Digital Signature Schemes and the Random Oracle Model

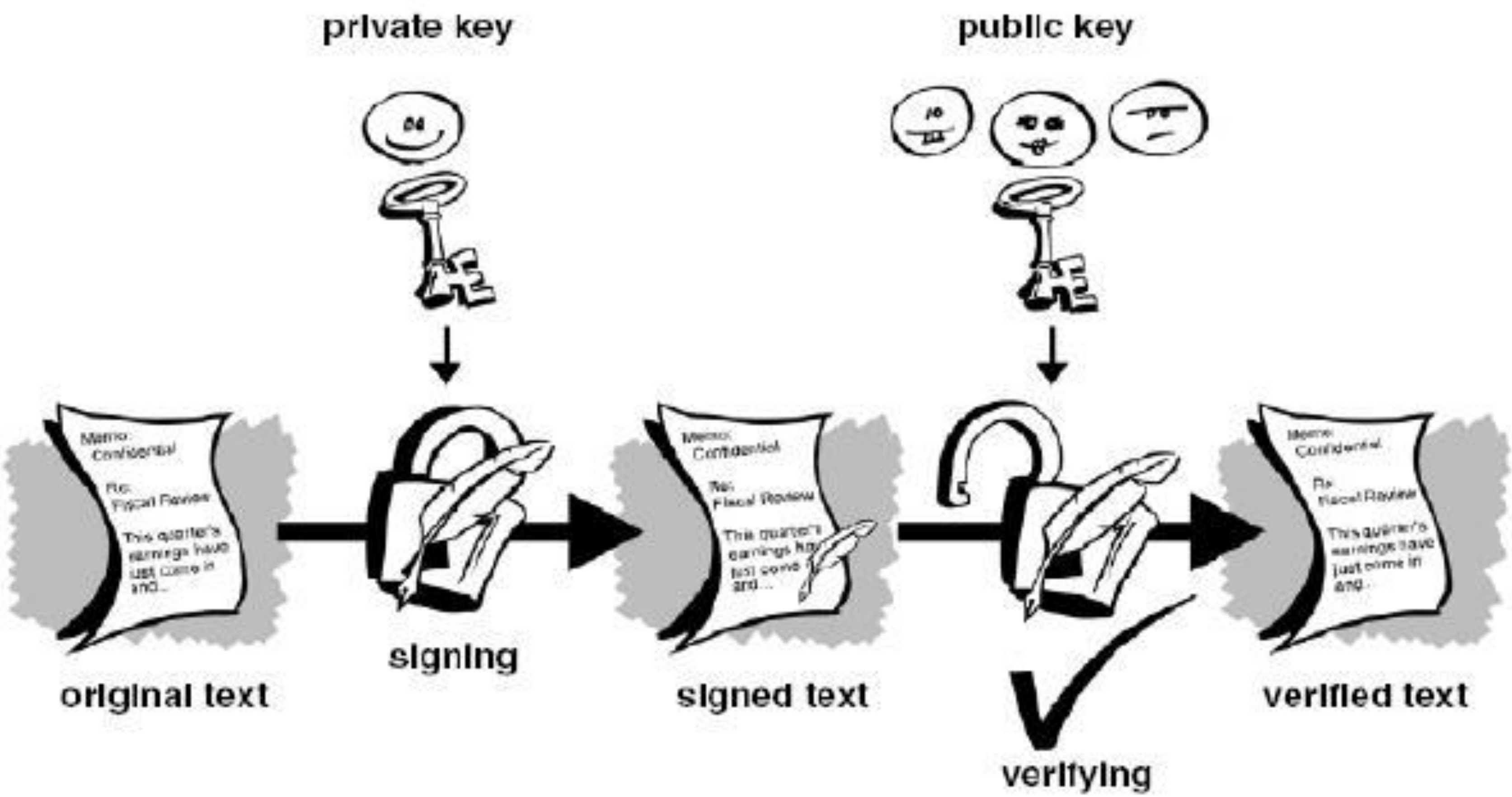
A. Hülsing

Technische Universiteit Eindhoven University of Technology

Today‘s goal

Review provable security of “in use” signature schemes. (PKCS #1 v2.x)

