, m) \to \sigma$ takes as input a secret key and a message and outputs a signature
- $\mathsf{Ver}(pk, m, \sigma) \to b$ takes as input a public key, a message, and a signature and outputs a bit

Correctness:

$$\Pr\left[\mathsf{Ver}(pk, m, \sigma) = 1 \;:\; \begin{array}{c}(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sign}(sk, m)\end{array}\right] = 1$$

Digital Signature Security

UF-CMA Security

A Digital Signature scheme (KeyGen, Tag, Ver) satisfies *unforgeability under chosen message attack* (UF-CMA) if for all NUPPT $\mathscr{A}$, there exists a negligible function negl($\cdot$). such that $\forall \lambda \in \mathbb{N}$:

$$\Pr[\mathscr{A} \text{ wins DSGame}] \leq \text{negl}(\lambda)$$

$(sk, pk) \leftarrow \text{KeyGen}(1^\lambda)$

$\sigma_i \leftarrow \text{Sign}(sk, m_i)$

pk

m_i

σ_i

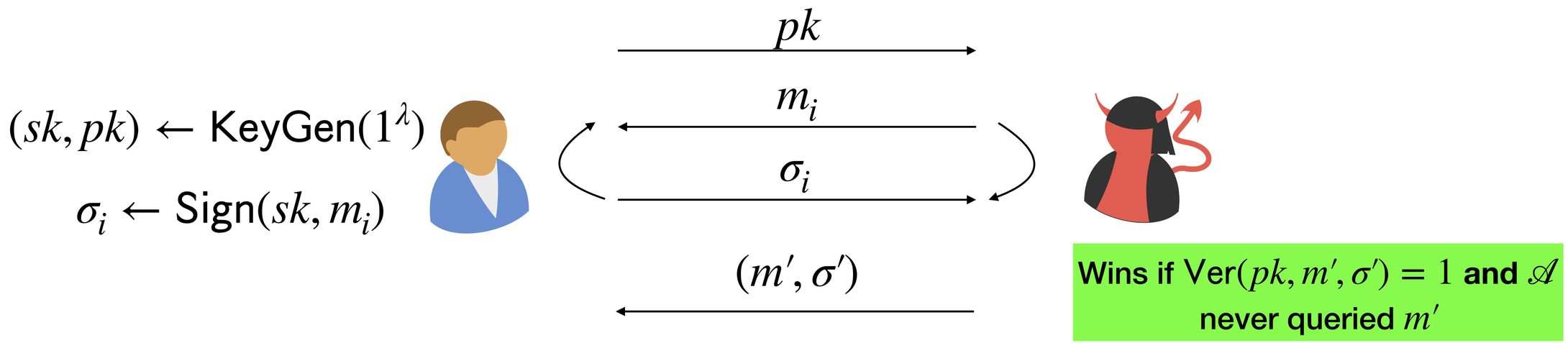
(m', σ')

Wins if $\text{Ver}(pk, m', \sigma') = 1$ **and** $\mathscr{A}$ **never queried** m'

Digital Signature Security

$$\Pr\left[\begin{array}{c}\mathsf{Ver}(pk, m', \sigma') = 1 \\ \textbf{and } \mathscr{A} \text{ never queried } m'\end{array} : \begin{array}{c}(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (m', \sigma') \leftarrow \mathscr{A}^{\mathsf{Sign}(k,\cdot)}(1^\lambda, pk)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

$\Pr[\mathscr{A} \text{ wins DSGame}] \leq \mathsf{negl}(\lambda)$



RSA Signatures

RSA Signatures

RSA Signatures

$$sk = d$$

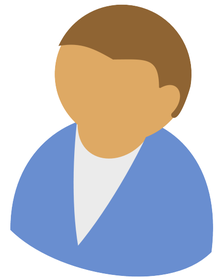
$$pk = (N, e)$$

RSA Signatures

$sk = d$

$pk = (N, e)$

(N, e)



RSA Signatures

$sk = d$

$pk = (N, e)$

m

(N, e)

RSA Signatures

$sk = d$

$pk = (N, e)$

m

(N, e)

Idea: RSA says that *only Alice* can find the e-th root of things.

RSA Signatures

$sk = d$

$pk = (N, e)$

m

(N, e)

Idea: RSA says that *only Alice* can find the e-th root of things.

Therefore, to sign a message Alice can compute it's e-th root, which proves it is her!

Digital Signature Construction?

Digital Signature Construction?

Plain RSA Signatures

Digital Signature Construction?

Plain RSA Signatures

- KeyGen(1^λ)

Digital Signature Construction?

Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)

Digital Signature Construction?

Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e); \quad sk := (N, d)$

Digital Signature Construction?

Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign($(N, d), m$): $\sigma := m^d \mod N$

Digital Signature Construction?

Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign($(N, d), m$): $\sigma := m^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} m \mod N$

Digital Signature Construction?

Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign($(N, d), m$): $\sigma := m^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} m \mod N$

This is completely insecure!

Attacking Plain RSA Signatures

Attacking Plain RSA Signatures

- KeyGen(1^λ)

Attacking Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)

Attacking Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e); \quad sk := (N, d)$

Attacking Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow \mathsf{RSASetup}(1^\lambda)$
  - $pk := (N, e); \quad sk := (N, d)$
- Sign($(N, d), m$): $\sigma := m^d \mod N$

Attacking Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow \mathsf{RSASetup}(1^\lambda)$
  - $pk := (N, e); \quad sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := m^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} m \mod N$

Attacking Plain RSA Signatures

The problem: $\mathscr{A}$ can pick *any* message to generate a forgery on.

- KeyGen(1^λ)
  - $N, e, d \leftarrow \mathsf{RSASetup}(1^\lambda)$
  - $pk := (N, e); \quad sk := (N, d)$
- Sign($(N, d), m$): $\sigma := m^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} m \mod N$

Attacking Plain RSA Signatures

The problem: $\mathscr{A}$ can pick *any* message to generate a forgery on.

Its easy to pick a message where you know the e-th root!

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign$((N, d), m)$: $\ \sigma := m^d \mod N$
- Ver$((N, e), m, \sigma)$: $\ \sigma^e \stackrel{?}{=} m \mod N$

Attacking Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign($(N, d), m$): $\sigma := m^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} m \mod N$

The problem: $\mathscr{A}$ can pick *any* message to generate a forgery on.

Its easy to pick a message where you know the e-th root!

You just pick the root, and then take it to the e.

Attacking Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign($(N, d), m$): $\sigma := m^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} m \mod N$

The problem: $\mathscr{A}$ can pick *any* message to generate a forgery on.

Its easy to pick a message where you know the e-th root!

You just pick the root, and then take it to the e.

N, e

Attacking Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := m^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} m \mod N$

The problem: $\mathscr{A}$ can pick *any* message to generate a forgery on.

Its easy to pick a message where you know the e-th root!

You just pick the root, and then take it to the e.

N, e

$$\sigma' \stackrel{\$}{\leftarrow} \mathbb{Z}_N^\times$$

Attacking Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign($(N, d), m$): $\sigma := m^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} m \mod N$

The problem: $\mathscr{A}$ can pick *any* message to generate a forgery on.

Its easy to pick a message where you know the e-th root!

You just pick the root, and then take it to the e.

N, e

$$\sigma' \stackrel{\$}{\leftarrow} \mathbb{Z}_N^\times$$

$$m' := (\sigma')^e \mod N$$

Attacking Plain RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign($(N, d), m$): $\sigma := m^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} m \mod N$

The problem: $\mathscr{A}$ can pick *any* message to generate a forgery on.

Its easy to pick a message where you know the e-th root!

You just pick the root, and then take it to the e.

N, e

$$\sigma' \stackrel{\$}{\leftarrow} \mathbb{Z}_N^\times$$

$$m' := (\sigma')^e \mod N$$

Clearly **Ver** will pass!

Where did we go wrong?

$$\Pr\left[\mathcal{A}(N, e, x^e \mod N) = x : \begin{array}{c}(p,q) \leftarrow \mathsf{GenRSA}(1^\lambda) \\ N := pq \\ e \xleftarrow{\$} \mathbb{A}^\times_{\varphi(N)} \\ x \xleftarrow{\$} \mathbb{Z}^\times_N\end{array}\right] \leq \mathsf{negl}(\lambda)$$

In other words, it is hard to take the e-th root of random elements!

Where did we go wrong?

$sk = d$

$pk = (N, e)$

m

(N, e)

Idea: RSA says that *only Alice* can find the e-th root of things.

Where did we go wrong?

$sk = d$

$pk = (N, e)$

m

(N, e)

This isn't true!

Idea: RSA says that *only Alice* can find the e-th root of things.

Where did we go wrong?

$sk = d$

$pk = (N, e)$

m

(N, e)

Idea: RSA says that *only Alice* can find the e-th root of **random** things.

Fixing Things

$sk = d$

$pk = (N, e)$

m

(N, e)

Idea: RSA says that *only Alice* can find the e-th root of **random** things.

Fixing Things

$sk = d$

$pk = (N, e)$

m

(N, e)

Idea: RSA says that *only Alice* can find the e-th root of **random** things.

What would be nice: a function H that maps messages m to completely random values.

Fixing Things

$sk = d$

$pk = (N, e)$

m

(N, e)

Idea: RSA says that *only Alice* can find the e-th root of **random** things.

What would be nice: a function H that maps messages m to completely random values.

Then Alice could sign $H(m)$ instead, and RSA would guarantee security!

Fixing Things

$sk = d$

$pk = (N, e)$

m

(N, e)

What would be nice: a function H that maps messages m to completely random values.

Then Alice could sign $H(m)$ instead, and RSA would guarantee security!

Fixing Things

$sk = d$

$pk = (N, e)$

m

(N, e)

What would be nice: a function H that maps messages m to completely random values.

Then Alice could sign $H(m)$ instead, and RSA would guarantee security!

What if we used a PRF? Alice could sign $F_k(m)$

Fixing Things

$sk = d$

$pk = (N, e)$

m

(N, e)

What would be nice: a function H that maps messages m to completely random values.

Then Alice could sign $H(m)$ instead, and RSA would guarantee security!

What if we used a PRF? Alice could sign $F_k(m)$

This doesn't work! Bob needs to compute $H(m)$ himself to check the signature, but they don't share a key!

Fixing Things

$sk = d$

$pk = (N, e)$

m

(N, e)

What would be nice: a function H that maps messages m to completely random values.

Then Alice could sign $H(m)$ instead, and RSA would guarantee security!

Fixing Things

$sk = d$

$pk = (N, e)$

m

(N, e)

What would be nice: a function H that maps messages m to completely random values.

Then Alice could sign $H(m)$ instead, and RSA would guarantee security!

What we need is a **public random function** that both Alice and Bob and run

Fixing Things

$sk = d$

$pk = (N, e)$

m

(N, e)

What would be nice: a function H that maps messages m to completely random values.

Then Alice could sign $H(m)$ instead, and RSA would guarantee security!

What we need is a **public random function** that both Alice and Bob and run

Let's just assume we have one!

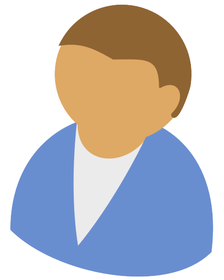
The Random Oracle Model

$sk = d$

$pk = (N, e)$

m

(N, e)



The Random Oracle Model

$sk = d$

$pk = (N, e)$

m

(N, e)

H

The Random Oracle Model

$sk = d$

$pk = (N, e)$

m

(N, e)

x

H

The Random Oracle Model

$sk = d$

$pk = (N, e)$

m

x

H

(N, e)

Keeps track of the random table inside

x	y
10101	11111

The Random Oracle Model

$sk = d$

$pk = (N, e)$

m

(N, e)

x

y

H

Keeps track of the random table inside

x	y
10101	11111

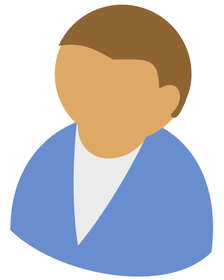
The Random Oracle Model

$sk = d$

$pk = (N, e)$

m

(N, e)



x

y

H

The Random Oracle Model

$sk = d$

$pk = (N, e)$

m

x

y

H

(N, e)

If Bob queries the same value x he gets the same response!

The Random Oracle Model

$sk = d$

$pk = (N, e)$

m

(N, e)

x

y

x

H

If Bob queries the same value x he gets the same response!

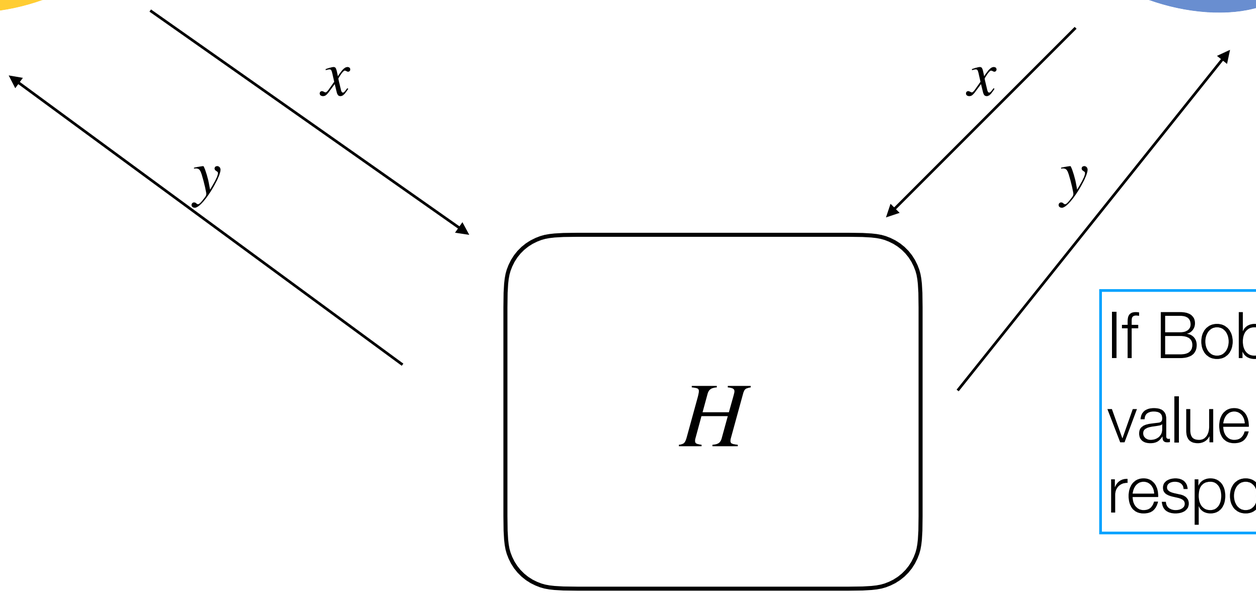
The Random Oracle Model

$sk = d$

$pk = (N, e)$

m

(N, e)



If Bob queries the same value x he gets the same response!

RSA Signatures in the Random Oracle Model

$sk = d$

$pk = (N, e)$

m

(N, e)

H

RSA Signatures in the Random Oracle Model

$sk = d$

$pk = (N, e)$

m

m

H

(N, e)

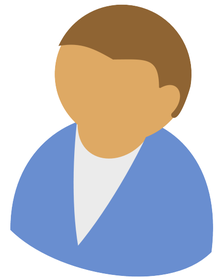
RSA Signatures in the Random Oracle Model

$sk = d$

$pk = (N, e)$

m

(N, e)



m

y

H

RSA Signatures in the Random Oracle Model

$sk = d$

$pk = (N, e)$

m

$\sigma := y^d \mod N$

(N, e)

m

y

H

RSA Signatures in the Random Oracle Model

$sk = d$

$pk = (N, e)$

m

$\sigma := y^d \mod N$

(m, σ)

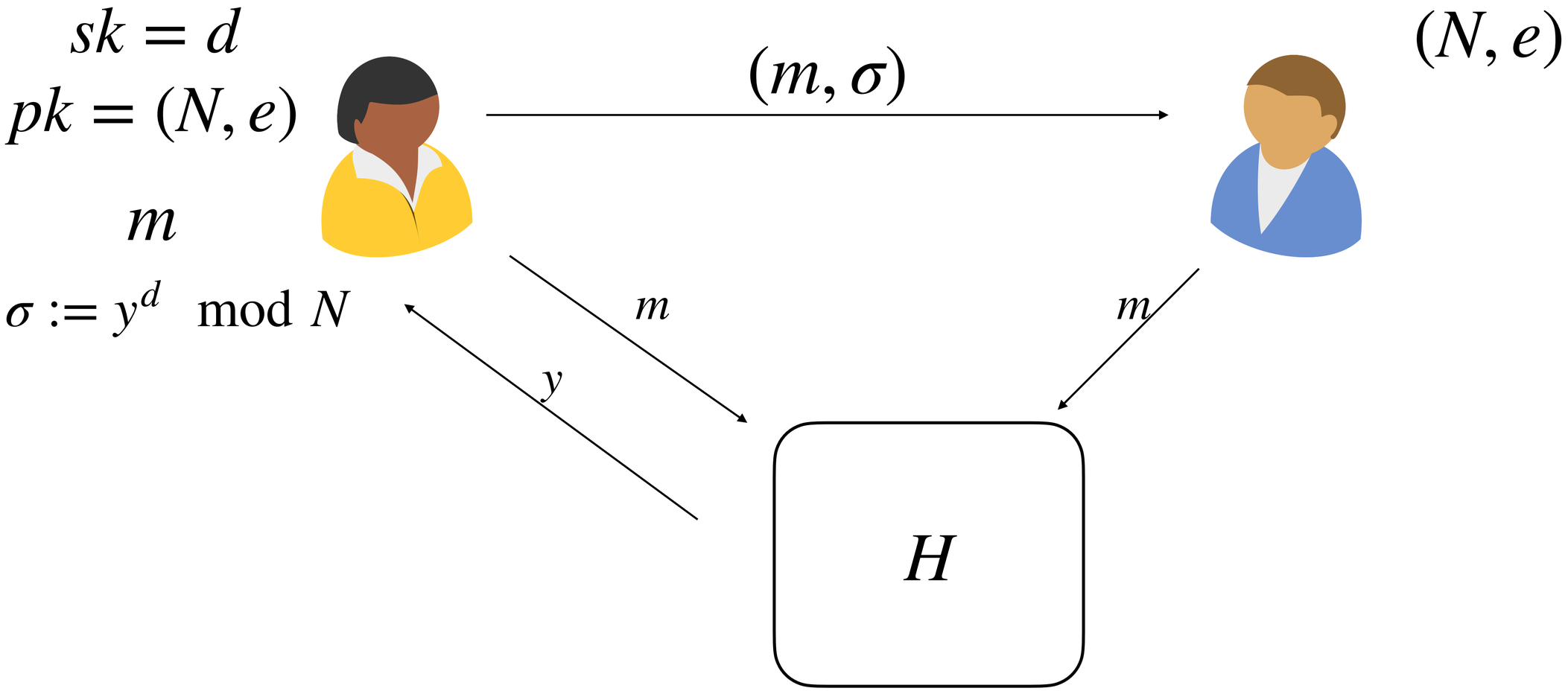
(N, e)

m

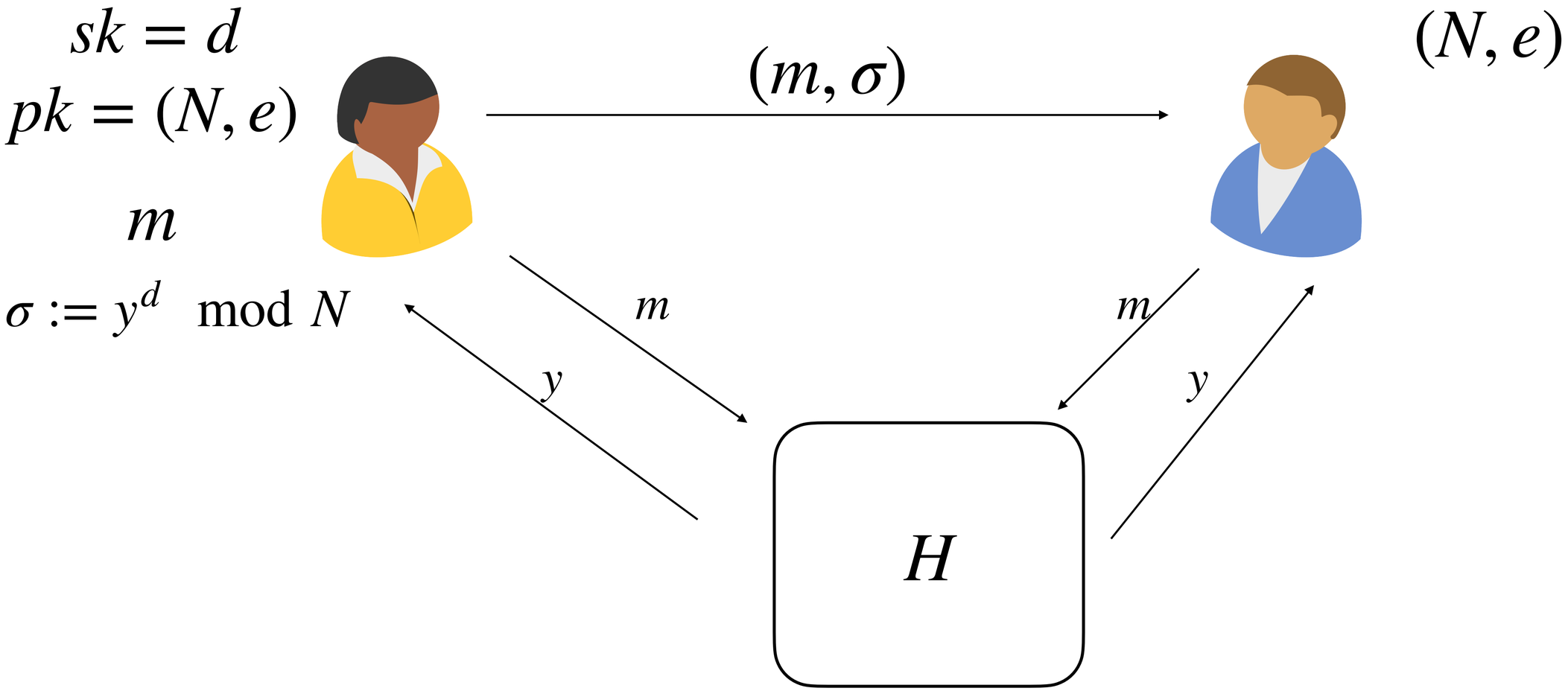
y

H

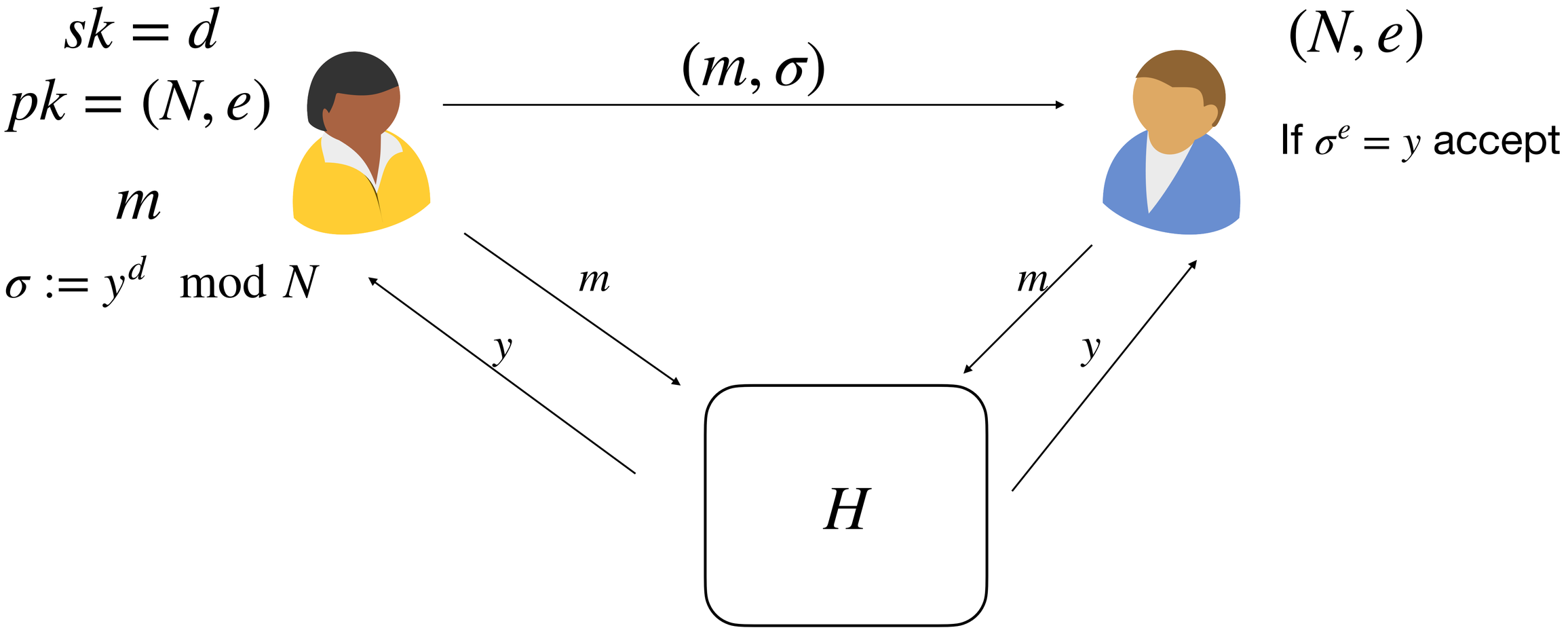
RSA Signatures in the Random Oracle Model