Digital Signature



Source: http://hari-cio-8a.blog.ugm.ac.id/files/2013/03/DSA.jpg

Definition: Digital Signature (formally)

Let $\mathcal{M}$ be the message space. A digital signature scheme **DSig = (KeyGen, Sign, Verify)** is a triple of PPT algorithms

- $\mathbf{KeyGen}(1^k)$**:** upon input of a security parameter $\mathbf{1}^k$ outputs a private signing key $\boldsymbol{sk}$ and a public verification key $\boldsymbol{pk}$,
- $\mathbf{Sign}(\boldsymbol{sk}, \boldsymbol{M})$**:** outputs a signature $\boldsymbol{\sigma}$ under $\boldsymbol{sk}$ for message $\boldsymbol{M}$, if $\boldsymbol{M} \in \mathcal{M}$,
- $\mathbf{Verify}(\boldsymbol{pk}, \boldsymbol{M}, \boldsymbol{\sigma})$**:** outputs 1 iff $\boldsymbol{\sigma}$ is a valid signature on $\boldsymbol{M}$ under $\boldsymbol{pk}$,

such that the following correctness condition is fulfilled:

$$\boldsymbol{Pr}[(\boldsymbol{pk}, \boldsymbol{sk}) \leftarrow \mathbf{KeyGen}(\mathbf{1}^k), \forall(\boldsymbol{M} \in \mathcal{M}): \mathbf{Verify}(\boldsymbol{pk}, \boldsymbol{M}, \mathbf{Sign}(\boldsymbol{sk}, \boldsymbol{M})) = \mathbf{1}] = \mathbf{1}.$$

Attack goals

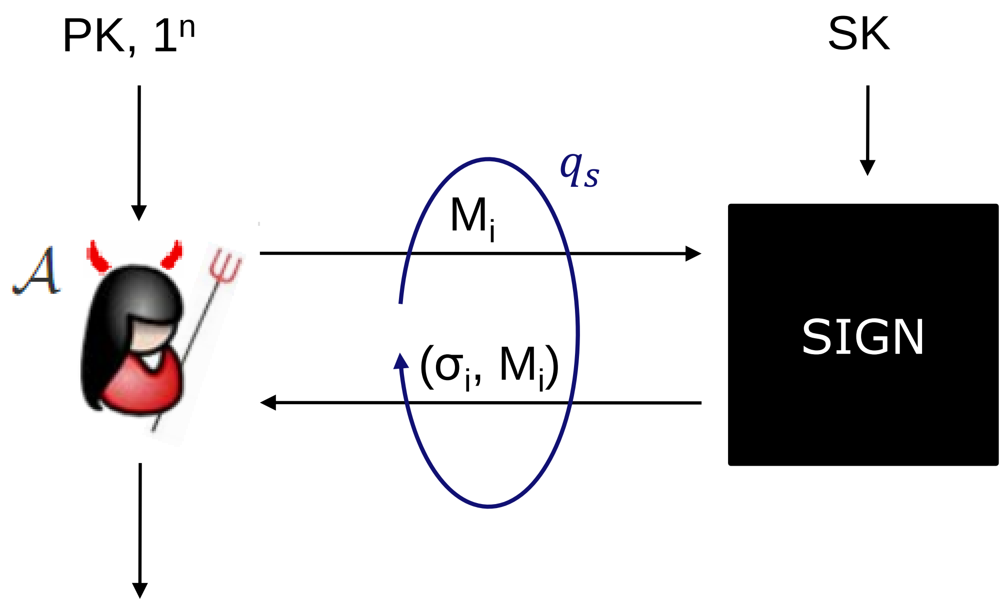
Consider adversary A

- **Full break (FB):** A can compute the secret key.
- **Universal forgery (UU):** A can forge a signature for any given message. A can efficiently answer any signing query.
- **Selective forgery (SU):** A can forge a signature for some message of its choice. In this case A commits itself to a message before the attack starts.
- **Existential forgery (EU):** A can forge a signature for one, arbitrary message. A might output a forgery for any message for which it did not learn the signature from an oracle during the attack.

Attack Models

- **Key-only attack (KOA):** A only gets the public key for which it has to forge a signature.
- **Random message attack (RMA):** A learns the public key and the signatures on a set of random messages.
- **Adaptively chosen message attack (CMA):** A learns the public key and is allowed to adaptively ask for the signatures on messages of its choice.

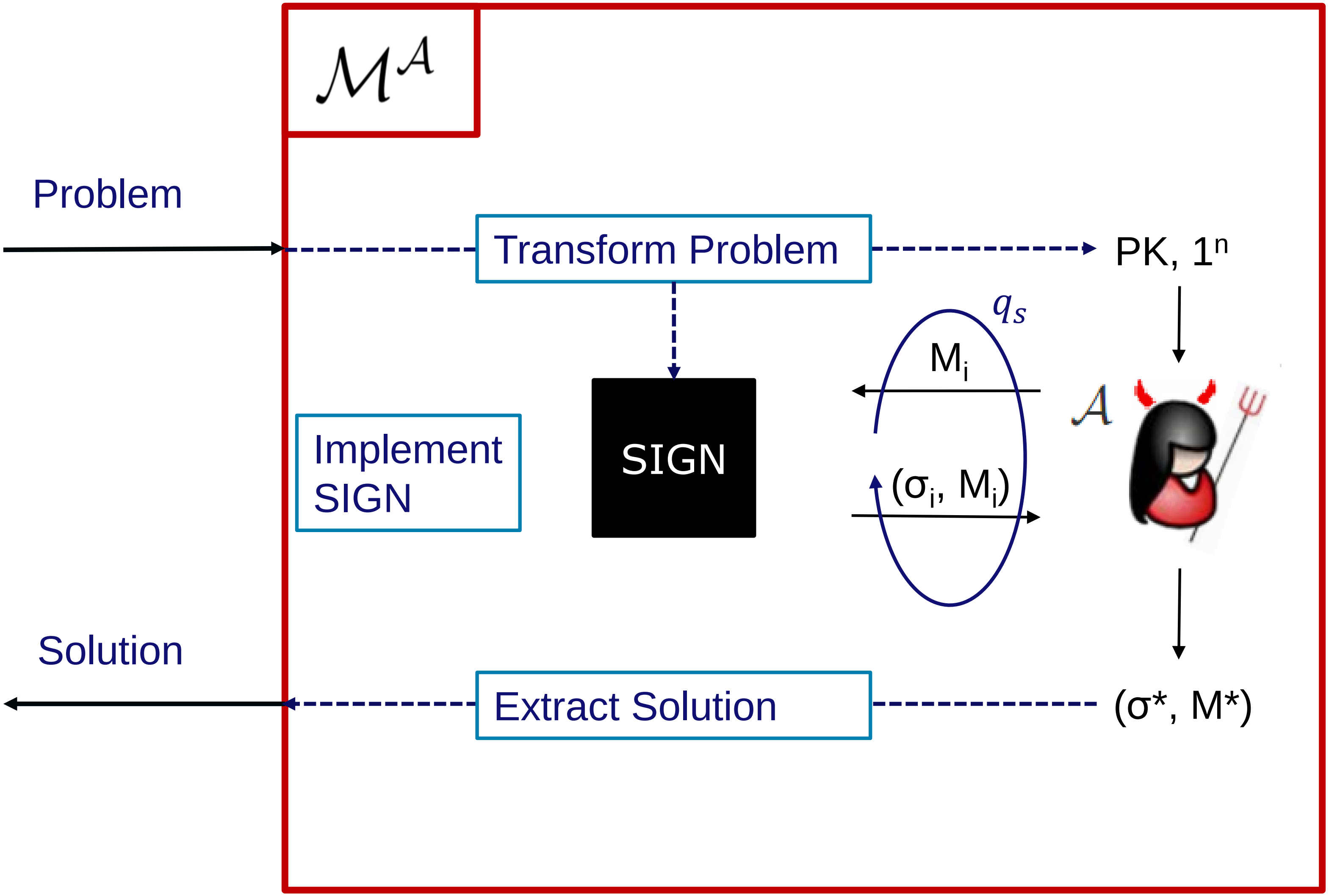
Existential unforgeability under adaptive chosen message attacks



(σ^*, M^*)

Success if $M^* \neq M_i$, $\forall$ i $\in [0, q]$ and Verify(pk,σ^*,M^*) = Accept

Reduction



$\mathcal{M}^{\mathcal{A}}$
Problem
Transform Problem
PK, 1^n
q_s
M_i
$\mathcal{A}$
Implement SIGN
SIGN
(σ_i, M_i)
Solution
Extract Solution
(σ^*, M^*)

Why security reductions?