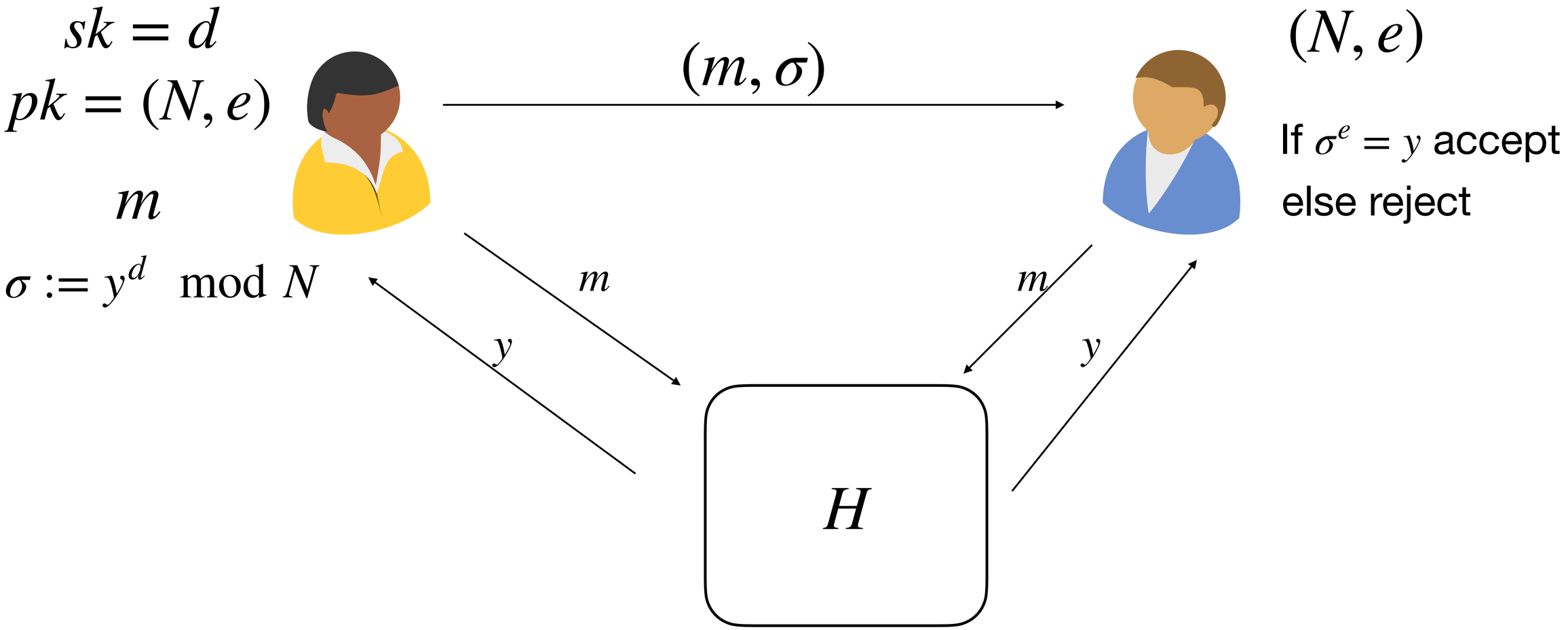
RSA Signatures in the Random Oracle Model



RSA Signatures in the Random Oracle Model



RSA Signatures in the Random Oracle Model



Digital Signature Construction

Digital Signature Construction

FDH RSA Signatures

Digital Signature Construction

FDH RSA Signatures

- KeyGen(1^λ)

Digital Signature Construction

FDH RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)

Digital Signature Construction

FDH RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e); \quad sk := (N, d)$

Digital Signature Construction

FDH RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$

Digital Signature Construction

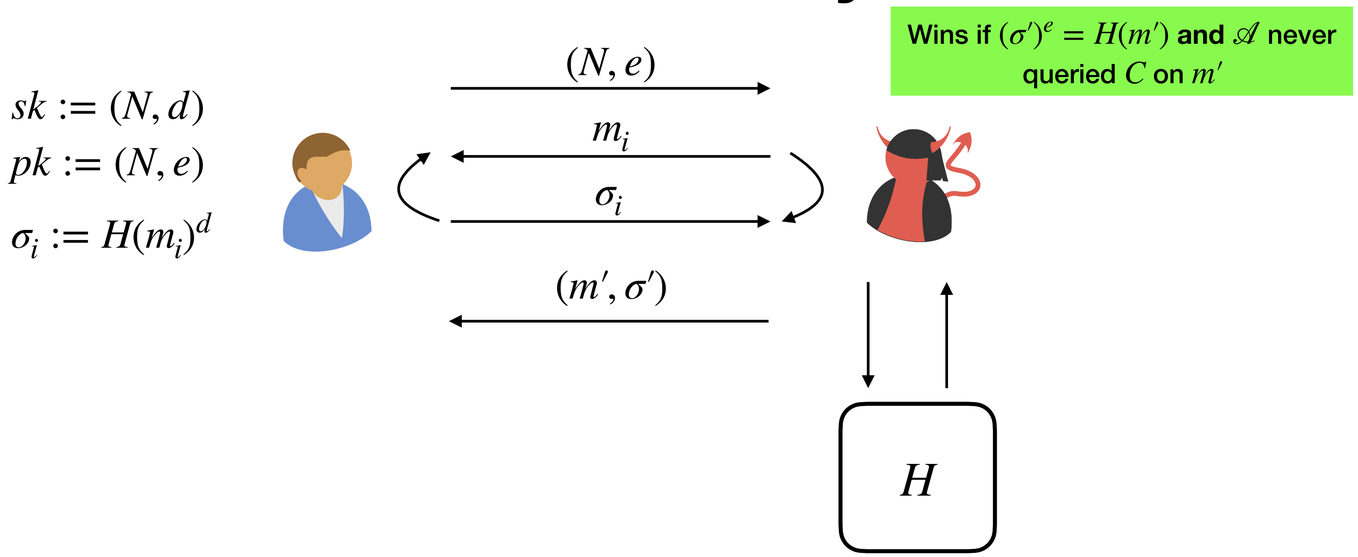
FDH RSA Signatures

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

Proof of Security

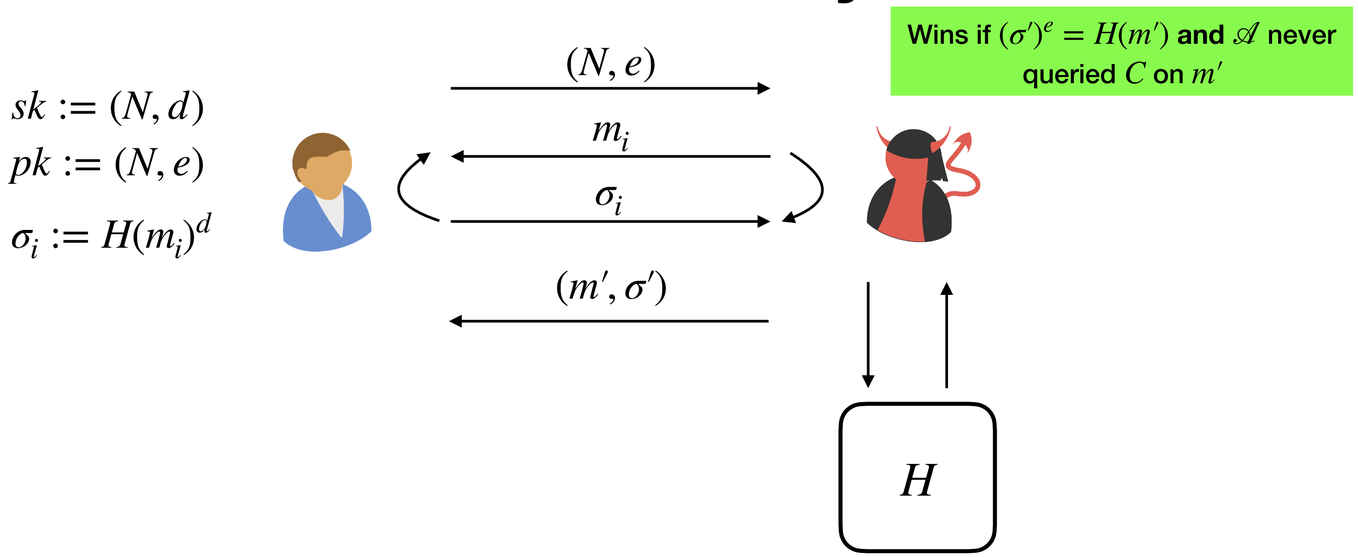
- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$

Proof of Security



- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

Proof of Security



Wins if $(\sigma')^e = H(m')$ **and** $\mathcal{A}$ never queried C on m'

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$

Claim: If RSA holds, then FDH RSA Signatures are secure

Recap Recap: RSA

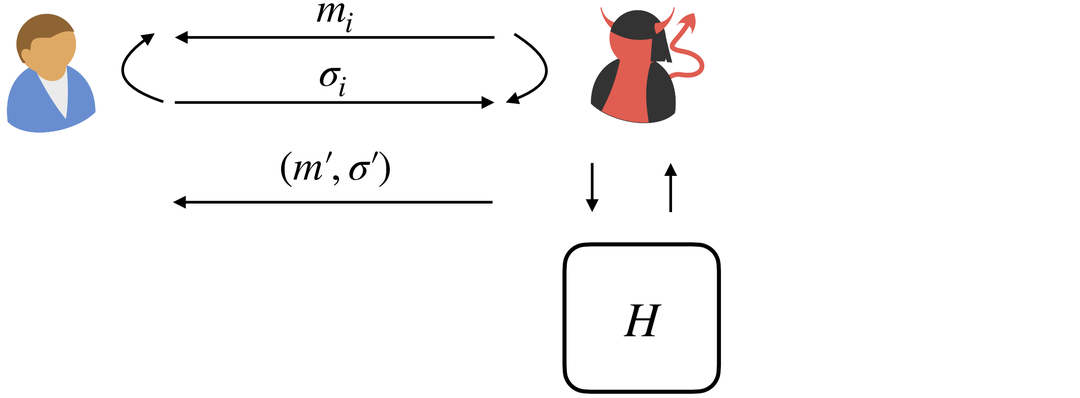
$$\Pr\left[\mathcal{A}(N, e, x^e \mod N) = x : \begin{array}{c}(p,q) \leftarrow \mathsf{GenRSA}(1^\lambda) \\ N := pq \\ e \xleftarrow{\$} \mathbb{A}^\times_{\varphi(N)} \\ x \xleftarrow{\$} \mathbb{Z}^\times_N\end{array}\right] \leq \mathsf{negl}(\lambda)$$

Proof of Security

$sk := (N, d)$

$pk := (N, e)$

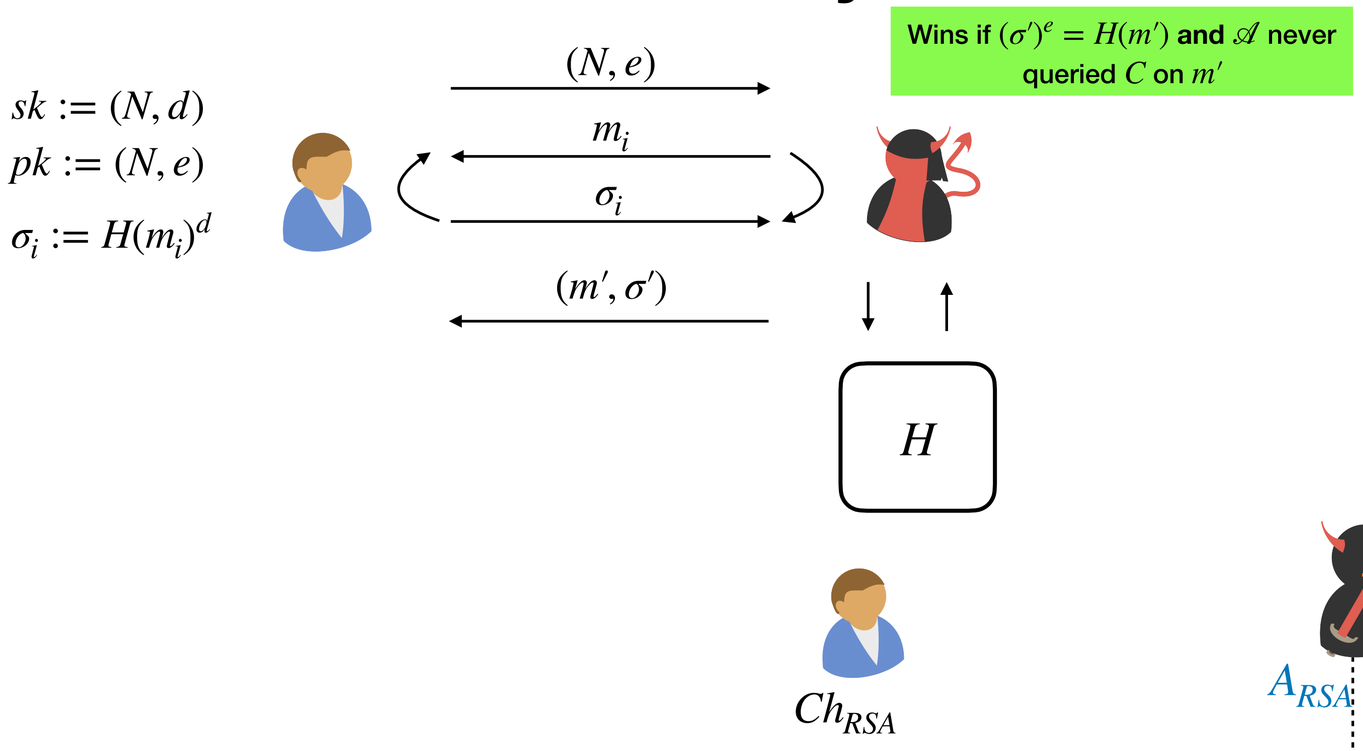
$\sigma_i := H(m_i)^d$



Wins if $(\sigma')^e = H(m')$ **and** $\mathcal{A}$ never queried C on m'

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$

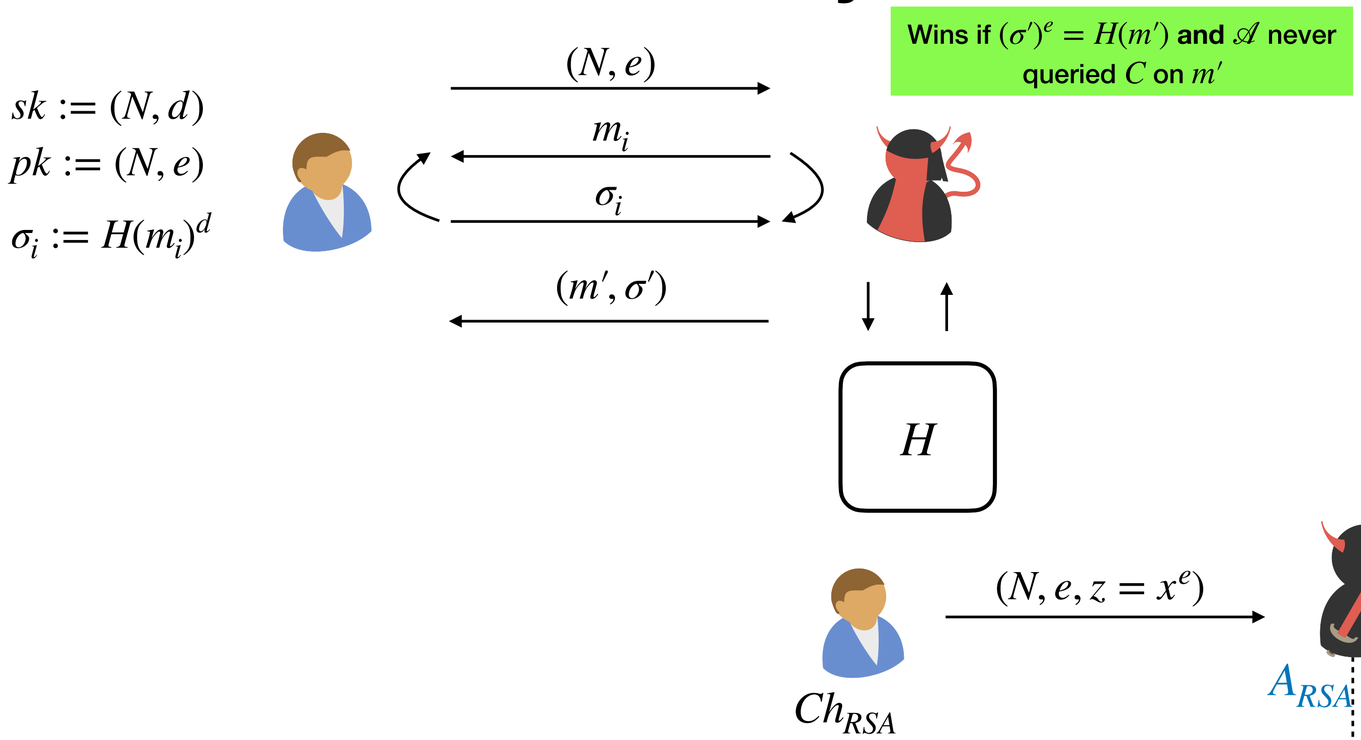
Proof of Security

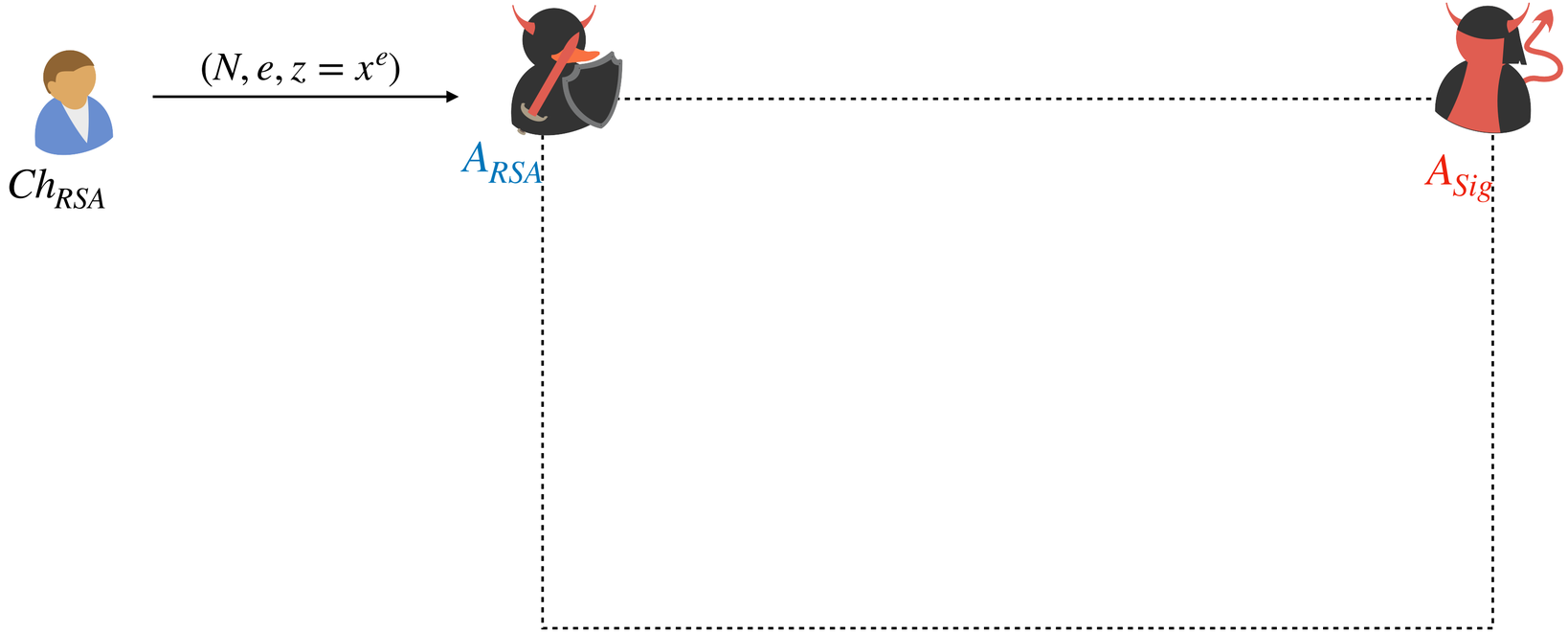


- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

Proof of Security

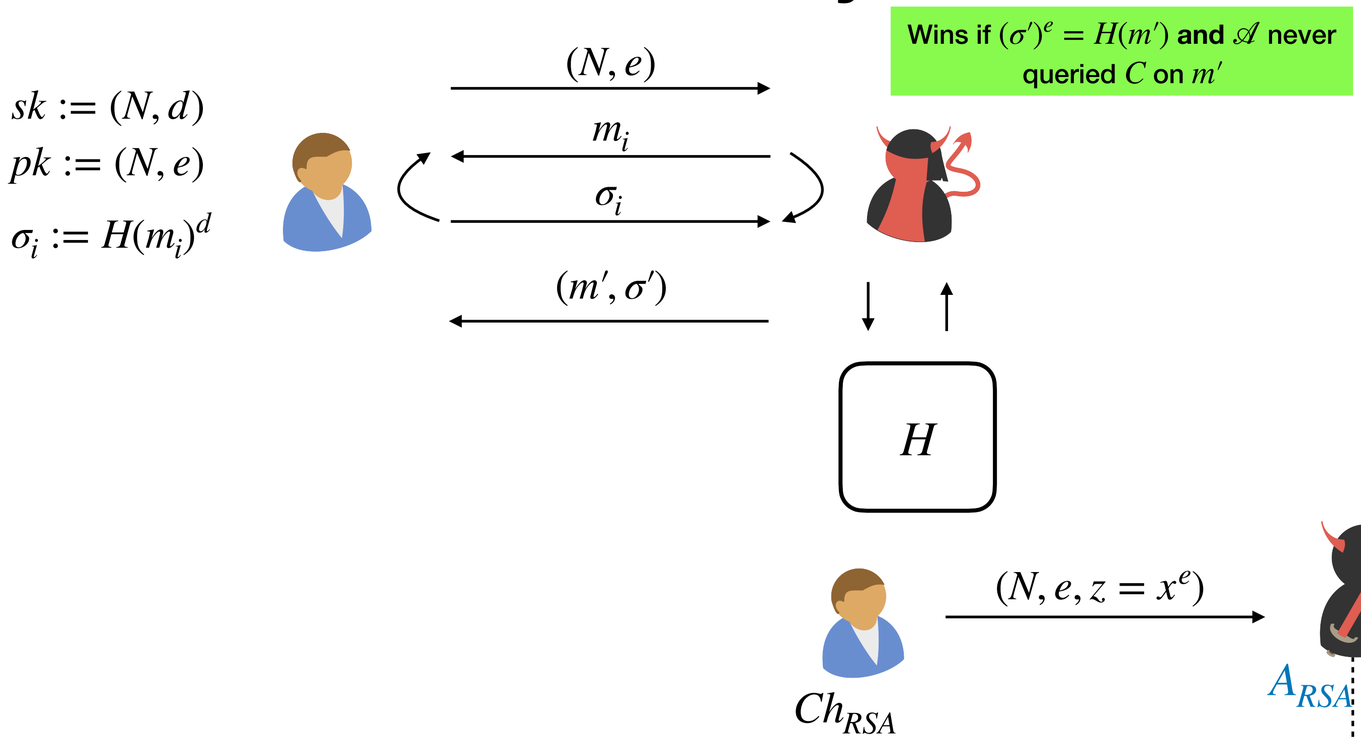
- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