- **Current RSA signature standards so far unbroken**
- **Vulnerabilities might exist! (And existed for previous proposals)**
- **Might be possible to forge RSA signatures without solving RSA problem or factoring!**

What could possibly go wrong?

RSA

Let $(\mathbf{N}, \mathbf{e}, \mathbf{d}) \longleftarrow \mathbf{GenRSA}(\mathbf{1}^{\boldsymbol{k}})$ be a PPT algorithm that outputs a modulus $\boldsymbol{N}$ that is the product of two $\boldsymbol{k}$-bit primes (except possibly with negligible probability), along with an integer $\boldsymbol{e} > \mathbf{0}$ with $\mathbf{gcd}(\boldsymbol{e}, \boldsymbol{\phi}(\boldsymbol{N})) = \mathbf{1}$ and an integer $\boldsymbol{d} > \mathbf{0}$ satisfying $\boldsymbol{ed} = \mathbf{1} \bmod \boldsymbol{\phi}(\boldsymbol{N})$.

For any $(\mathbf{N}, \mathbf{e}, \mathbf{d}) \longleftarrow \mathbf{GenRSA}(\mathbf{1}^{\boldsymbol{k}})$ and any $\boldsymbol{y} \in \mathbb{Z}_{\boldsymbol{N}}^*$ we have

$$(\boldsymbol{y}^{\boldsymbol{d}})^{\boldsymbol{e}} = \boldsymbol{y}^{\boldsymbol{de}} = \boldsymbol{y}^{\boldsymbol{de} \bmod \boldsymbol{\phi}(\boldsymbol{N})} = \boldsymbol{y}^{\mathbf{1}} = \boldsymbol{y} \bmod \boldsymbol{N}$$

RSA Assumption

Definition 1. We say that the RSA problem is hard relative to $\mathbf{GenRSA}$ if for all PPT algorithms **A**, the following is negligible:

$$\boldsymbol{Pr}[(\boldsymbol{N},\boldsymbol{e},\boldsymbol{d}) \leftarrow \mathbf{GenRSA}(\mathbf{1}^{\boldsymbol{k}});\ \boldsymbol{y} \leftarrow \mathbb{Z}_{\boldsymbol{N}}^{*};$$
$$\boldsymbol{x} \leftarrow \boldsymbol{A}(\boldsymbol{N},\boldsymbol{e},\boldsymbol{y}) : \boldsymbol{x}^{\boldsymbol{e}} = \boldsymbol{y} \bmod \boldsymbol{N}].$$

A simple RSA Signature (a.k.a. textbook RSA)

$\mathbf{KeyGen}(1^k)$: Run $(\boldsymbol{N}, \boldsymbol{e}, \boldsymbol{d}) \leftarrow \mathbf{GenRSA}(1^k)$.

Return $(\boldsymbol{pk}, \boldsymbol{sk})$ with $\boldsymbol{pk} = (\boldsymbol{N}, \boldsymbol{e}), \boldsymbol{sk} = \boldsymbol{d}$.

$\mathbf{Sign}(\boldsymbol{sk}, \boldsymbol{M})$: Return $\boldsymbol{\sigma} = (\boldsymbol{M}^d \ \mathbf{mod}\ \boldsymbol{N})$

$\mathbf{Verify}(\boldsymbol{pk}, \boldsymbol{M}, \boldsymbol{\sigma})$: Return **1** iff $\boldsymbol{\sigma}^e \mathbf{mod}\ \boldsymbol{N} == \boldsymbol{M}$