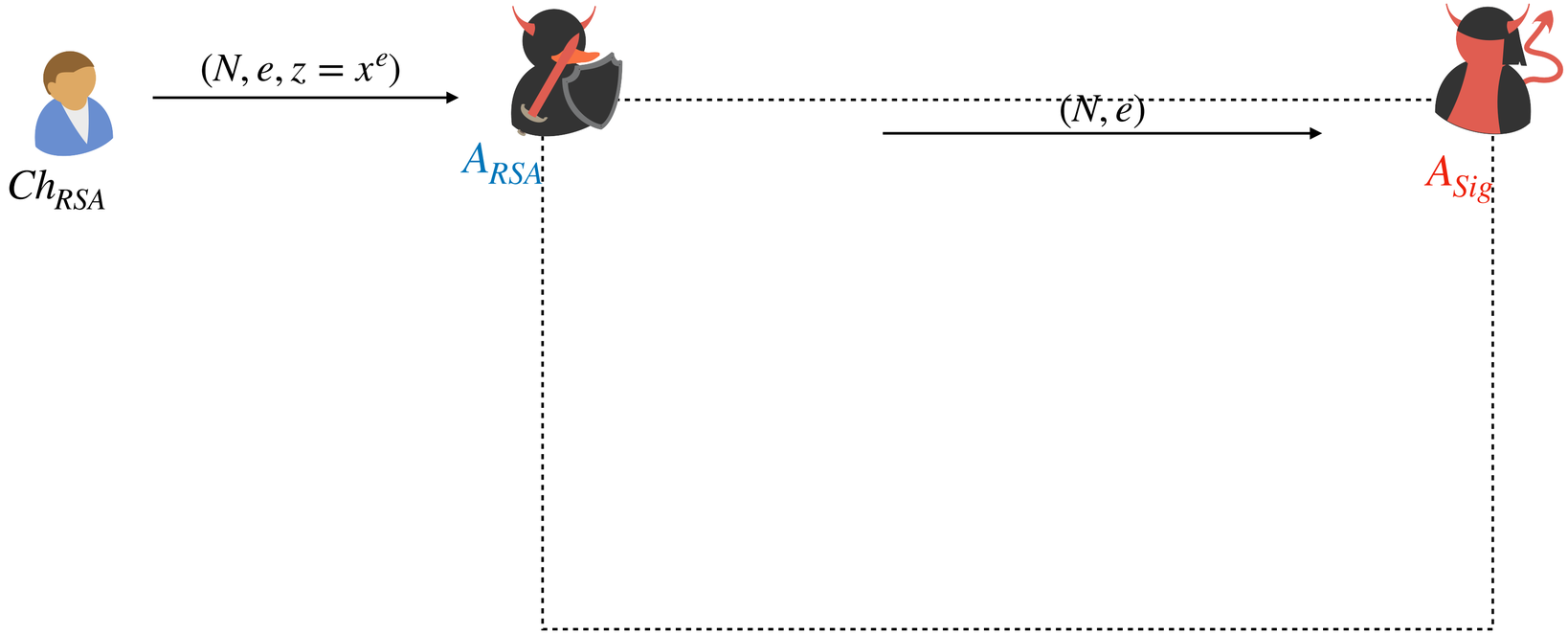




Proof of Security

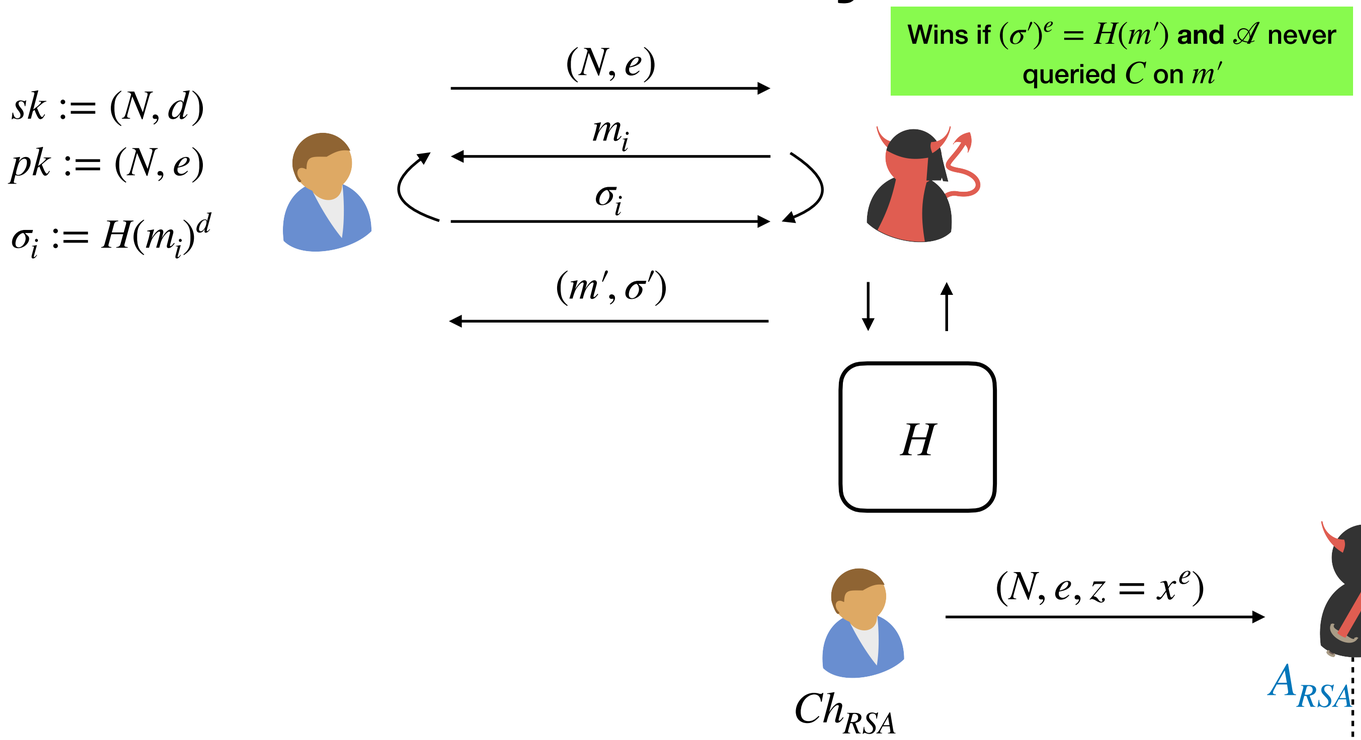
- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

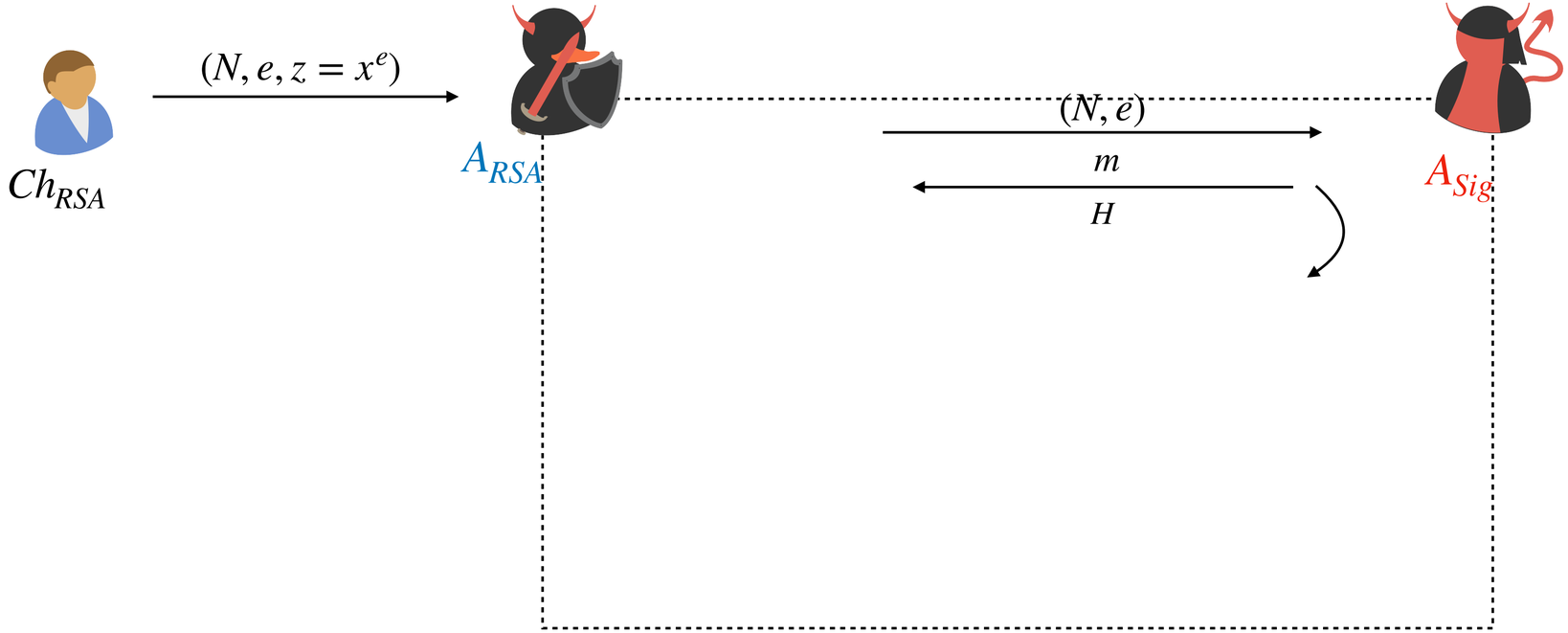




Proof of Security

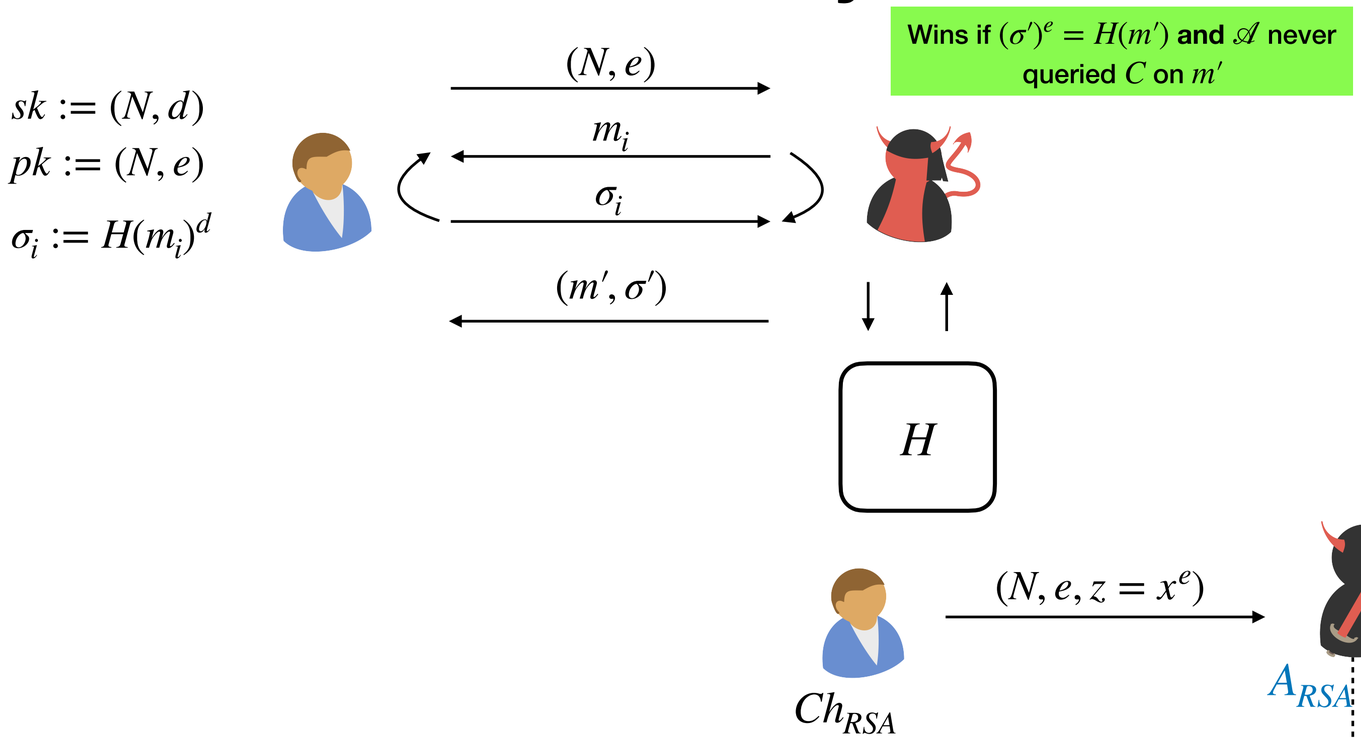
- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

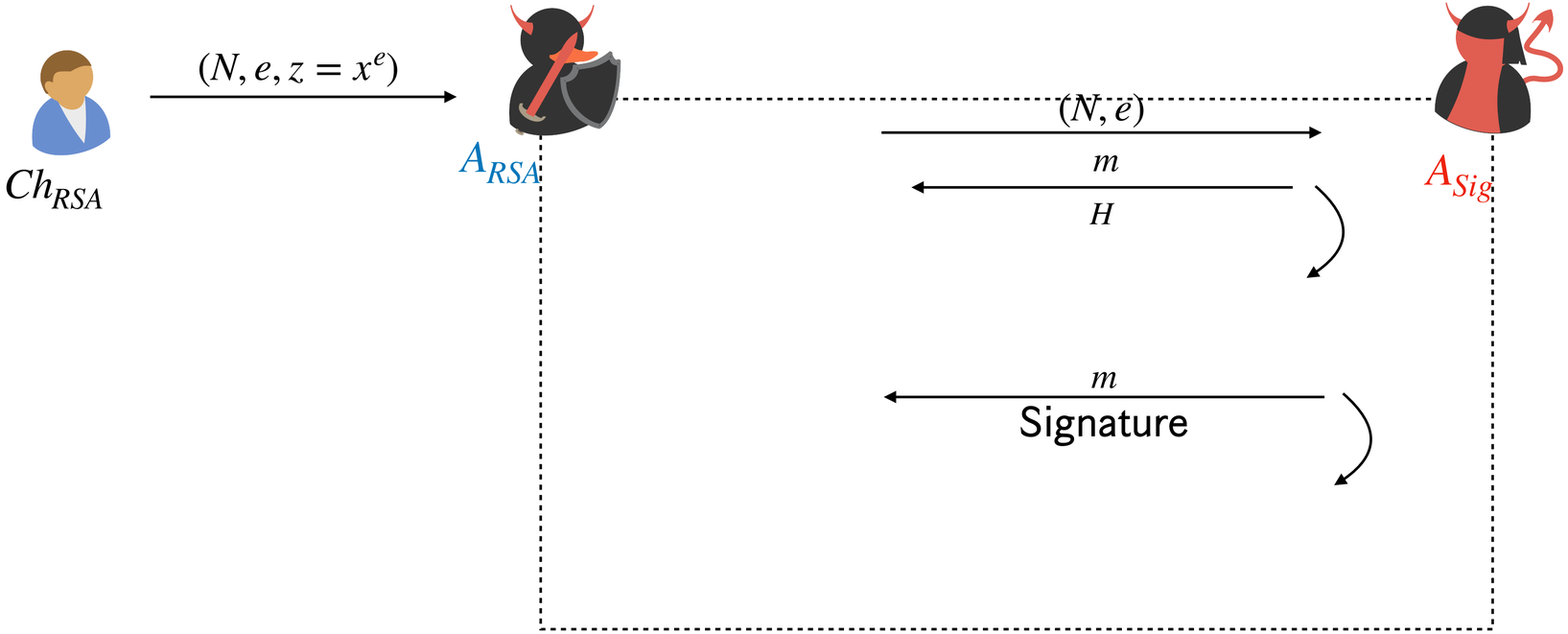




Proof of Security

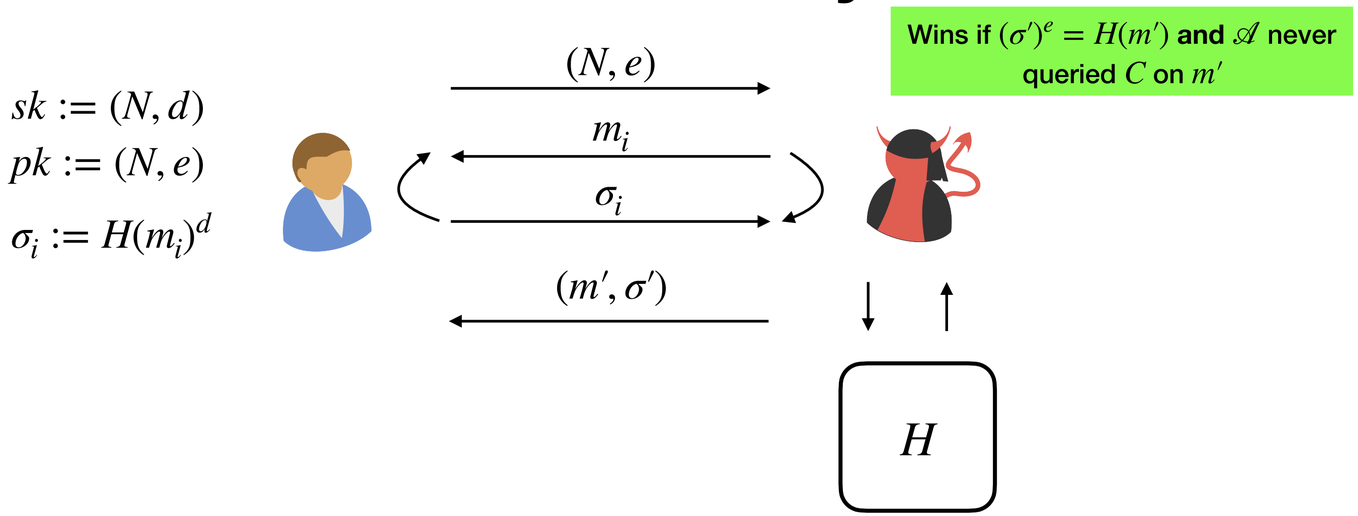
- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

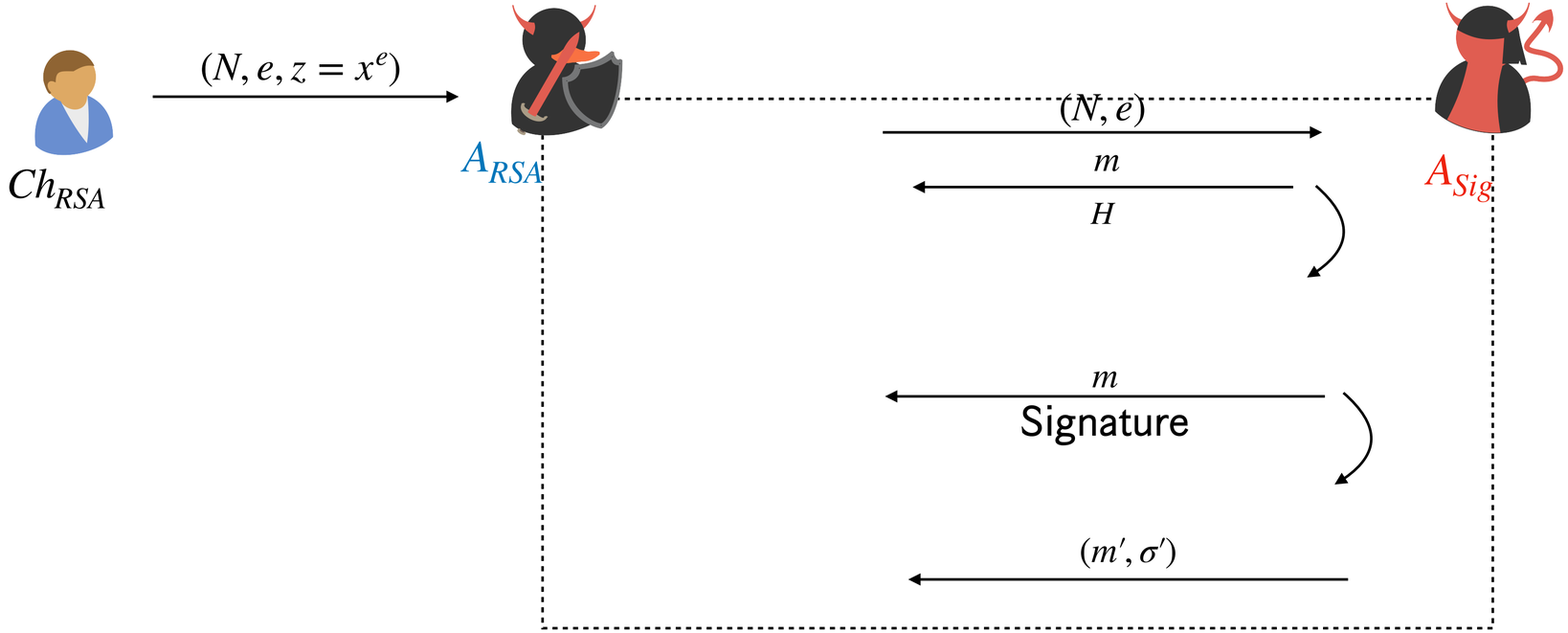




Proof of Security

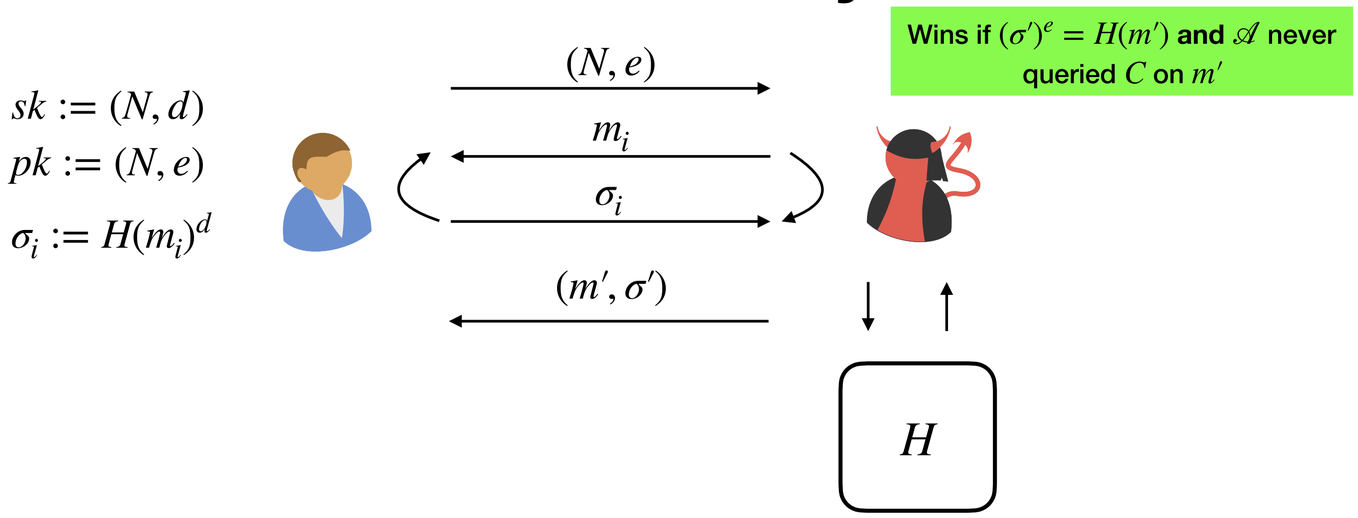
- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

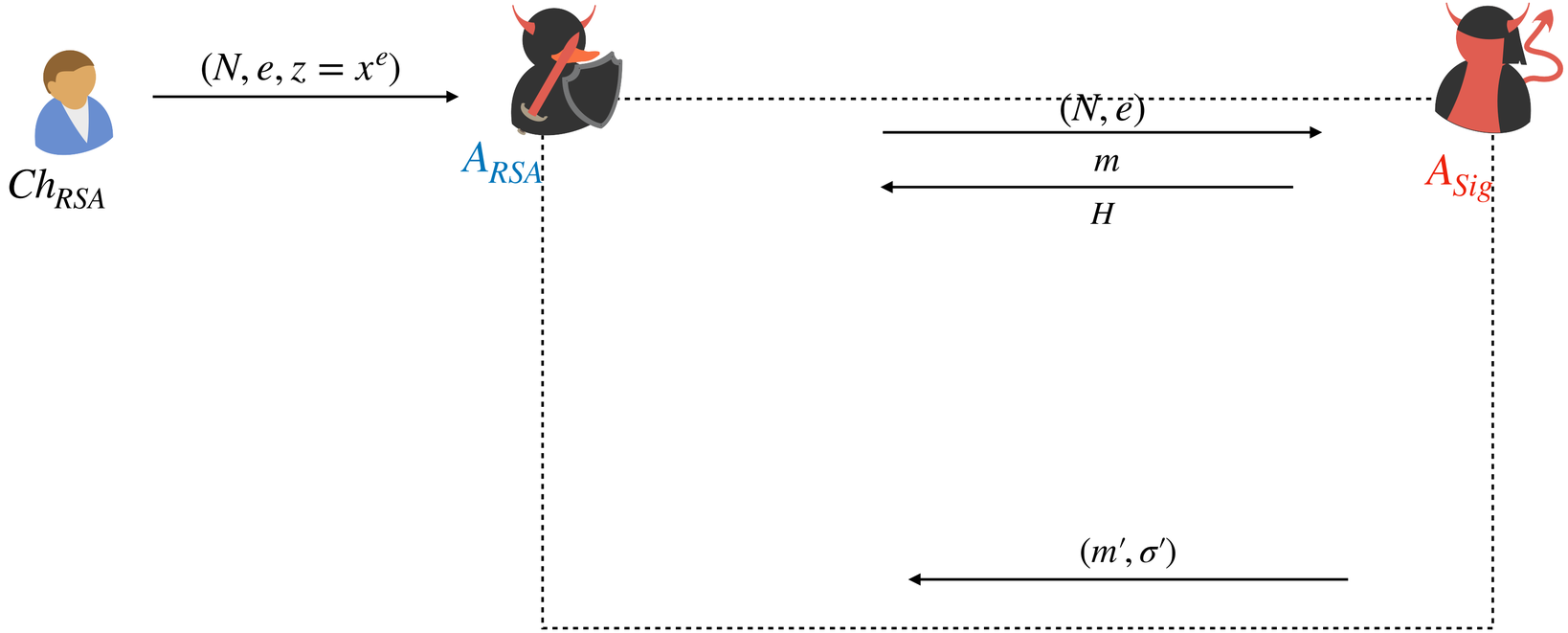




Proof of Security

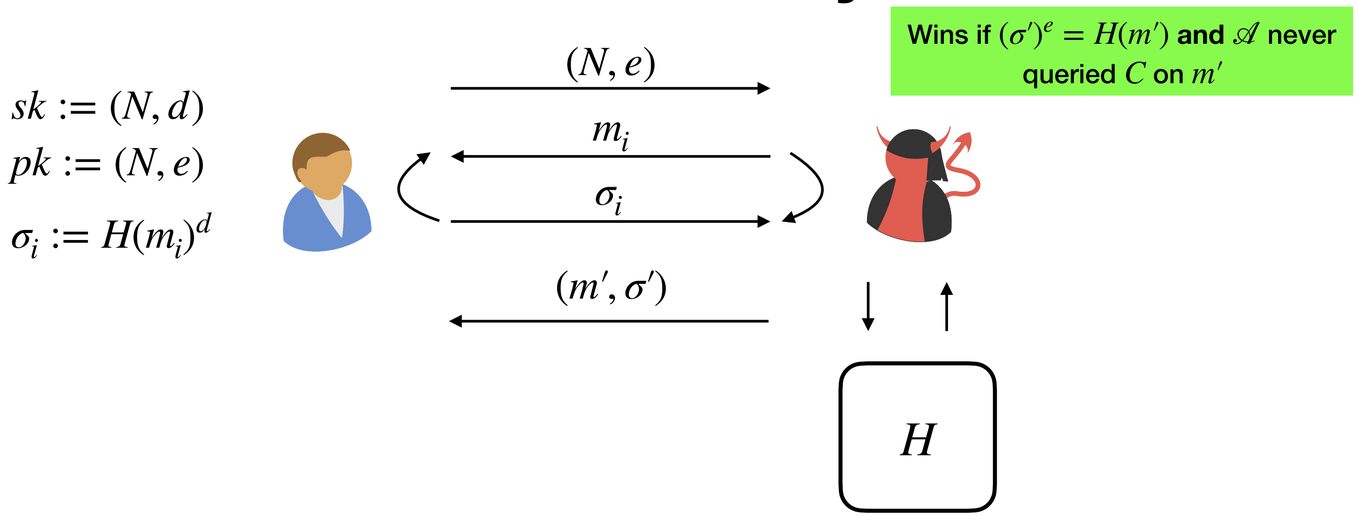
- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