Some RSA properties

- **Public function:**

$$P(x) = x^e \bmod N$$

- **Secret function:**

$$S(x) = x^d \bmod N$$

- **Reciprocal property (RP):**

$$P \circ S = S \circ P = Id$$

- **Multiplicative property (MP):**

$$\forall x, y \in \mathbb{Z}_N: \; S(xy) = S(x)S(y)$$

Existential forgery under KOA

Given public key $\boldsymbol{pk} = (\boldsymbol{N}, \boldsymbol{e})$

- Choose random $\boldsymbol{\sigma} \in \mathbb{Z}_{\boldsymbol{N}}$.
- Apply public function:

$$\boldsymbol{P(\sigma) = \sigma^{e} \bmod N = M}$$

- Return signature-message pair $(\boldsymbol{\sigma}, \boldsymbol{M})$.

Universal forgery under CMA

Given public key $\boldsymbol{pk} = (\boldsymbol{N}, \boldsymbol{e})$

To create a forgery on a given message $\boldsymbol{M}$:

1. Choose two messages $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_{\boldsymbol{N}}$ such that
$$\boldsymbol{xy} = \boldsymbol{M} \bmod \boldsymbol{N}$$
2. Ask for signatures $\boldsymbol{\sigma_x}$ of $\boldsymbol{x}$ and $\boldsymbol{\sigma_y}$ of $\boldsymbol{y}$
3. Output forgery $(\boldsymbol{\sigma_x \sigma_y}, \boldsymbol{M})$

Universal forgery under CMA: The Blinding Attack

Given public key $\boldsymbol{pk} = (\boldsymbol{N}, \boldsymbol{e})$

To create a forgery on any given message $\boldsymbol{M}$:

1. Sample random $\boldsymbol{r} \in \mathbb{Z}_{\boldsymbol{N}}^{*}$
2. Ask for signature $\boldsymbol{\sigma}$ on $\boldsymbol{r}^{\boldsymbol{e}}\boldsymbol{M} \bmod \boldsymbol{N}$
3. Output forgery $\left(\frac{\boldsymbol{\sigma}}{\boldsymbol{r}} \bmod \boldsymbol{N}, \boldsymbol{M}\right)$

Recall $\boldsymbol{\sigma} = (\boldsymbol{r}^{\boldsymbol{e}}\boldsymbol{M})^{\boldsymbol{d}} = \boldsymbol{r}^{\boldsymbol{ed}}\boldsymbol{M}^{\boldsymbol{d}} = \boldsymbol{r}\boldsymbol{M}^{\boldsymbol{d}} \bmod \boldsymbol{N}$

Hence $\frac{\boldsymbol{\sigma}}{\boldsymbol{r}} = \boldsymbol{M}^{\boldsymbol{d}} \bmod \boldsymbol{N}$

A slightly better RSA Signature

Assume Hashfunction $\boldsymbol{H}: \{\mathbf{0},\mathbf{1}\}^* \to \{\mathbf{0},\mathbf{1}\}^{\boldsymbol{n}}$ for $\textbf{e.g.}\ \boldsymbol{n} = \mathbf{160}$ (like with SHA1)

$\textbf{KeyGen}(\mathbf{1}^{\boldsymbol{k}})$: Run $(\boldsymbol{N},\boldsymbol{e},\boldsymbol{d}) \leftarrow \textbf{GenRSA}(\mathbf{1}^{\boldsymbol{k}})$.

Return $(\boldsymbol{pk},\boldsymbol{sk})$ with $\boldsymbol{pk} = (\boldsymbol{N},\boldsymbol{e}), \boldsymbol{sk} = \boldsymbol{d}$.

$\textbf{Sign}(\boldsymbol{sk},\boldsymbol{M})$: Pad with suff. zeros that

$$\boldsymbol{\mu}(\boldsymbol{M}) = (\mathbf{0}\ldots\mathbf{0}||\boldsymbol{H}(\boldsymbol{M})) \in \mathbb{Z}_{\boldsymbol{N}}^*$$

Return $\boldsymbol{\sigma} = \boldsymbol{\mu}(\boldsymbol{M})^{d} \textbf{ mod } \boldsymbol{N}$

$\textbf{Verify}(\boldsymbol{pk},\boldsymbol{M},\boldsymbol{\sigma})$: Return **1** iff

$$\boldsymbol{\sigma}^{e} \textbf{ mod } \boldsymbol{N} == \boldsymbol{\mu}(\boldsymbol{M}) = (\mathbf{0}\ldots\mathbf{0}||\boldsymbol{H}(\boldsymbol{M}))$$

B-smooth

B-smooth: An integer is B-smooth if all its prime factors are smaller than B.