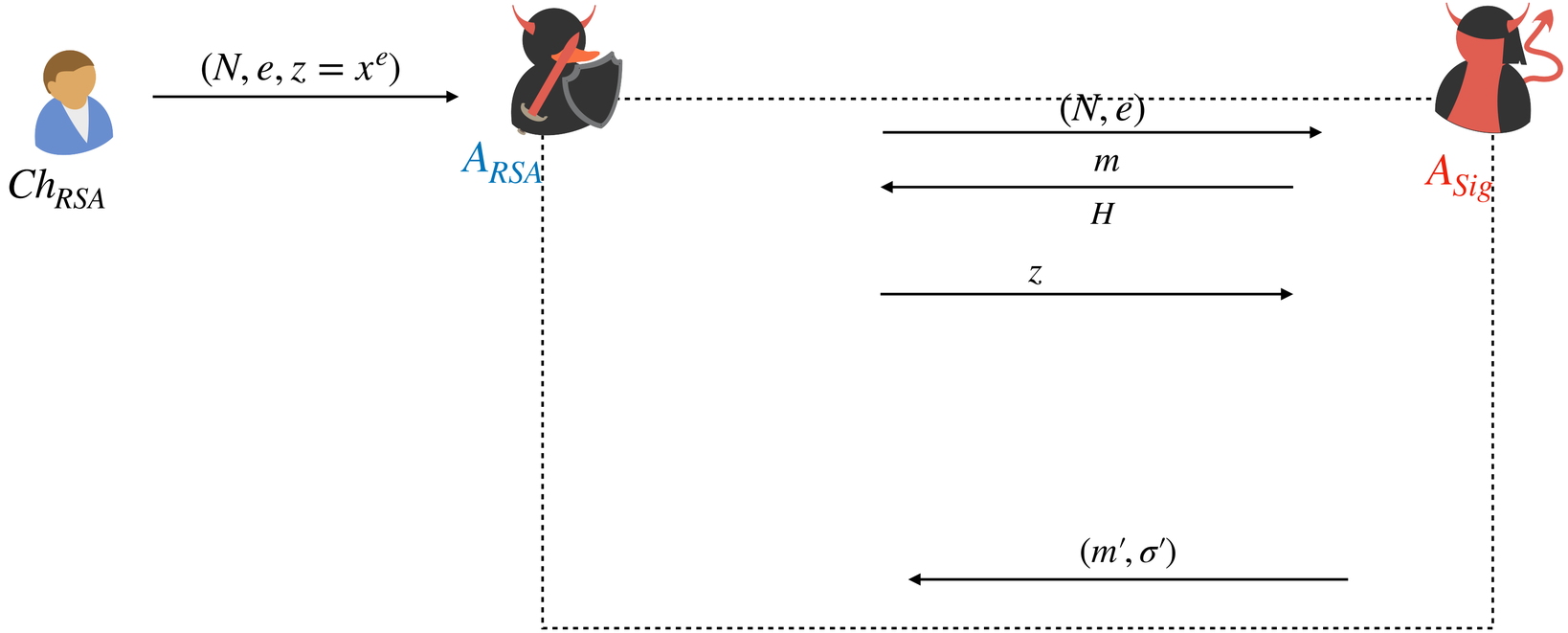




Proof of Security

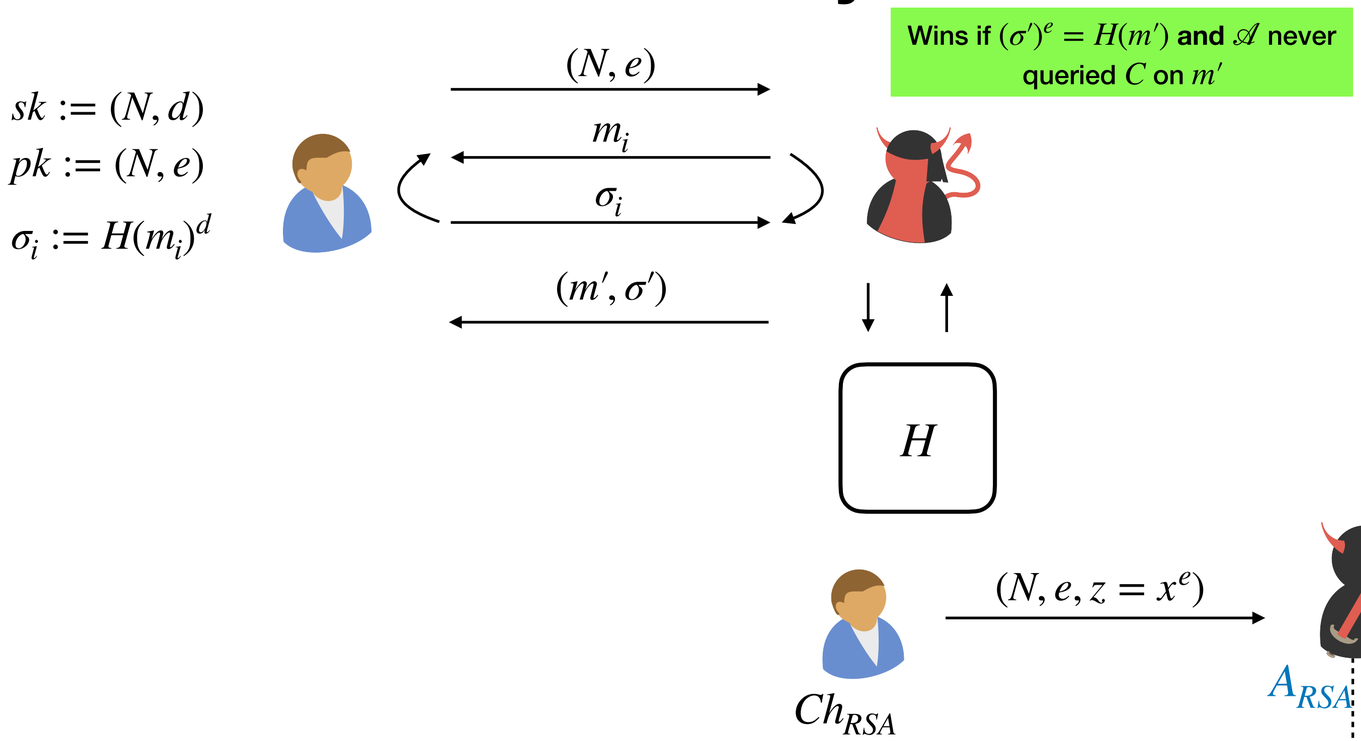
- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

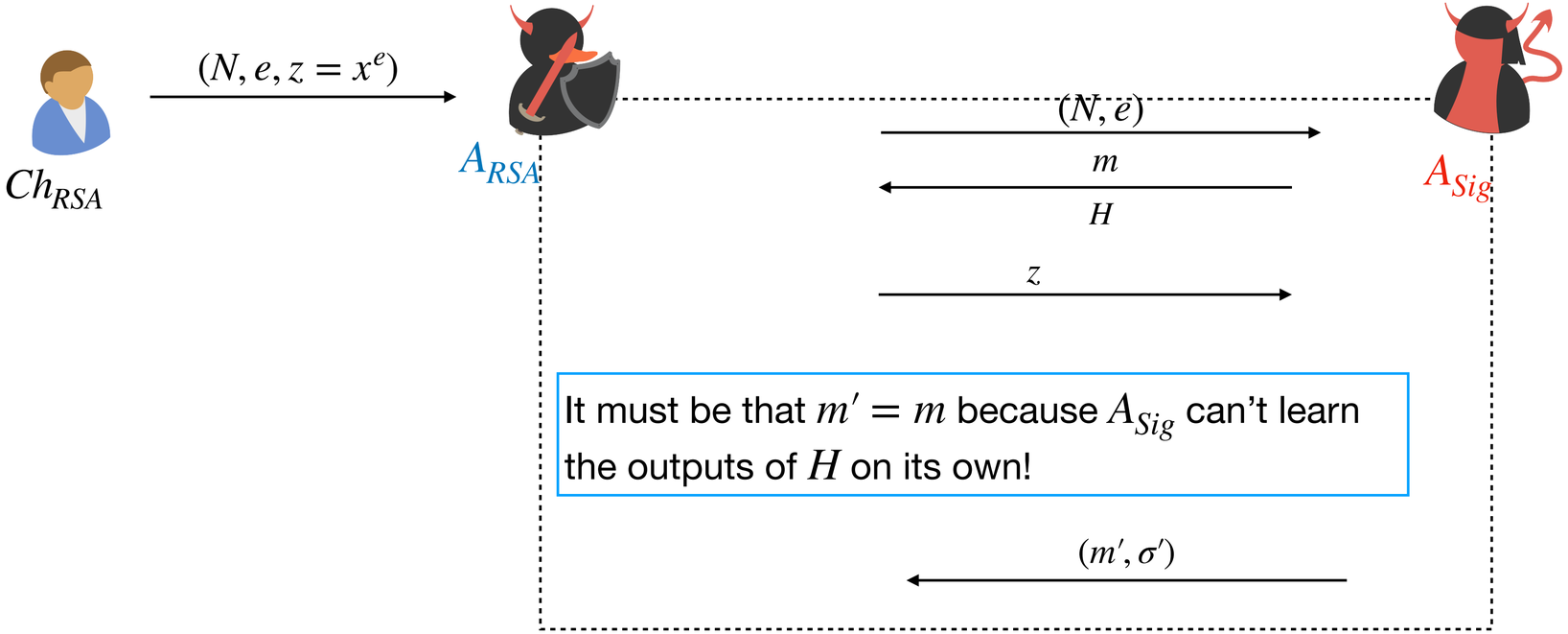




Proof of Security

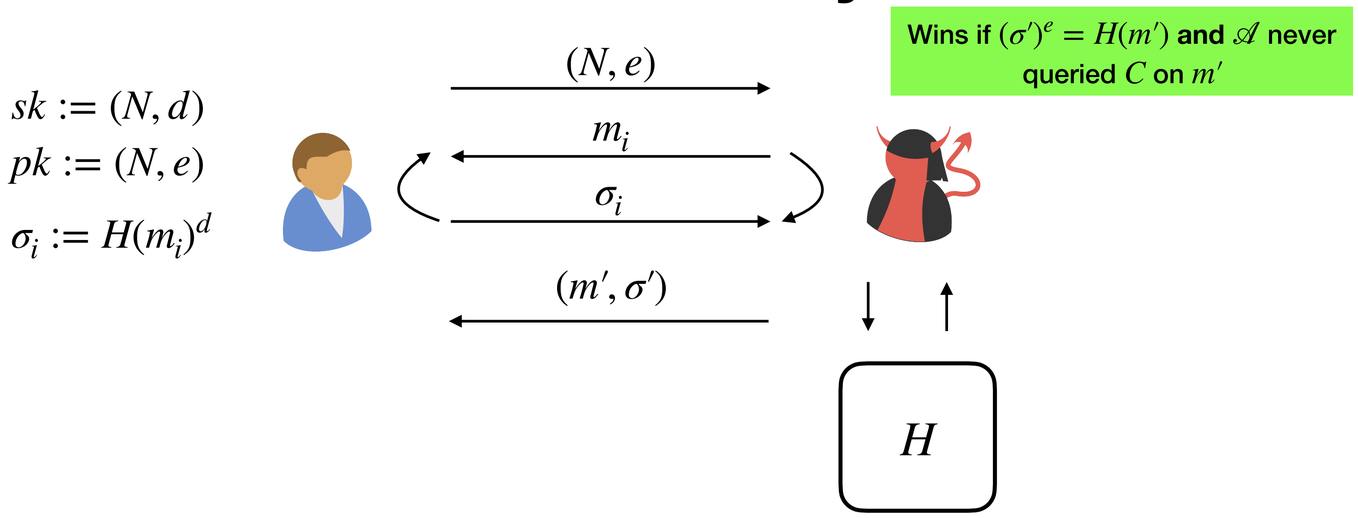
- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e); \quad sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$

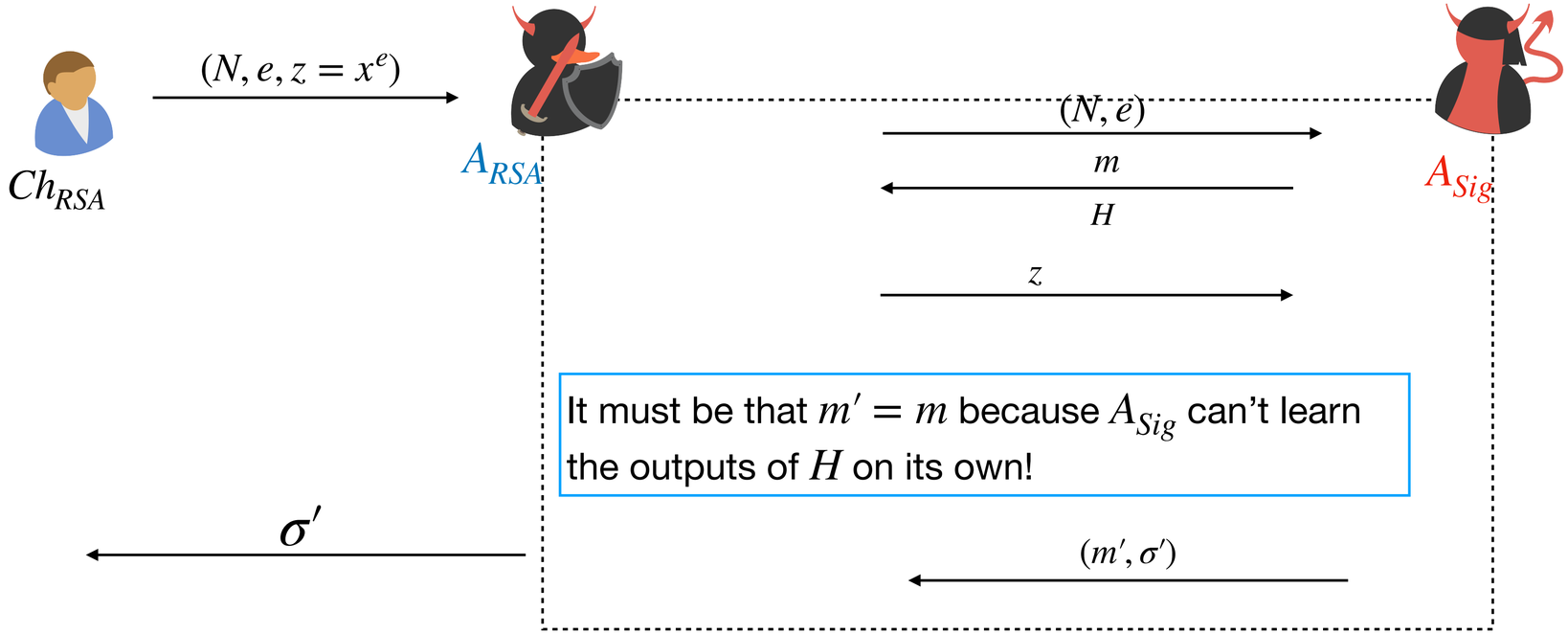




Proof of Security

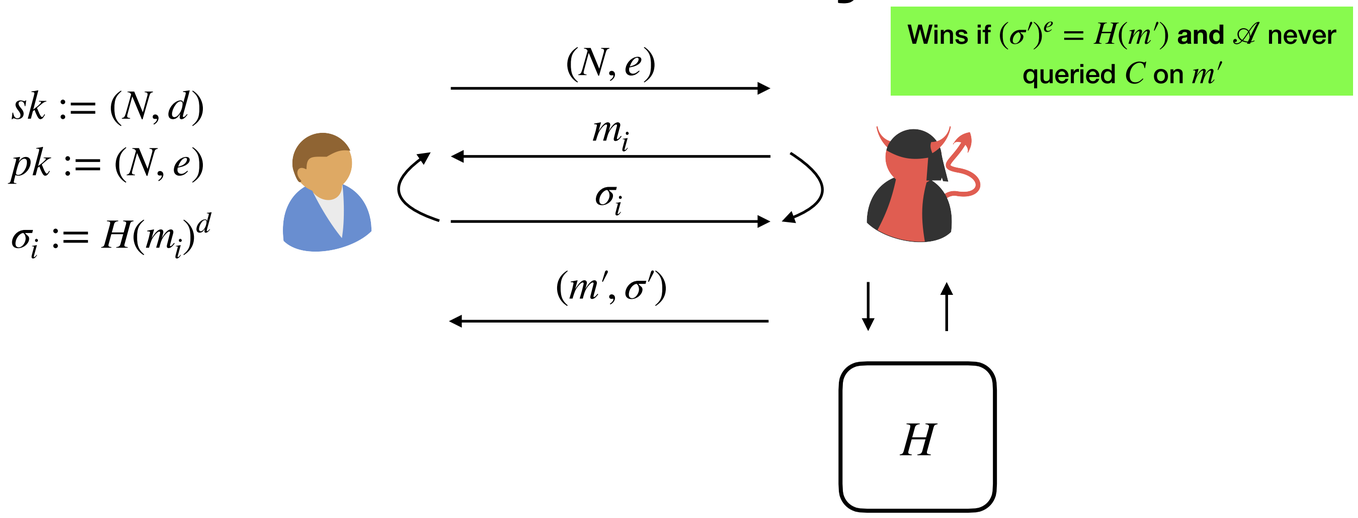
- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

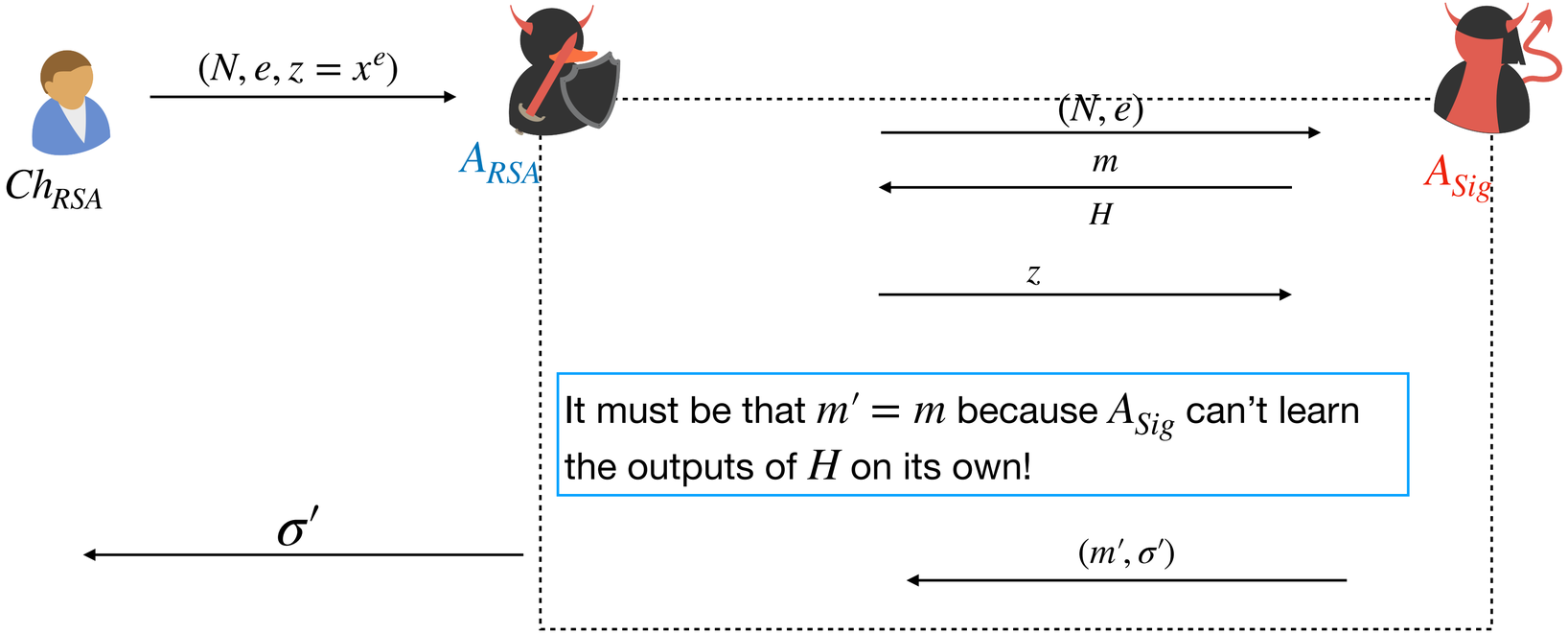




Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e); \quad sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$

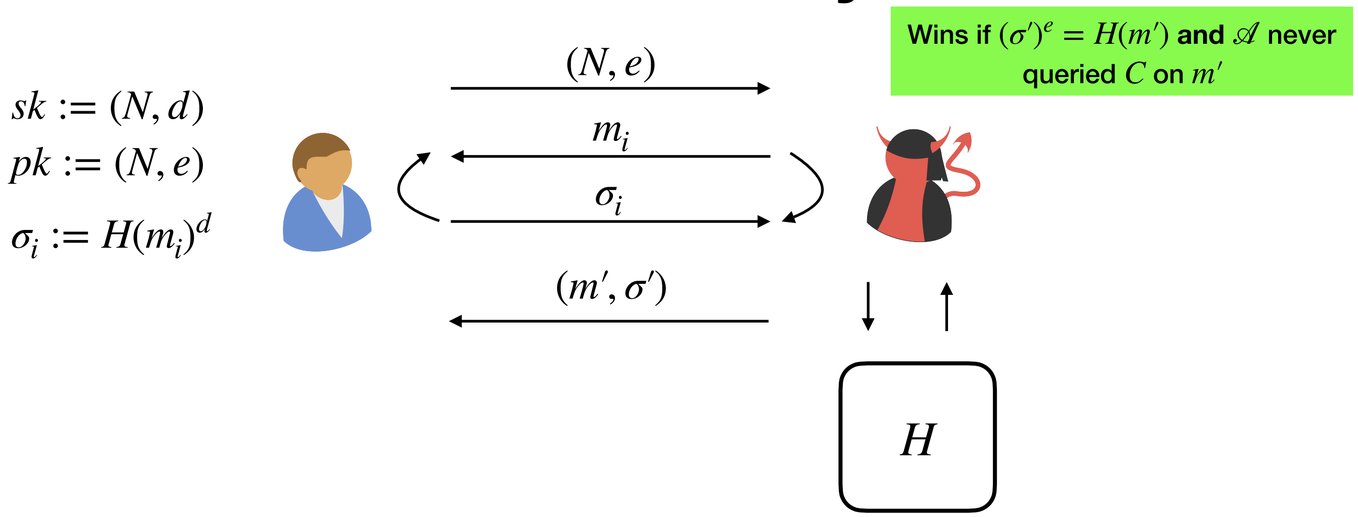


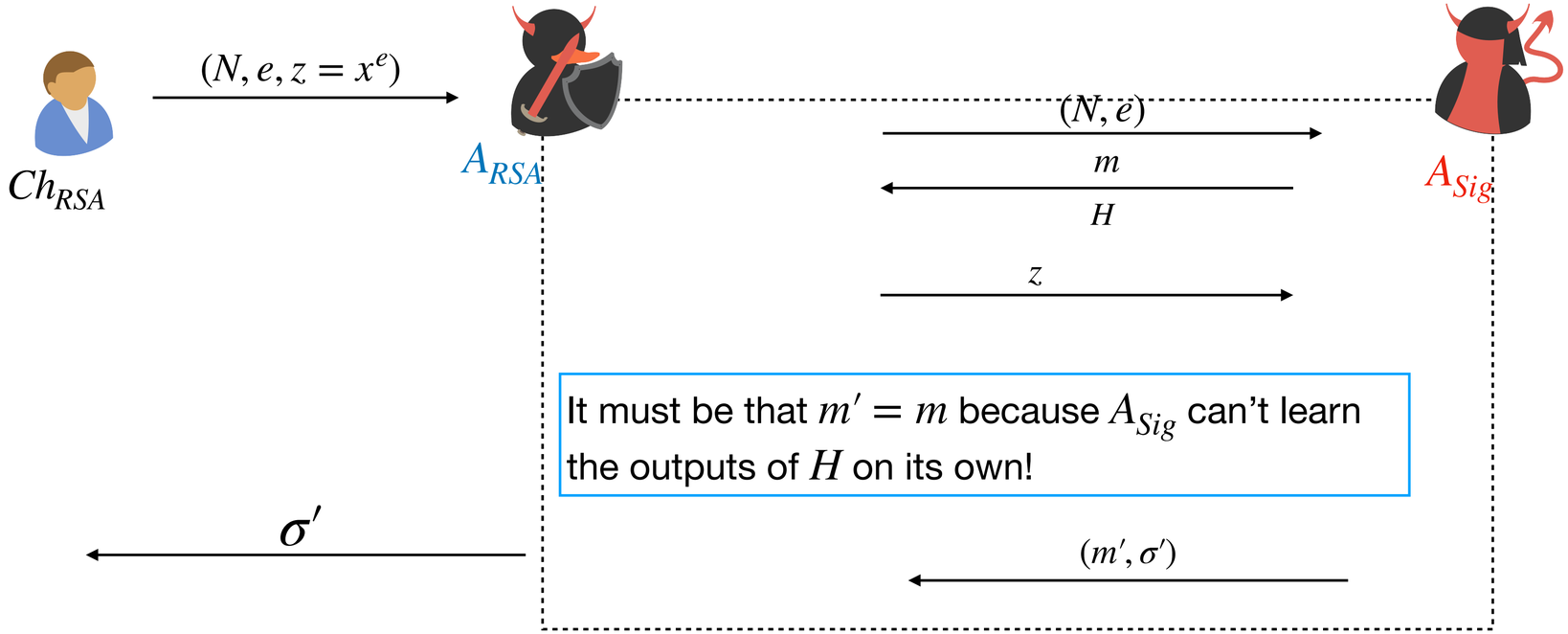


$$(\sigma')^e = H(m')$$
$$(\sigma')^e = z$$
$$(\sigma')^e = x^e$$
$$\sigma' = x$$

Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

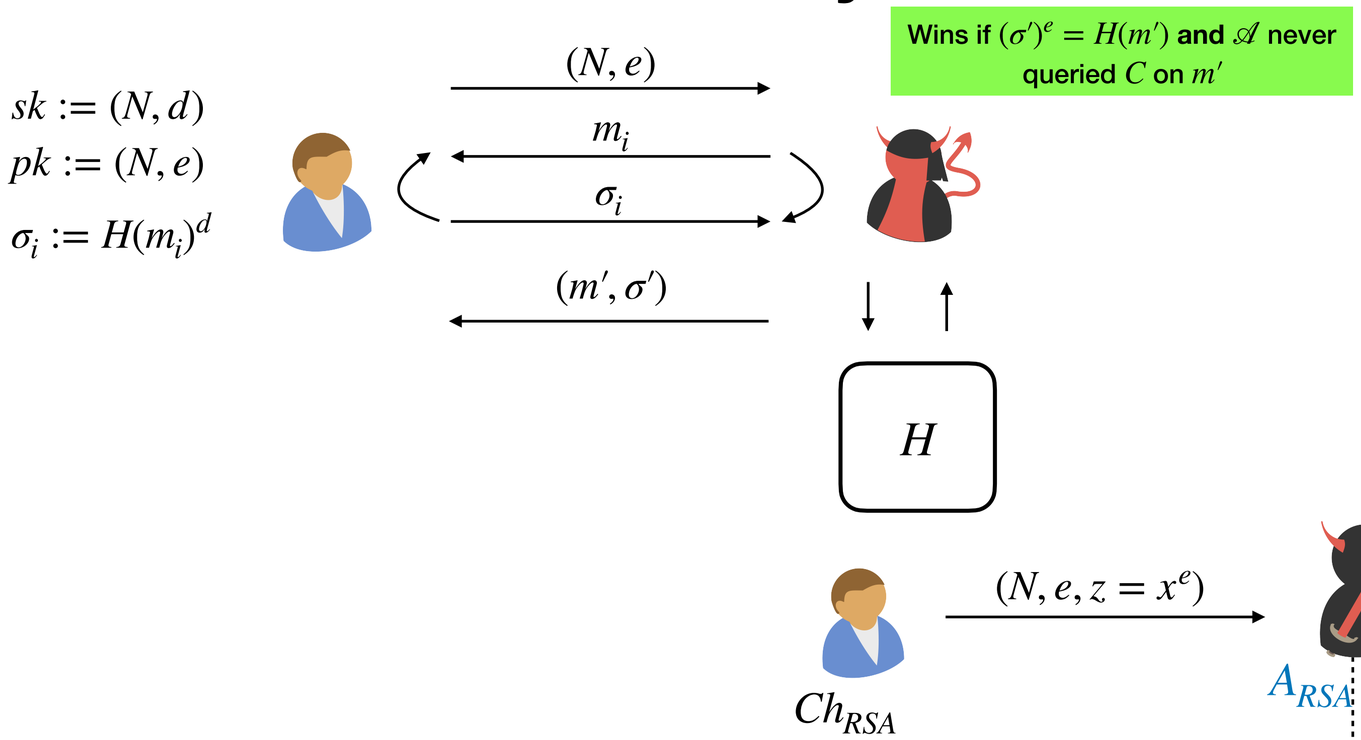


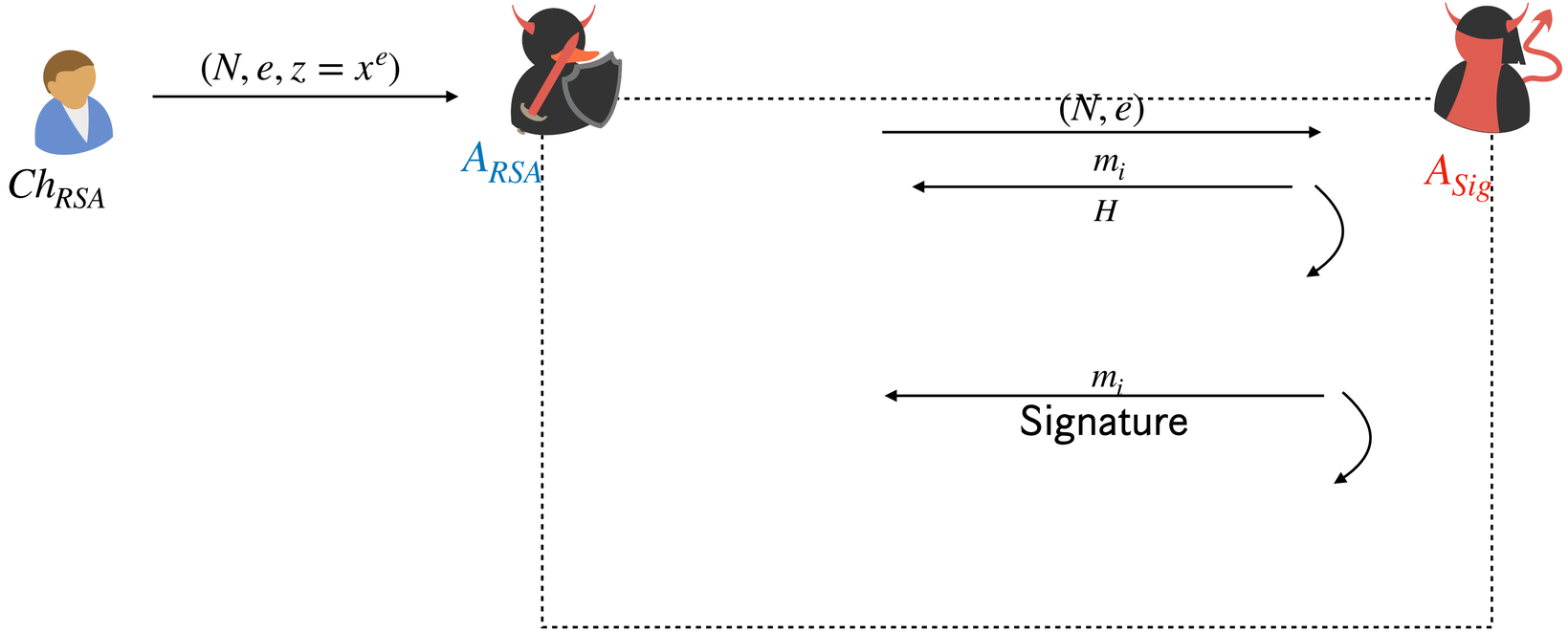


$$(\sigma')^e = H(m')$$
$$(\sigma')^e = z$$
$$(\sigma')^e = x^e$$
$$\sigma' = x$$

Proof of Security

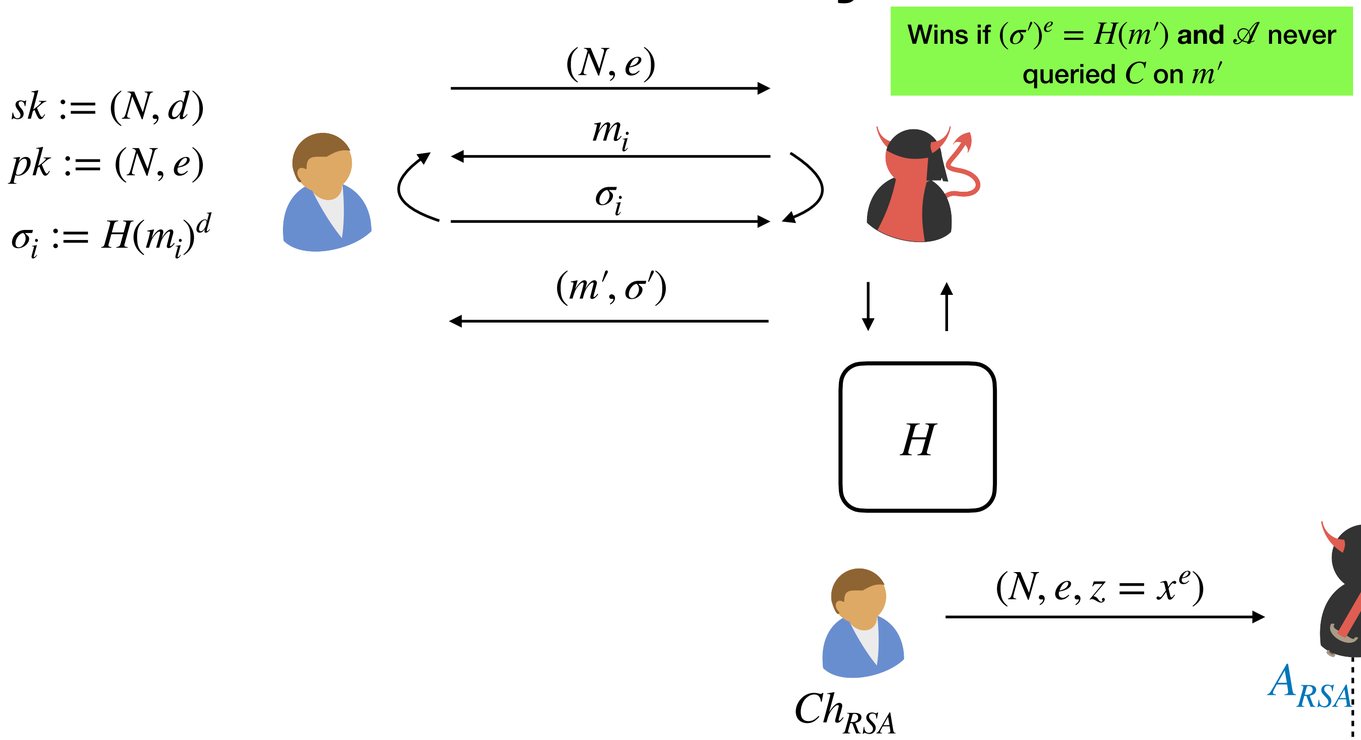
- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

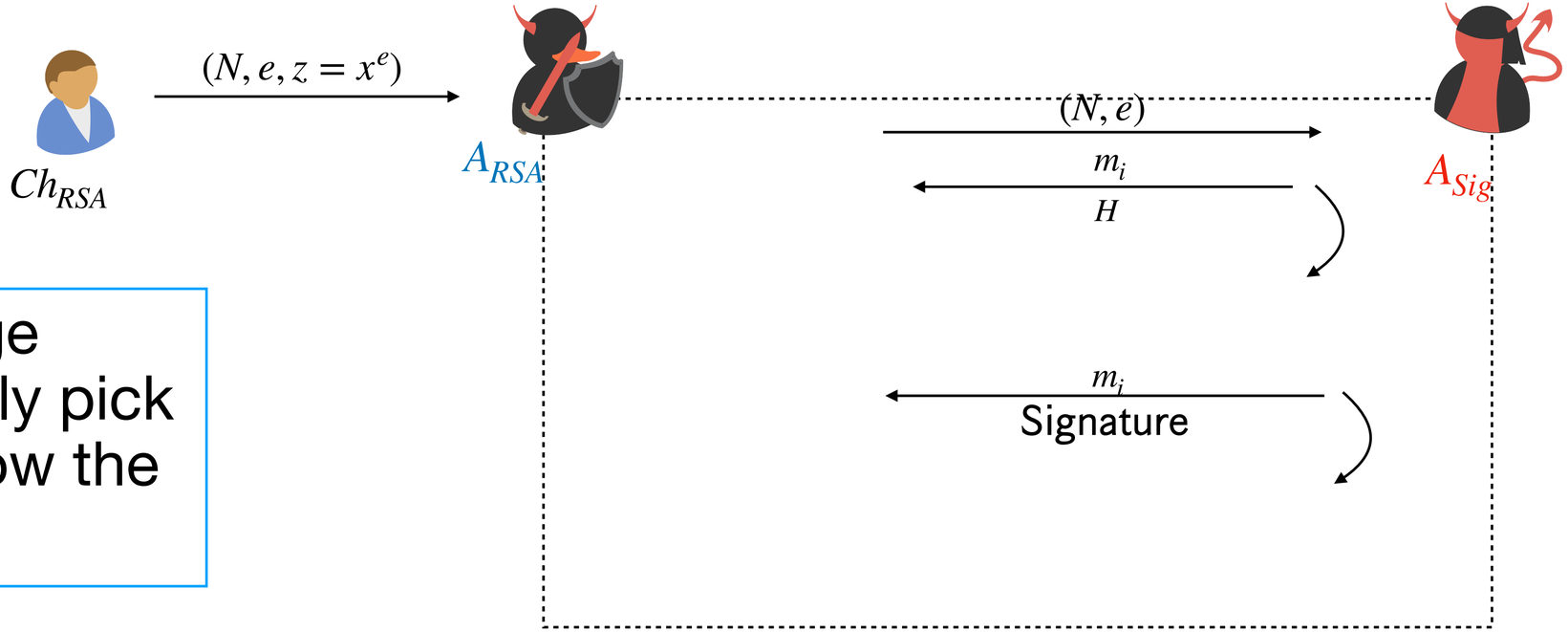




Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

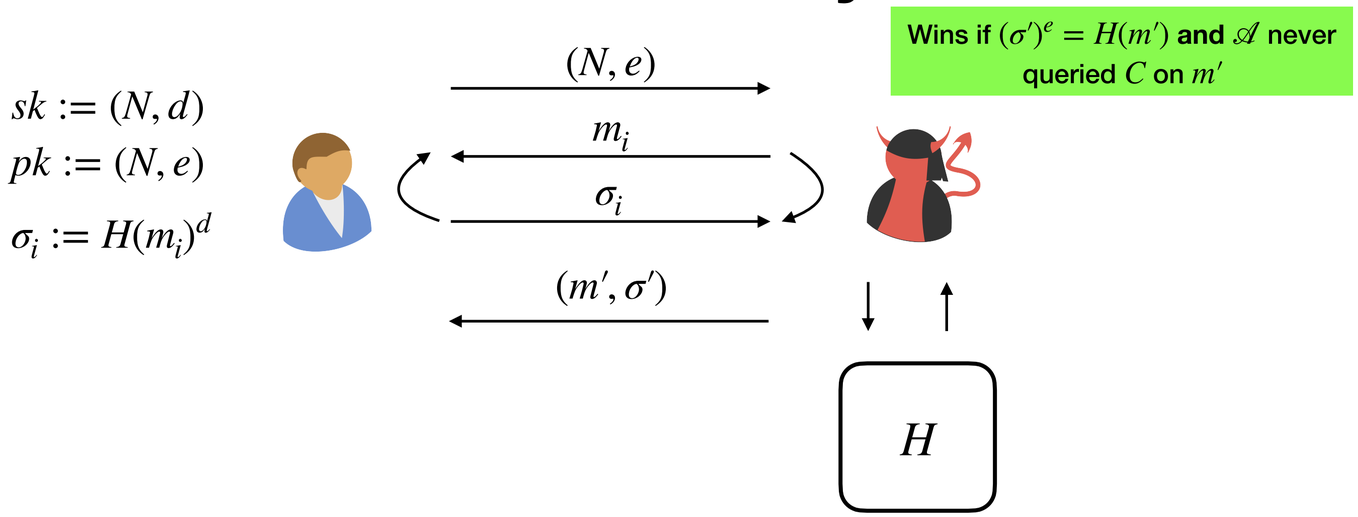


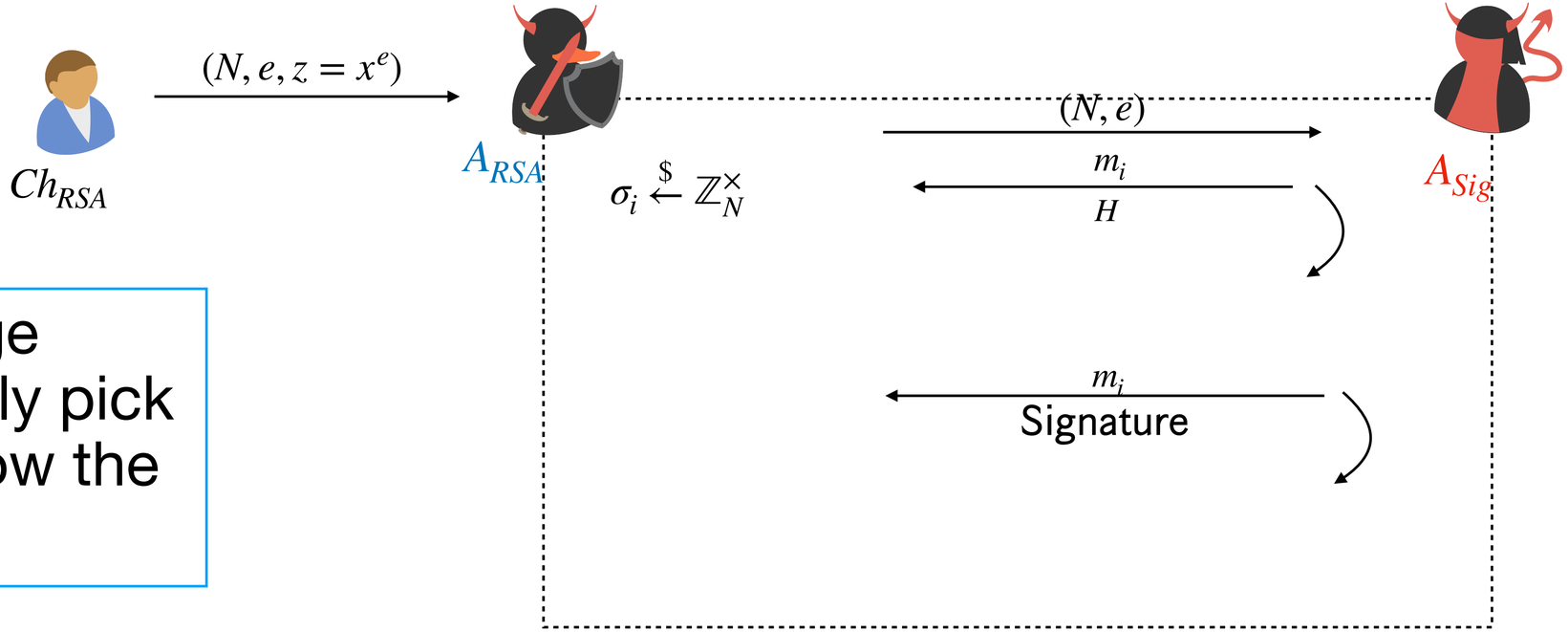


So that we can forge signatures, we'll only pick hash values we know the e-th root of!

Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

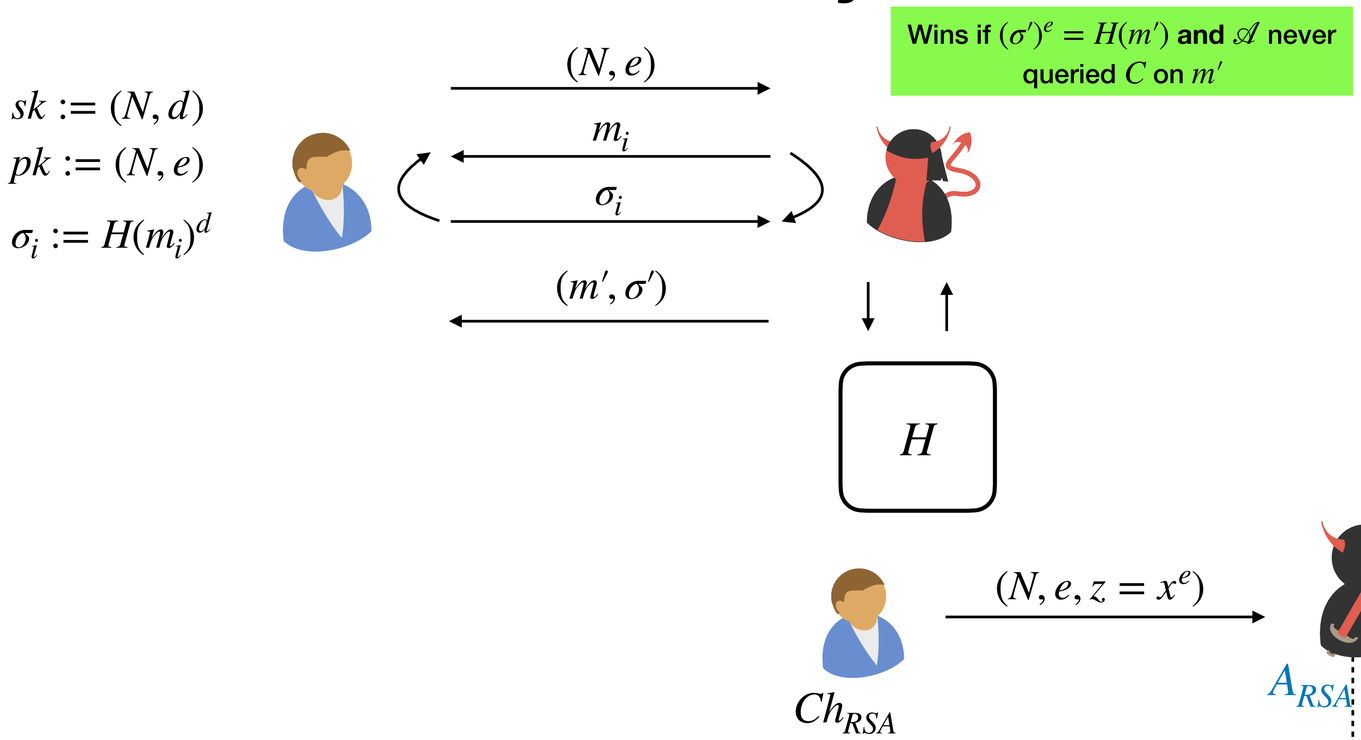


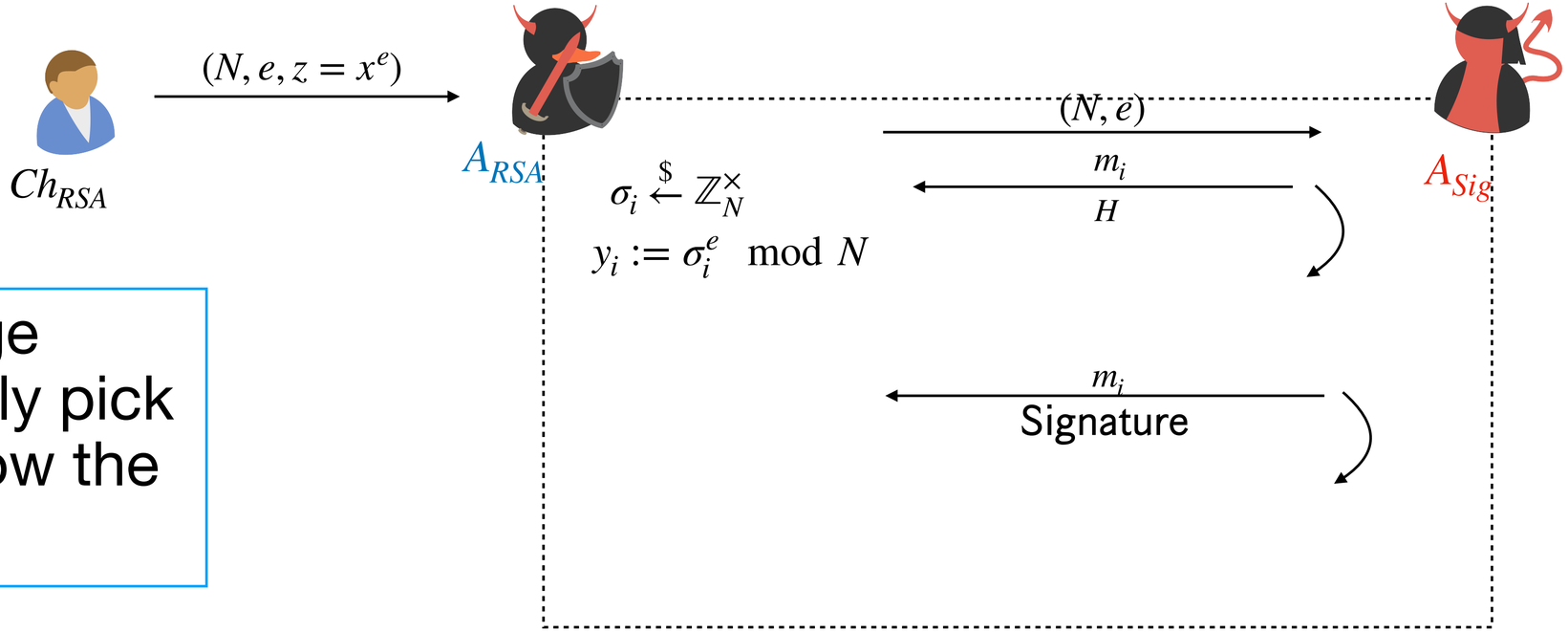


So that we can forge signatures, we'll only pick hash values we know the e-th root of!

Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$

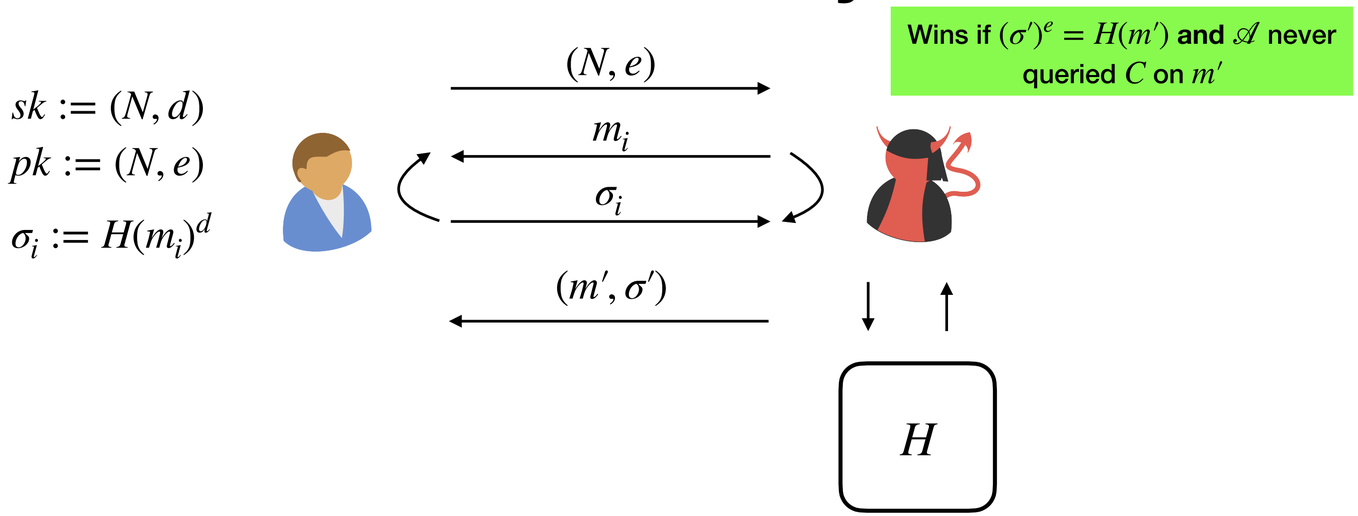


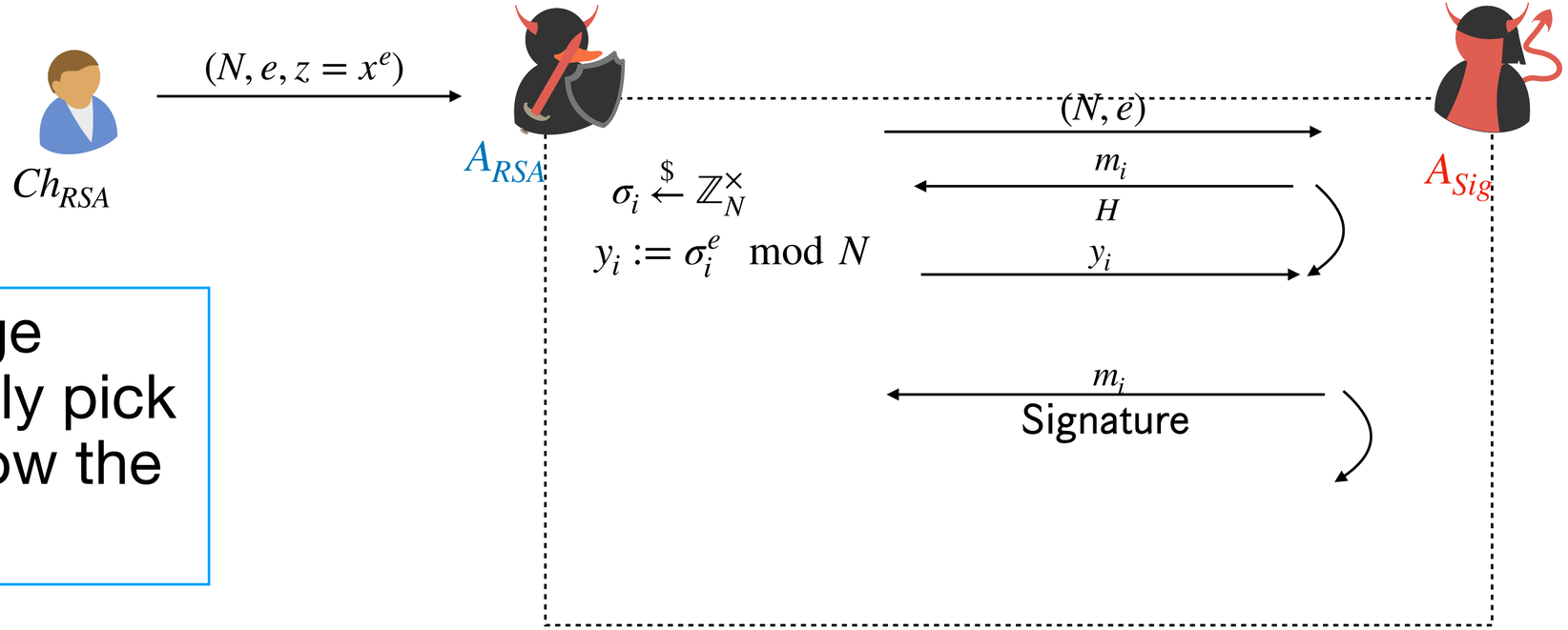


So that we can forge signatures, we'll only pick hash values we know the e-th root of!

Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

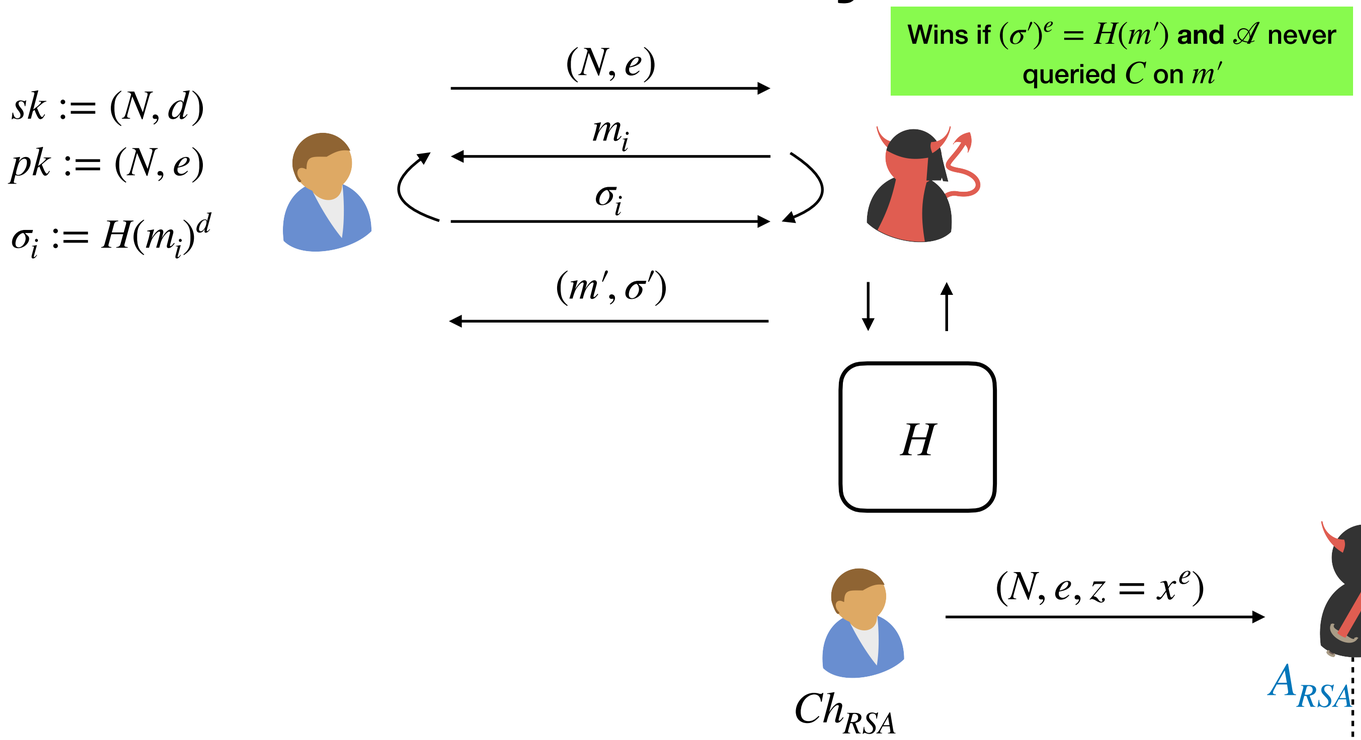


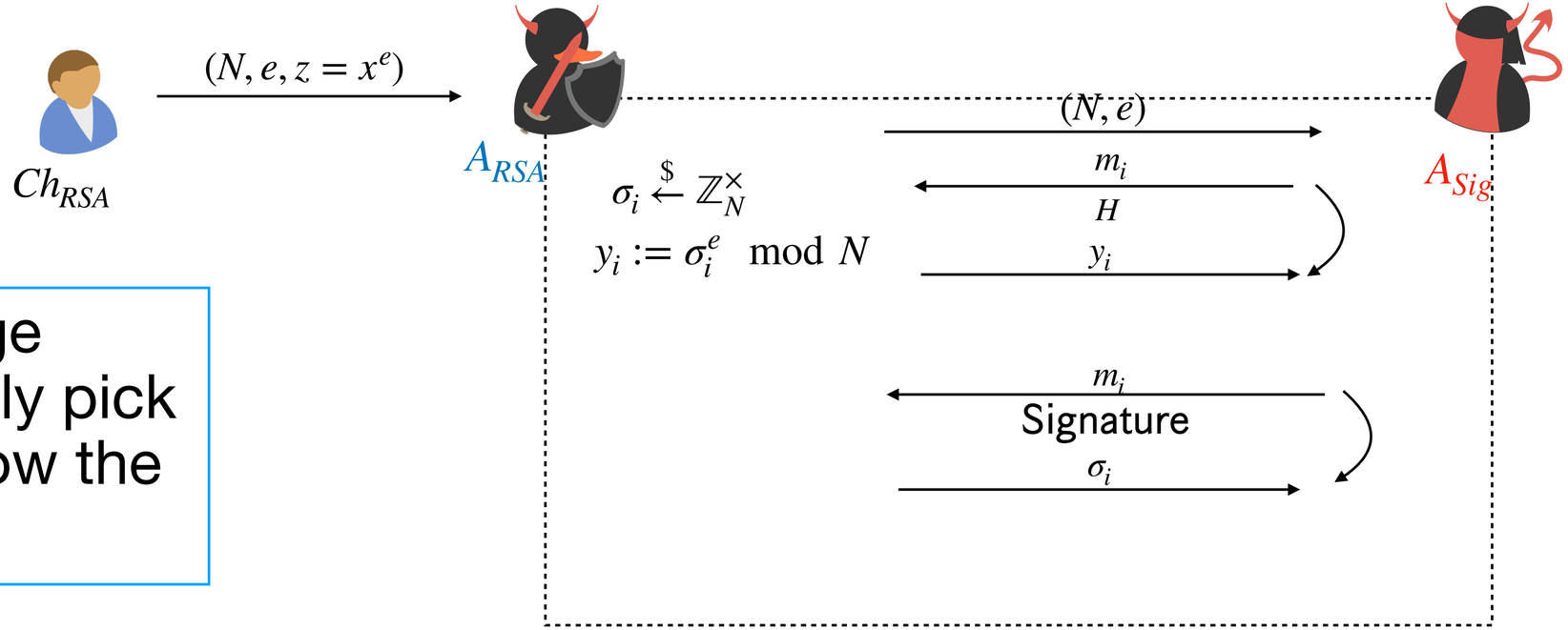


So that we can forge signatures, we'll only pick hash values we know the e-th root of!

Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

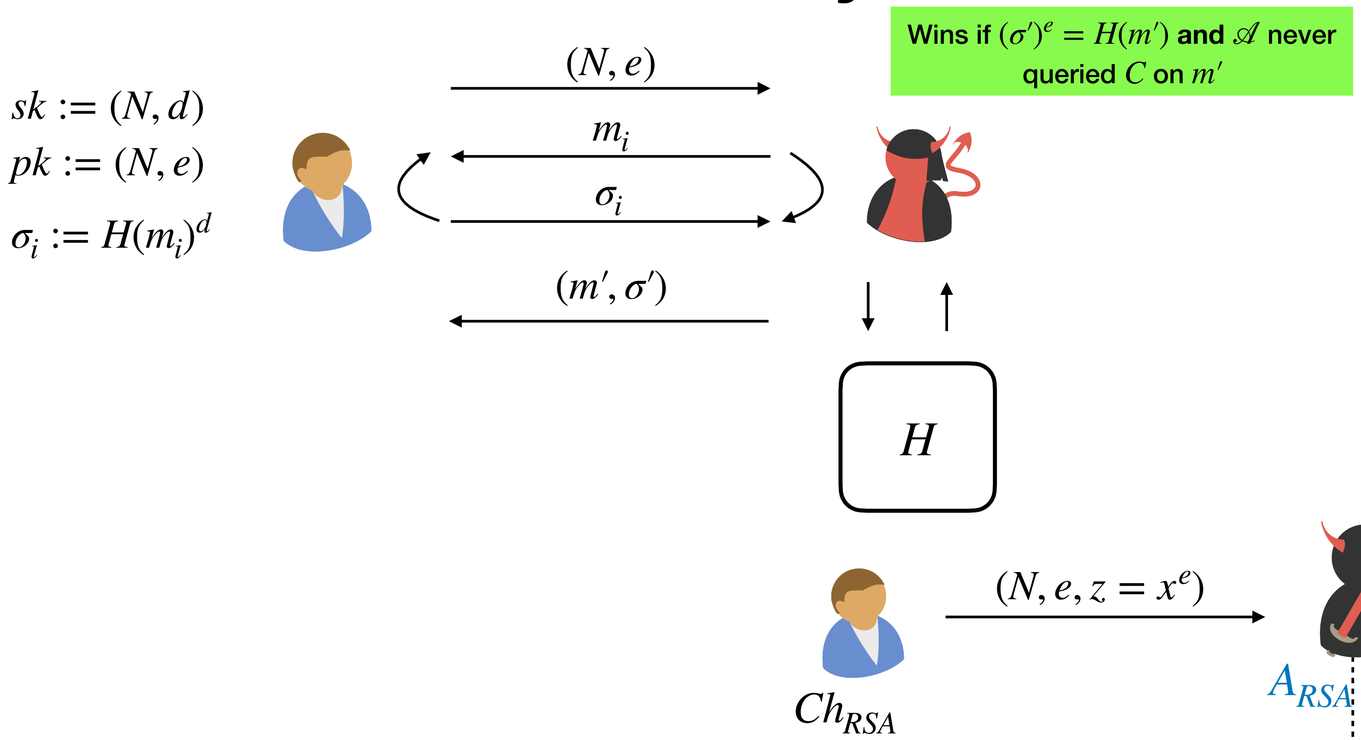


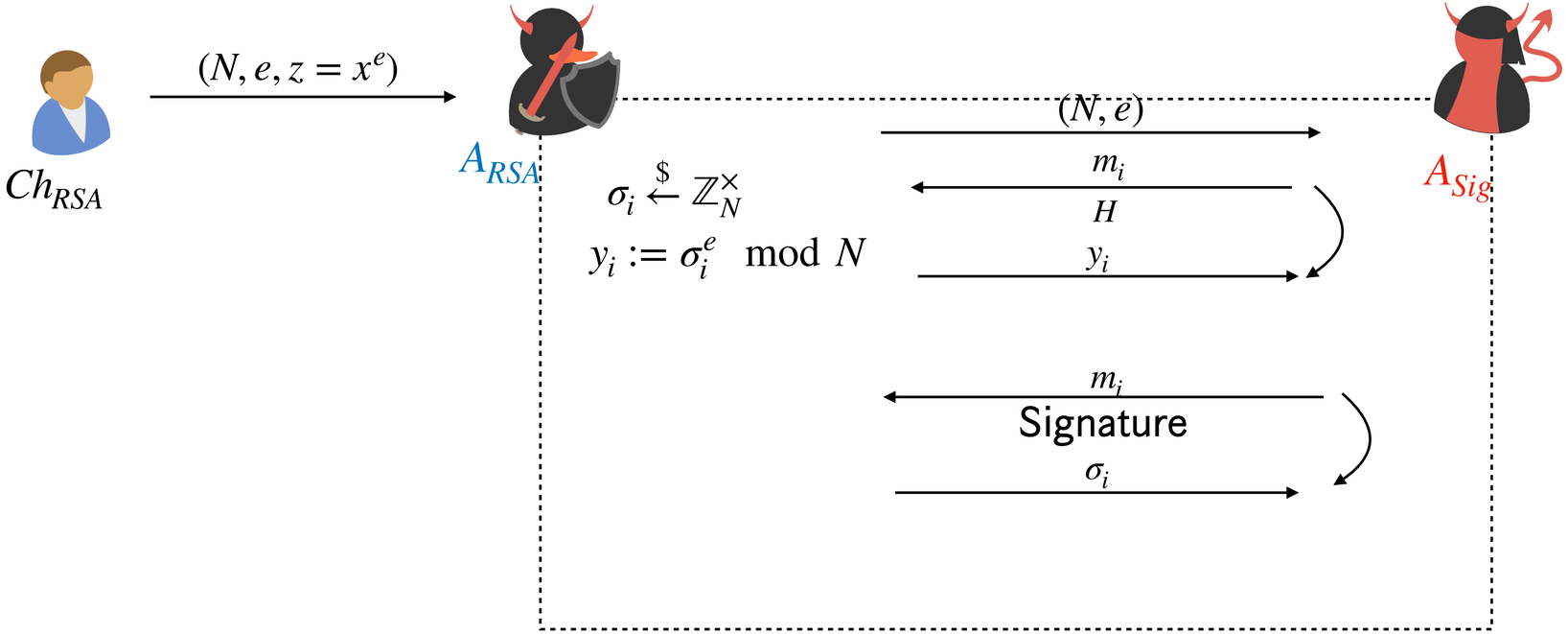


So that we can forge signatures, we'll only pick hash values we know the e-th root of!

Proof of Security

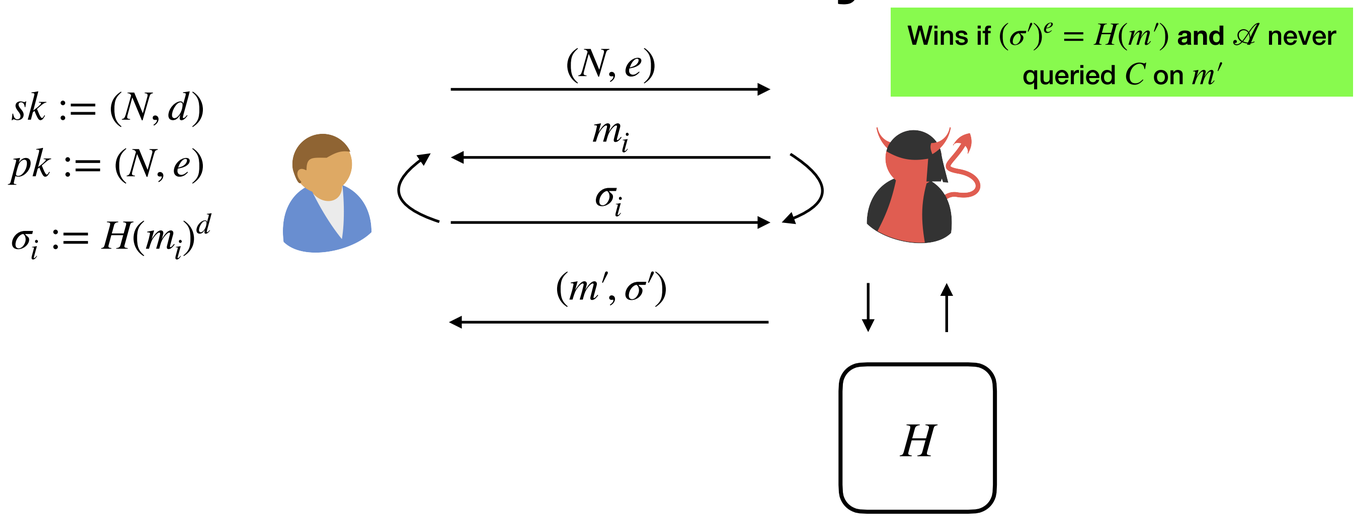
- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$

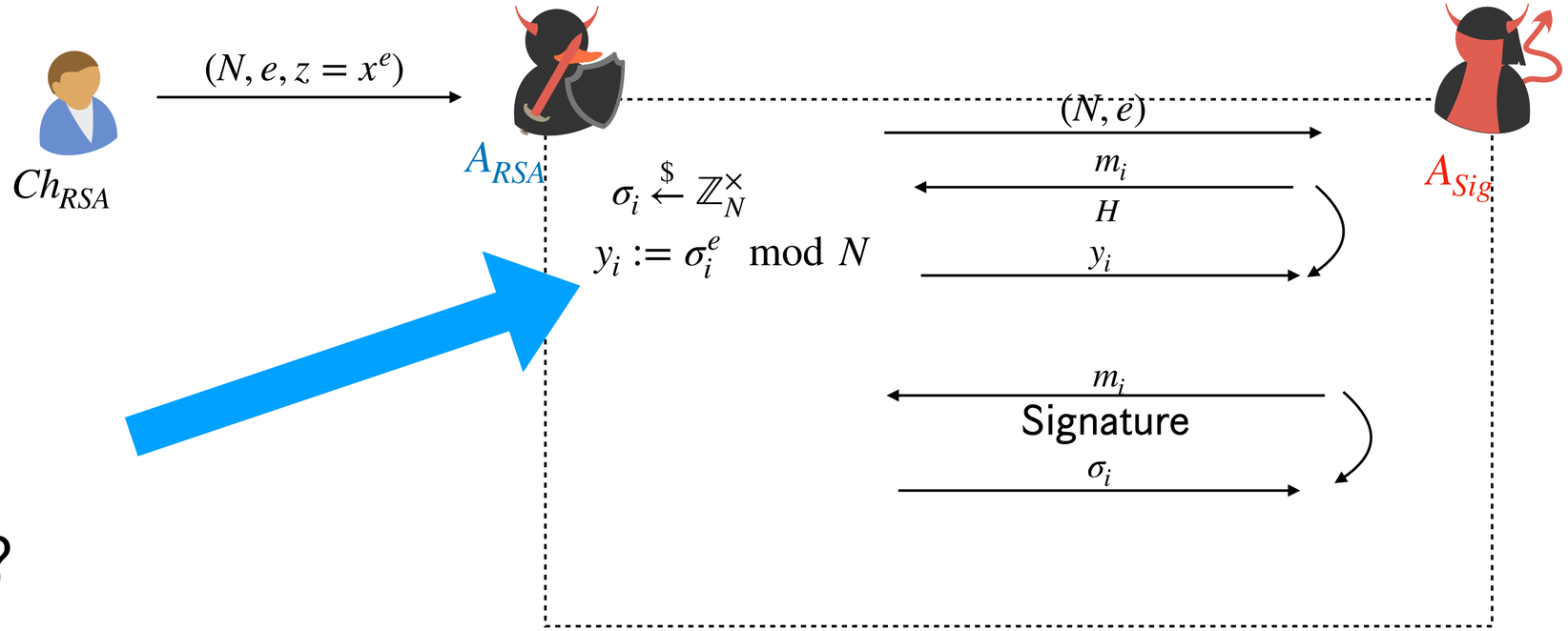




Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$

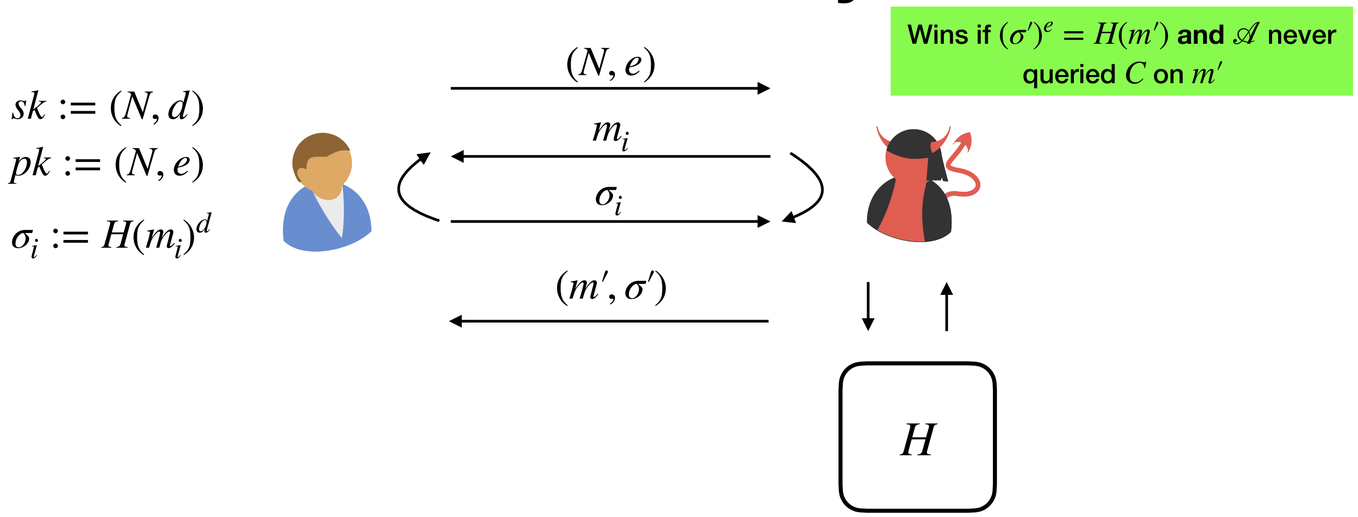


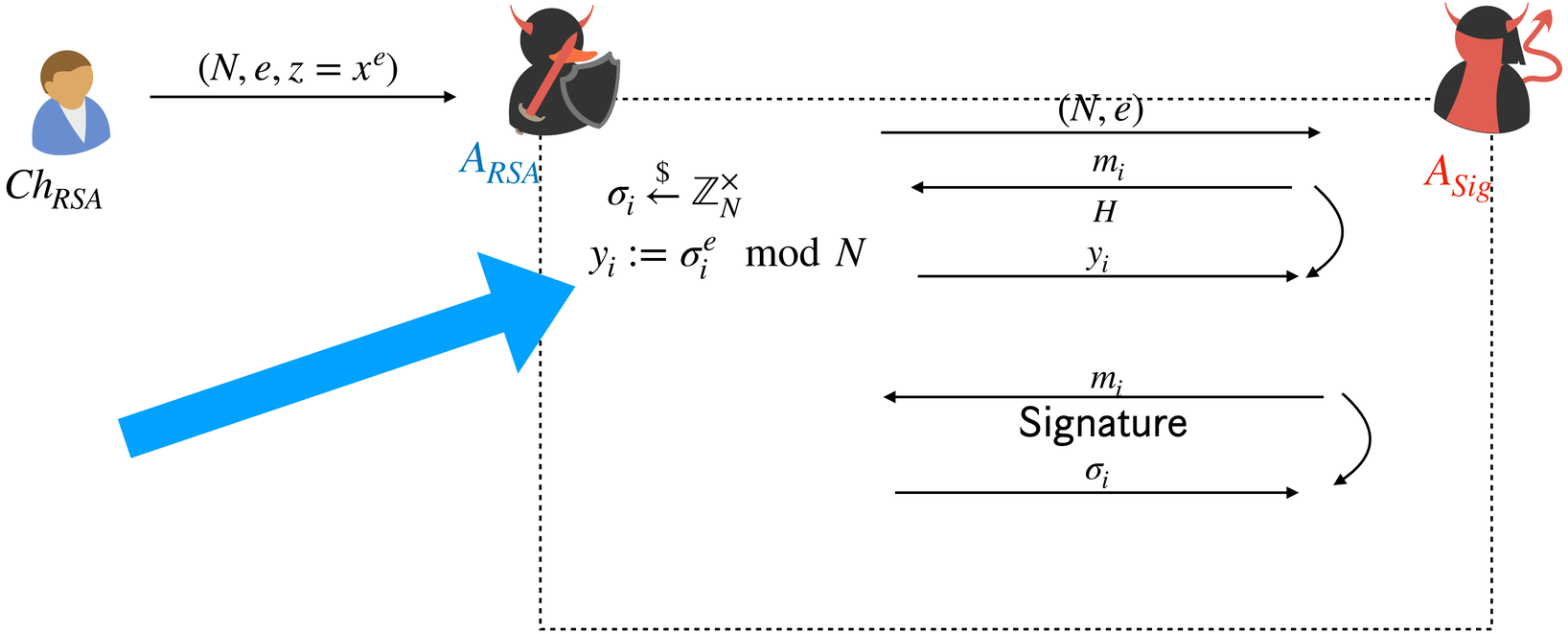


When do we set $y_i = z$? How do we know which m_i A_{Sig} is going to forge on?

Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow$ RSASetup(1^λ)
  - $pk := (N, e)$; $sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$

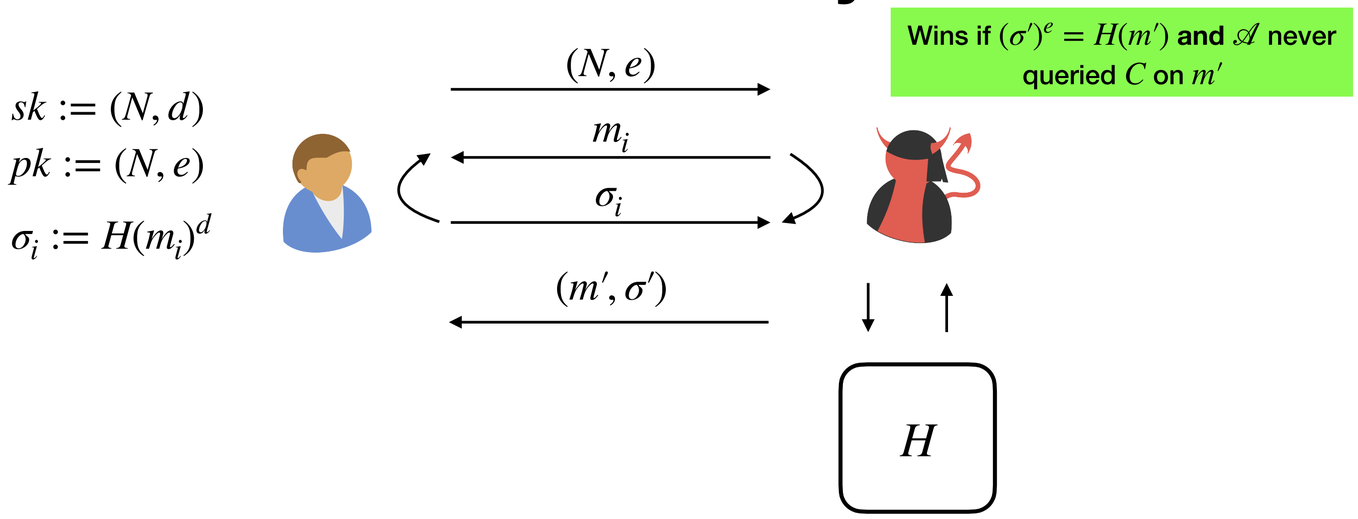


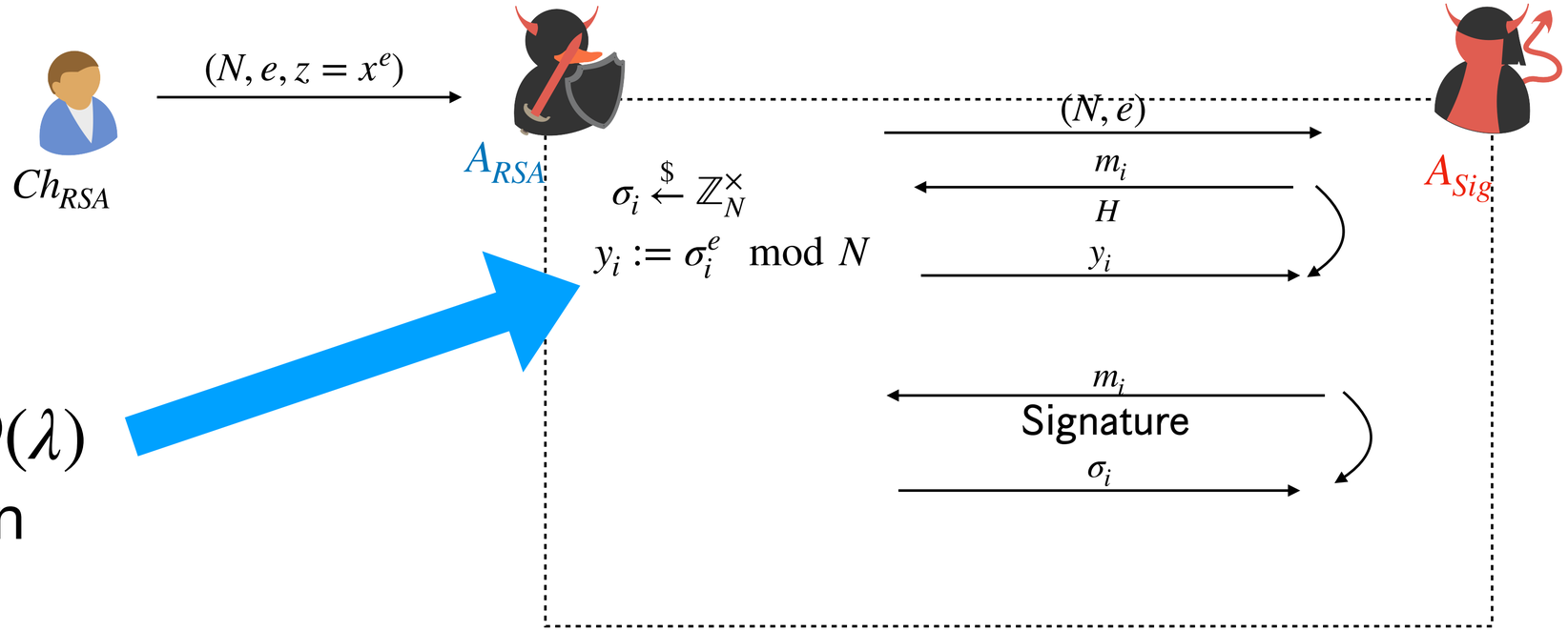


Just guess!

Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$



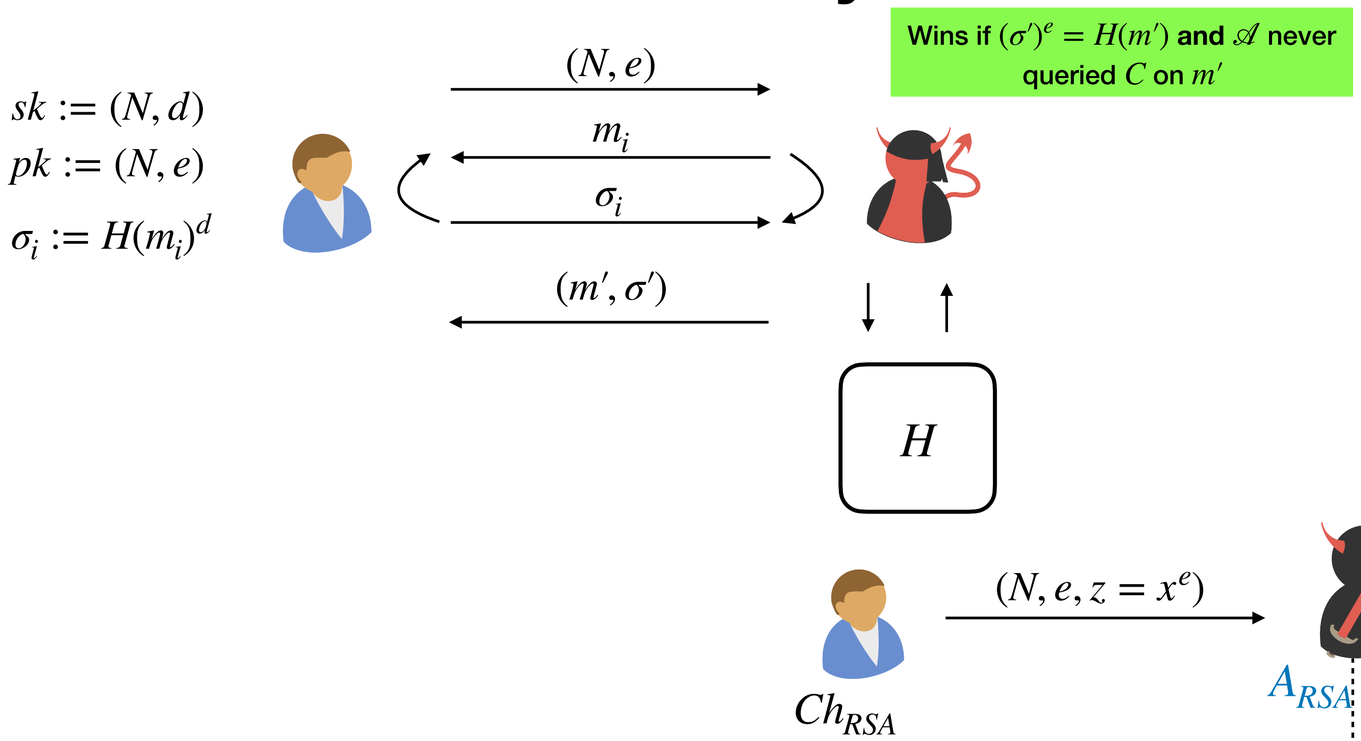


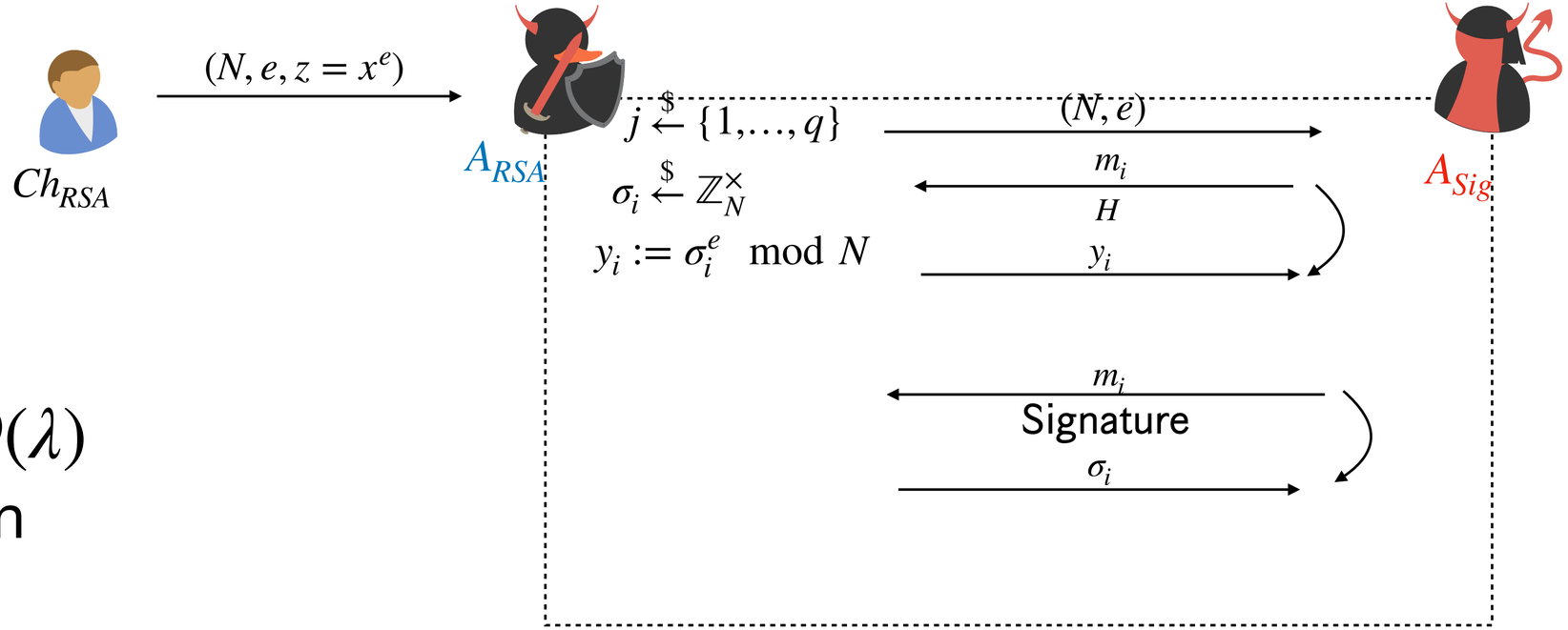
Just guess!

A_{Sig} can only make $O(\lambda)$ queries, so we still win with good odds!

Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign($(N, d), m$): $\sigma := H(m)^d \mod N$
- Ver($(N, e), m, \sigma$): $\sigma^e \stackrel{?}{=} H(m) \mod N$



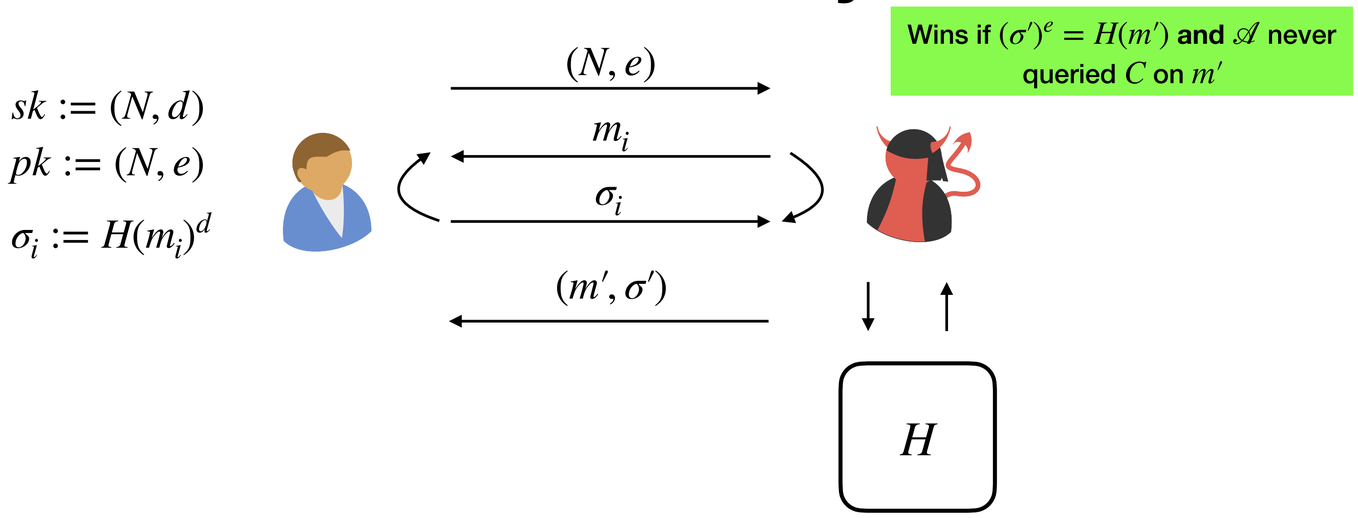


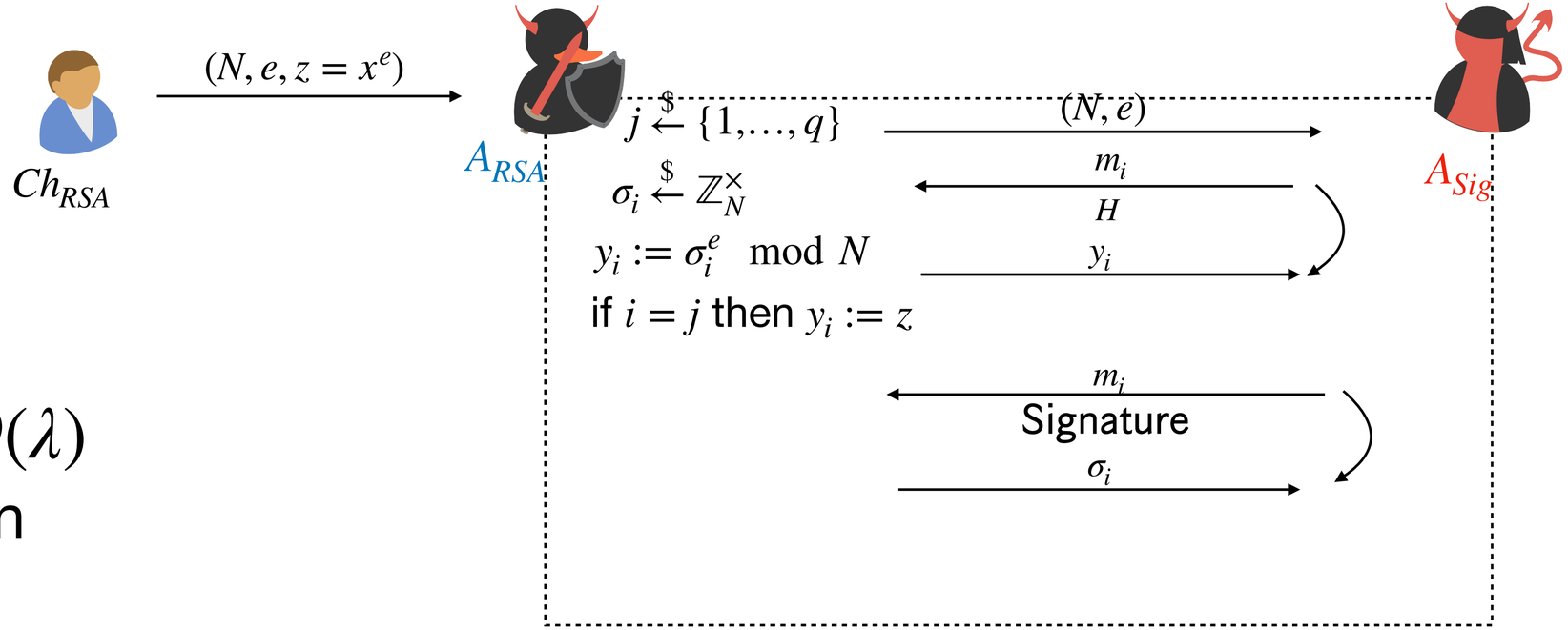
Just guess!

A_{Sig} can only make $O(\lambda)$ queries, so we still win with good odds!

Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$



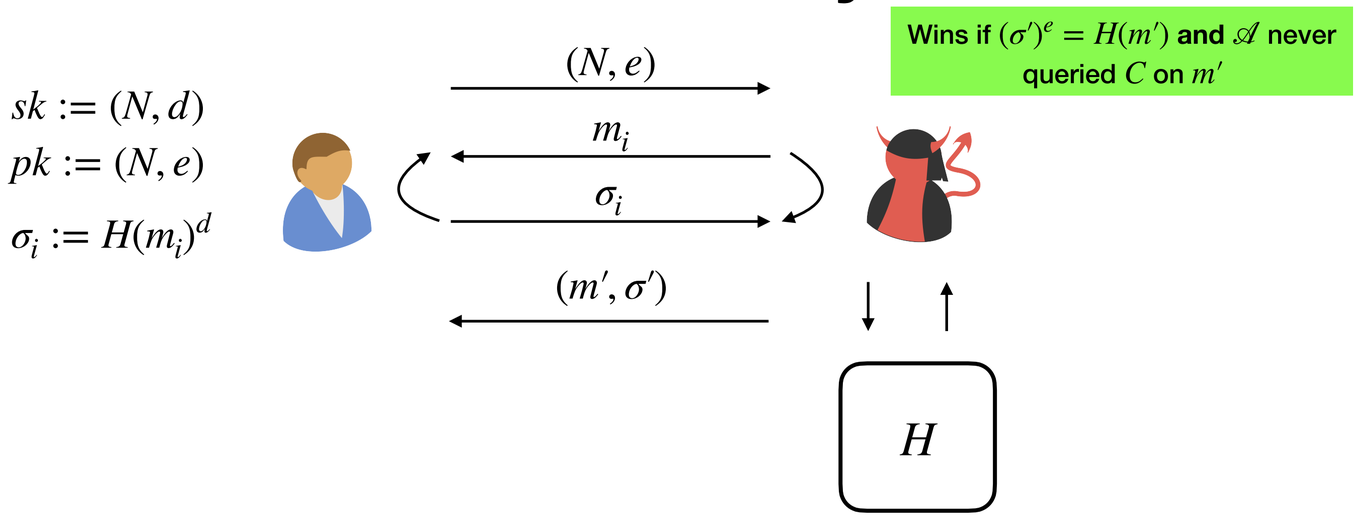


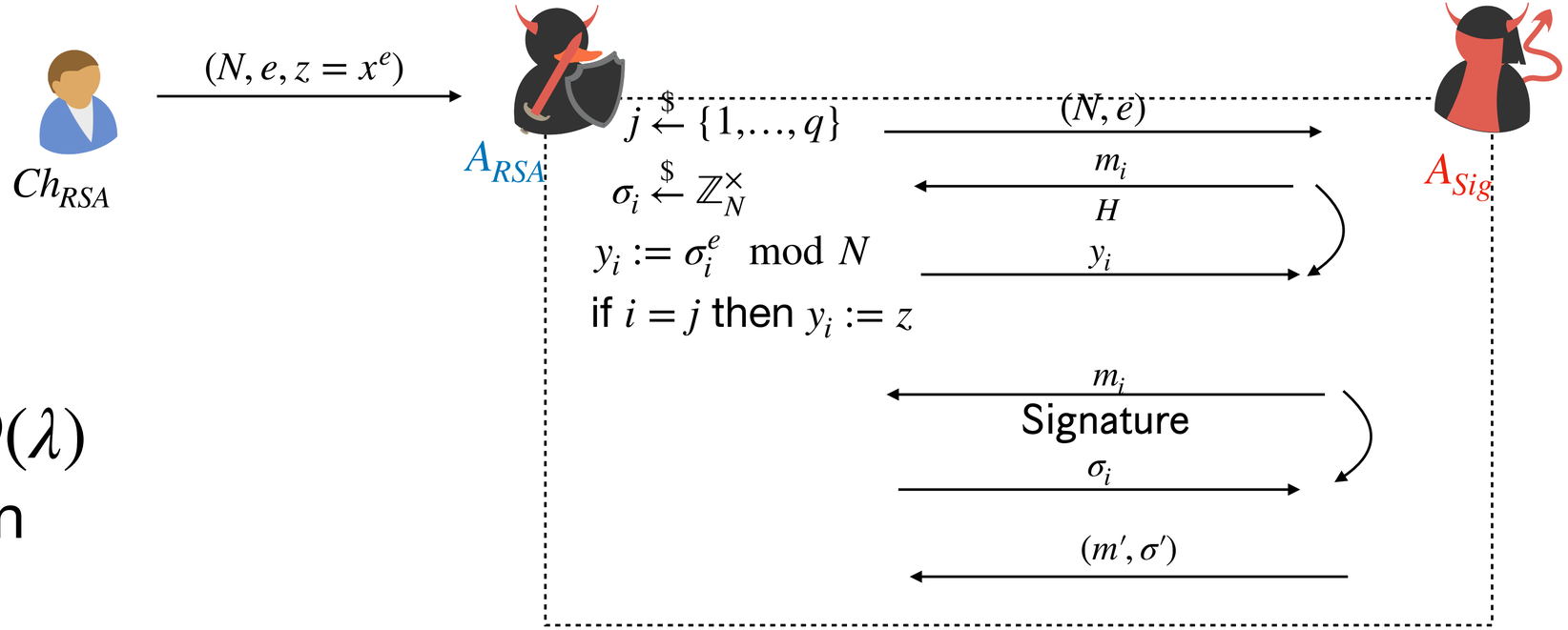
Just guess!

A_{Sig} can only make $O(\lambda)$ queries, so we still win with good odds!

Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$



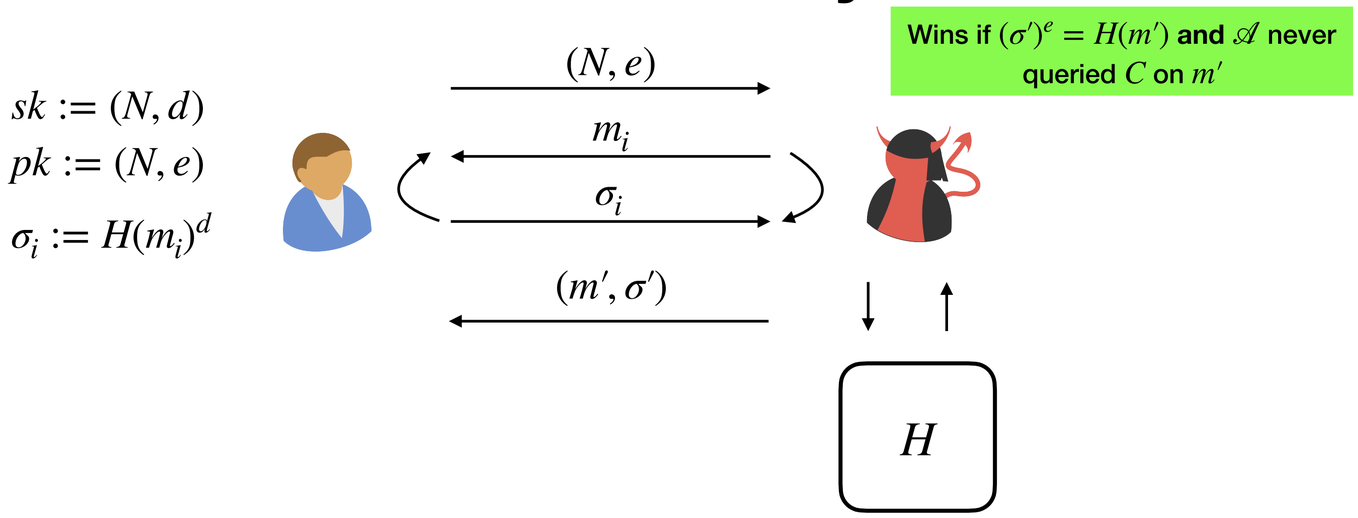


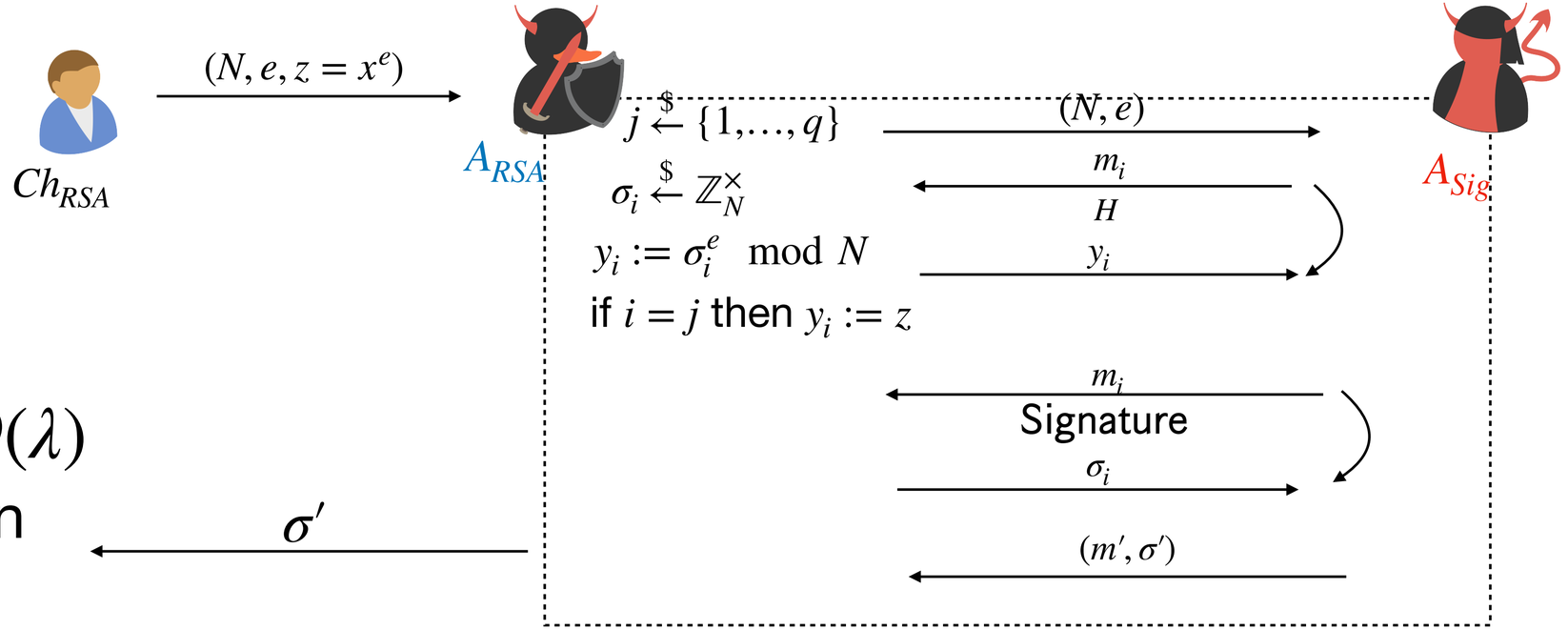
Just guess!

A_{Sig} can only make $O(\lambda)$ queries, so we still win with good odds!

Proof of Security

- KeyGen(1^λ)
  - $N, e, d \leftarrow \text{RSASetup}(1^\lambda)$
  - $pk := (N, e);\ \ sk := (N, d)$
- Sign$((N, d), m)$: $\sigma := H(m)^d \mod N$
- Ver$((N, e), m, \sigma)$: $\sigma^e \stackrel{?}{=} H(m) \mod N$

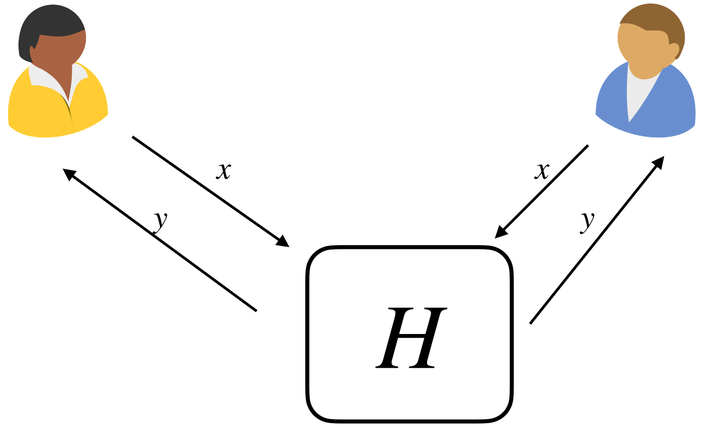




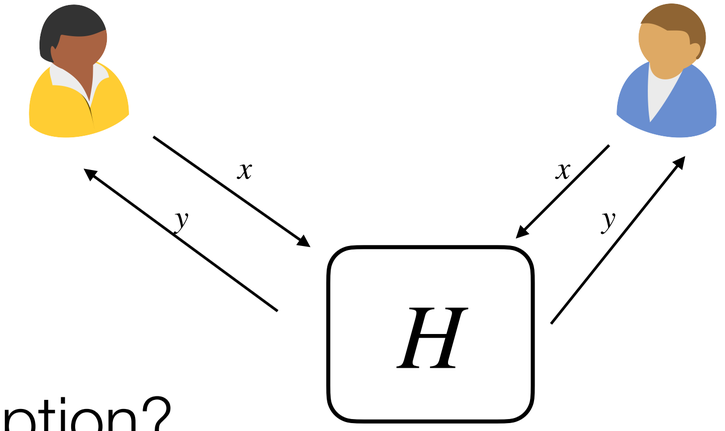
Just guess!

A_{Sig} can only make $O(\lambda)$ queries, so we still win with good odds!

The Random Oracle Model

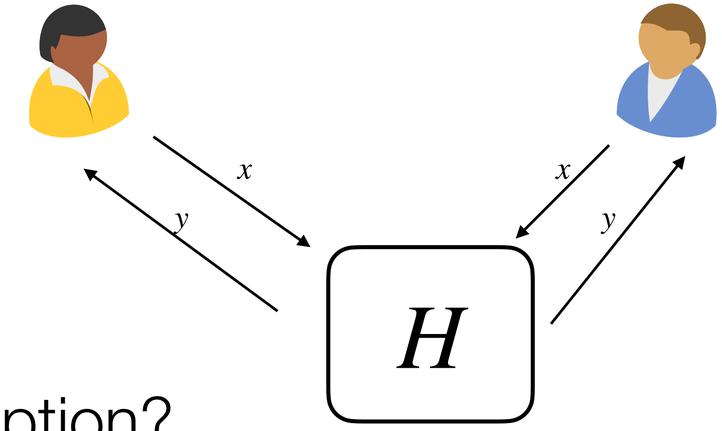


The Random Oracle Model



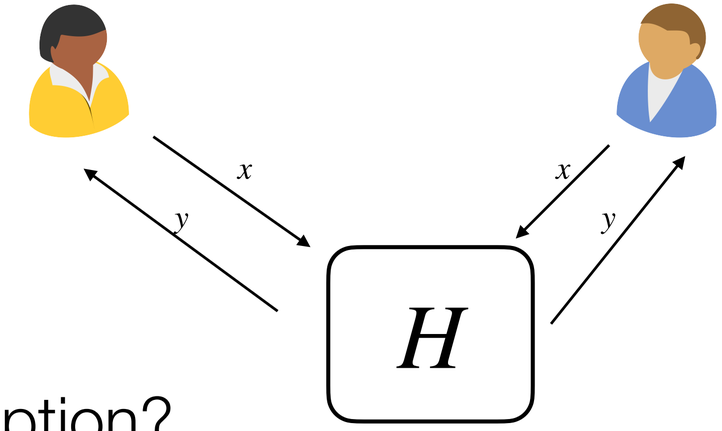
- Is “there is some public random function” a realistic assumption?

The Random Oracle Model



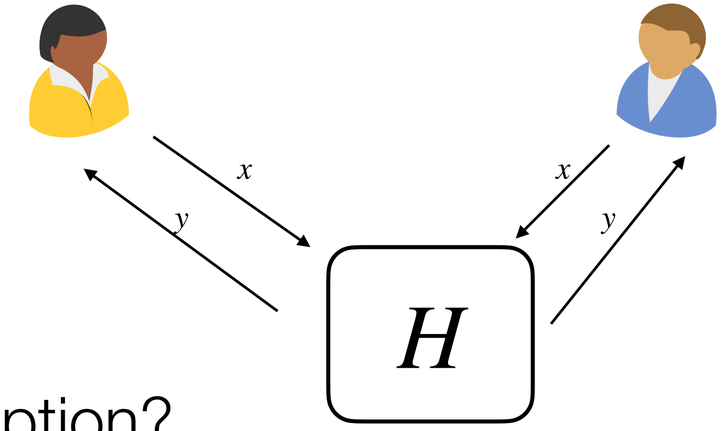
- Is “there is some public random function” a realistic assumption?
  - NO! Nothing like this exists at all in the real world.

The Random Oracle Model



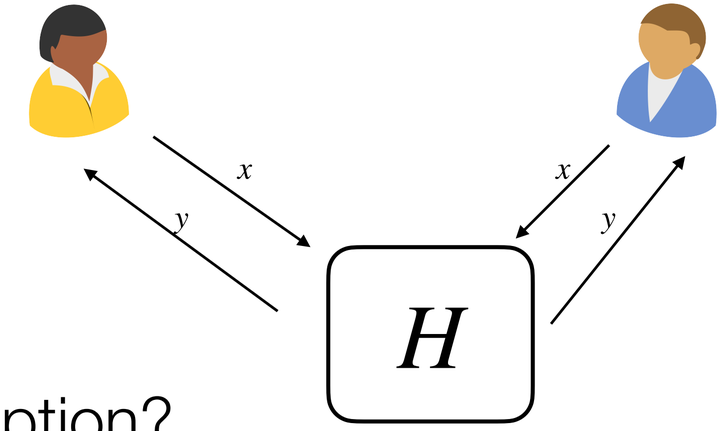
- Is “there is some public random function” a realistic assumption?
  - NO! Nothing like this exists at all in the real world.
- If you design a scheme with random oracles how do you go deploy it? What code do you write at the step “compute $H(m)$”?

The Random Oracle Model



- Is “there is some public random function” a realistic assumption?
  - NO! Nothing like this exists at all in the real world.
- If you design a scheme with random oracles how do you go deploy it? What code do you write at the step “compute $H(m)$”?
  - You use a hash function! (e.g. SHA)

The Random Oracle Model



- Is “there is some public random function” a realistic assumption?
  - NO! Nothing like this exists at all in the real world.
- If you design a scheme with random oracles how do you go deploy it? What code do you write at the step “compute $H(m)$”?
  - You use a hash function! (e.g. SHA)
- We just assume that if you prove something is secure in the random oracle model, then it is *probably* secure if you use a hash function

Minimal Assumptions needed for Signatures

Minimal Assumptions needed for Signatures

- Cryptographers like investigating what the minimal assumptions needed for any primitive are

Minimal Assumptions needed for Signatures

- Cryptographers like investigating what the minimal assumptions needed for any primitive are
- Public key encryption required special (i.e. “non-minicrypt”) assumptions, what about digital signatures?

Minimal Assumptions needed for Signatures

- Cryptographers like investigating what the minimal assumptions needed for any primitive are
- Public key encryption required special (i.e. “non-minicrypt”) assumptions, what about digital signatures?
- We can build signatures from OWFs!

One-Time Signatures

$$\Pr\left[\begin{array}{c}\mathsf{Ver}(pk, m', \sigma') = 1 \\ \textbf{and } \mathscr{A} \text{ never queried } m'\end{array} : \begin{array}{c}(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (m', \sigma') \leftarrow \mathscr{A}^{\mathsf{OneSign}(k,\cdot)}(1^\lambda, pk)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

$\Pr[\mathscr{A} \text{ wins DSGame}] \leq \mathsf{negl}(\lambda)$

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$\sigma_i \leftarrow \mathsf{Sign}(sk, m)$

pk →

← m

σ →

← (m', σ')

Wins if $\mathsf{Ver}(pk, m', \sigma') = 1$ **and** $\mathscr{A}$ never queried m'

One-Time Signature Construction

Lamport Signatures

Let f be a one-way function

One-Time Signature Construction

Lamport Signatures

Let f be a one-way function

- KeyGen(1^λ)

One-Time Signature Construction

Lamport Signatures

Let f be a one-way function

- KeyGen(1^λ)
  - $sk := \begin{pmatrix} x_1^0 & x_2^0 & \dots & x_n^0 \\ x_1^1 & x_2^1 & \dots & x_n^1 \end{pmatrix}$ where $x_i^b \xleftarrow{\$} \{0,1\}$ for all $i \in \{1,\dots,\lambda\}$ and $b \in \{0,1\}$

One-Time Signature Construction

Lamport Signatures

Let f be a one-way function

- KeyGen(1^λ)
  - $sk := \begin{pmatrix} x_1^0 & x_2^0 & \dots & x_n^0 \\ x_1^1 & x_2^1 & \dots & x_n^1 \end{pmatrix}$ where $x_i^b \xleftarrow{\$} \{0,1\}$ for all $i \in \{1,\dots,\lambda\}$ and $b \in \{0,1\}$
  - $sk := \begin{pmatrix} y_1^0 & y_2^0 & \dots & y_n^0 \\ y_1^1 & y_2^1 & \dots & y_n^1 \end{pmatrix}$ where $y_i^b := f(x_i^b)$ for all $i \in \{1,\dots,\lambda\}$ and $b \in \{0,1\}$

One-Time Signature Construction

Lamport Signatures

Let f be a one-way function

- KeyGen(1^λ)
  - $sk := \begin{pmatrix} x_1^0 & x_2^0 & \dots & x_n^0 \\ x_1^1 & x_2^1 & \dots & x_n^1 \end{pmatrix}$ where $x_i^b \xleftarrow{\$} \{0,1\}$ for all $i \in \{1,\dots,\lambda\}$ and $b \in \{0,1\}$
  - $sk := \begin{pmatrix} y_1^0 & y_2^0 & \dots & y_n^0 \\ y_1^1 & y_2^1 & \dots & y_n^1 \end{pmatrix}$ where $y_i^b := f(x_i^b)$ for all $i \in \{1,\dots,\lambda\}$ and $b \in \{0,1\}$
- Sign(sk, m): $\sigma := \left(x_1^{m[1]}, x_2^{m[2]}, \dots, x_n^{m[n]}\right)$

One-Time Signature Construction

Lamport Signatures

Let f be a one-way function

- KeyGen(1^λ)
  - $sk := \begin{pmatrix} x_1^0 & x_2^0 & \dots & x_n^0 \\ x_1^1 & x_2^1 & \dots & x_n^1 \end{pmatrix}$ where $x_i^b \xleftarrow{\$} \{0,1\}$ for all $i \in \{1,\dots,\lambda\}$ and $b \in \{0,1\}$
  - $sk := \begin{pmatrix} y_1^0 & y_2^0 & \dots & y_n^0 \\ y_1^1 & y_2^1 & \dots & y_n^1 \end{pmatrix}$ where $y_i^b := f(x_i^b)$ for all $i \in \{1,\dots,\lambda\}$ and $b \in \{0,1\}$
- Sign(sk, m): $\sigma := \left(x_1^{m[1]}, x_2^{m[2]}, \dots, x_n^{m[n]}\right)$
- Ver(pk, m, σ): $\bigwedge_{i \in \{1,\dots,n\}} f(\sigma_i) \stackrel{?}{=} y_i^{m[i]}$

One-Time Signature Construction

Lamport Signatures

Let f be a one-way function

- KeyGen(1^λ)
  - $sk := \begin{pmatrix} x_1^0 & x_2^0 & \dots & x_n^0 \\ x_1^1 & x_2^1 & \dots & x_n^1 \end{pmatrix}$ where $x_i^b \xleftarrow{\$} \{0,1\}$ for all $i \in \{1,\dots,\lambda\}$ and $b \in \{0,1\}$
  - $sk := \begin{pmatrix} y_1^0 & y_2^0 & \dots & y_n^0 \\ y_1^1 & y_2^1 & \dots & y_n^1 \end{pmatrix}$ where $y_i^b := f(x_i^b)$ for all $i \in \{1,\dots,\lambda\}$ and $b \in \{0,1\}$
- Sign(sk, m): $\sigma := \left(x_1^{m[1]}, x_2^{m[2]}, \dots, x_n^{m[n]}\right)$
- Ver(pk, m, σ): $\bigwedge_{i \in \{1,\dots,n\}} f(\sigma_i) \stackrel{?}{=} y_i^{m[i]}$

Example:

$m = 0110$

$\sigma = (x_1^0, x_2^1, x_3^1, x_4^0)$

One-Time to Multi-Message Signatures

One-Time to Multi-Message Signatures

- Say I have a one-time secure signature scheme (**KeyGen, Sign, Ver**)

One-Time to Multi-Message Signatures

- Say I have a one-time secure signature scheme (**KeyGen, Sign, Ver**)
- I could run **KeyGen** a bunch of times to make a bunch of public keys $\{pk_1, pk_2, \ldots, pk_N\}$

One-Time to Multi-Message Signatures

- Say I have a one-time secure signature scheme (**KeyGen, Sign, Ver**)
- I could run **KeyGen** a bunch of times to make a bunch of public keys $\{pk_1, pk_2, \ldots, pk_N\}$
- Then I can just sign each message with a new public key! Now I can sign as many messages as I want.

One-Time to Multi-Message Signatures

- Say I have a one-time secure signature scheme (**KeyGen, Sign, Ver**)
- I could run **KeyGen** a bunch of times to make a bunch of public keys $\{pk_1, pk_2, \ldots, pk_N\}$
- Then I can just sign each message with a new public key! Now I can sign as many messages as I want.
- To verify, you just need to check that the public key I signed under really was in my big list of public keys $pk_i \in \{pk1, pk2, \ldots, pk_N\}$.

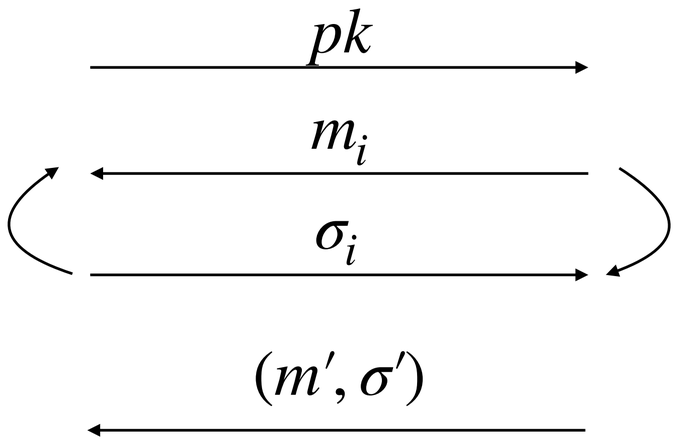
One-Time to Multi-Message Signatures

- Say I have a one-time secure signature scheme (**KeyGen, Sign, Ver**)
- I could run **KeyGen** a bunch of times to make a bunch of public keys $\{pk_1, pk_2, \ldots, pk_N\}$
- Then I can just sign each message with a new public key! Now I can sign as many messages as I want.
- To verify, you just need to check that the public key I signed under really was in my big list of public keys $pk_i \in \{pk1, pk2, \ldots, pk_N\}$.
- We'll see later how to compress this "super" public key so that it isn't burdensome to store

Strong UF-CMA

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$

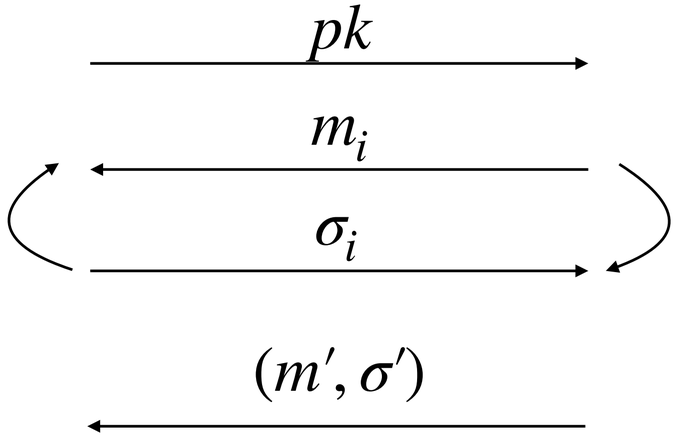


Wins if $\mathsf{Ver}(pk, m', \sigma') = 1$ **and** $\mathcal{A}$ **never queried** m'

Strong UF-CMA

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$



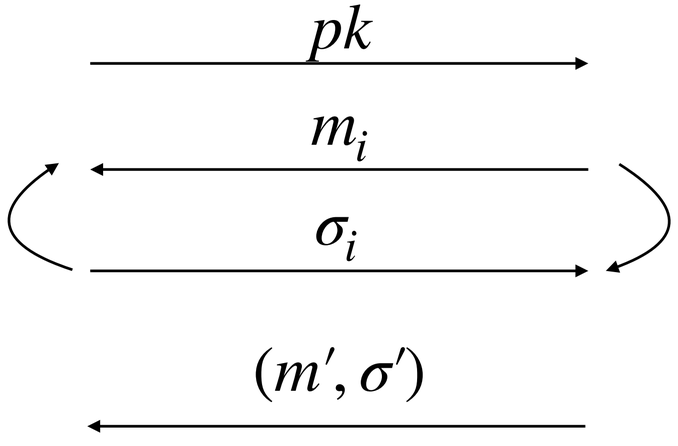
Wins if $\mathsf{Ver}(pk, m', \sigma') = 1$ **and** $\mathcal{A}$ **never queried** m'

UF-CMA says that an adversary can't forge a signature on a message hasn't seen a signature for

Strong UF-CMA

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$



Wins if $\mathsf{Ver}(pk, m', \sigma') = 1$ **and** $\mathscr{A}$ **never queried** m'

UF-CMA says that an adversary can't forge a signature on a message hasn't seen a signature for

We can make this even stronger by saying that an adversary can't even make *new* signatures for messages its seen signed!

Strong UF-CMA

UF-CMA Security

A Digital Signature scheme (KeyGen, Tag, Ver) satisfies *unforgeability under chosen message attack* (UF-CMA) if for all NUPPT $\mathscr{A}$, there exists a negligible function negl($\cdot$). such that $\forall \lambda \in \mathbb{N}$:

$$\Pr[\mathscr{A} \text{ wins DSGame}] \leq \text{negl}(\lambda)$$

$(sk, pk) \leftarrow \text{KeyGen}(1^\lambda)$

$\sigma_i \leftarrow \text{Sign}(sk, m_i)$

pk

m_i

σ_i

(m', σ')

Wins if $\text{Ver}(pk, m', \sigma') = 1$ **and** $\mathscr{A}$ **never queried** m'

Strong UF-CMA

Strong UF-CMA Security

A Digital Signature scheme (KeyGen, Tag, Ver) satisfies *unforgeability under chosen message attack* (UF-CMA) if for all NUPPT $\mathscr{A}$, there exists a negligible function negl($\cdot$). such that $\forall \lambda \in \mathbb{N}$:

$$\Pr[\mathscr{A} \text{ wins SDSGame}] \leq \text{negl}(\lambda)$$

$(sk, pk) \leftarrow \text{KeyGen}(1^\lambda)$

$\sigma_i \leftarrow \text{Sign}(sk, m_i)$

pk

m_i

σ_i

(m', σ')

Wins if $\text{Ver}(pk, m', \sigma') = 1$ **and** $\mathscr{A}$ **never got** σ' **as a response after querying** m'