Remember Index Calculus?

Given public key $\boldsymbol{pk} = (\boldsymbol{N}, \boldsymbol{e})$

1. Select a bound $\boldsymbol{y}$ and let $\boldsymbol{S} = (\boldsymbol{p_1}, \ldots, \boldsymbol{p_l})$ be the list of primes smaller than $\boldsymbol{y}$.
2. Find at least $\boldsymbol{l} + \mathbf{1}$ messages $\boldsymbol{M_i}$ such that each $\boldsymbol{\mu}(\boldsymbol{M_i}) = (\mathbf{0} \ldots \mathbf{0} || \boldsymbol{H}(\boldsymbol{M_i}))$ is a product of primes in $\boldsymbol{S}$ (i.e. $\boldsymbol{y}$–smooth).
3. Express one $\boldsymbol{\mu}(\boldsymbol{M_j})$ as a multiplicative combination of the other $\boldsymbol{\mu}(\boldsymbol{M_i})$ by solving a linear system given by the exponent vectors of the $\boldsymbol{\mu}(\boldsymbol{M_i})$ with respect to the primes in $\boldsymbol{S}$.
4. Ask for the signatures on all $\boldsymbol{M_i}, \boldsymbol{i} \neq \boldsymbol{j}$ and forge signature on $\boldsymbol{M_j}$.

Step 3

Recall $\boldsymbol{\mu}(\boldsymbol{M_i}) = (\mathbf{0} \ldots \mathbf{0} || \boldsymbol{H}(\boldsymbol{M_i}))$

1. We can write $\forall \boldsymbol{M_i}, \mathbf{1} \le \boldsymbol{i} \le \boldsymbol{\tau}$: $\boldsymbol{\mu}(\boldsymbol{M_i}) = \prod_{j=1}^{l} \boldsymbol{p_j}^{\boldsymbol{v_{i,j}}}$
2. Associate with $\boldsymbol{\mu}(\boldsymbol{M_i})$ length $\boldsymbol{l}$ vector
$$\boldsymbol{V_i}\left(\boldsymbol{v_{i,1}} \bmod \boldsymbol{e}, \ldots, \boldsymbol{v_{i,l}} \bmod \boldsymbol{e}\right)$$
3. $\boldsymbol{\tau} \ge \boldsymbol{l} + 1$ and there are only $\boldsymbol{l}$ linearly independent length $\boldsymbol{l}$ vectors: We can express one vector as combination of others mod e. Let this be
$$\boldsymbol{V_\tau} = \sum_{i=1}^{\tau-1} \boldsymbol{\beta_i V_i} + \boldsymbol{e\Gamma}\,; \text{ for some } \boldsymbol{\Gamma} = (\boldsymbol{\gamma_1}, \ldots, \boldsymbol{\gamma_l})$$
4. Hence,
$$\boldsymbol{\mu}(\boldsymbol{M_\tau}) = \left(\prod_{j=1}^{l} \boldsymbol{p_j}^{\gamma_j}\right)^{\boldsymbol{e}} \prod_{i=1}^{\tau-1} \boldsymbol{\mu}(\boldsymbol{M_i})^{\beta_i}$$

Step 4

1. Ask for signatures $\boldsymbol{\sigma_i = \mu(M_i)^d \bmod N}$ on $\boldsymbol{M_i}$ for $\boldsymbol{1 \le i < \tau}$

2. Compute:

$$\boldsymbol{\sigma^* = \mu(M_\tau)^d = \left(\prod_{j=1}^{l} {p_j}^{\gamma_j}\right) \prod_{i=1}^{\tau-1} \left(\mu(M_i)^d\right)^{\beta_i} \bmod N}$$

3. Output forgery $\boldsymbol{(\sigma^*, M_\tau)}$

Summing up

- Original attack (Misarsky, PKC'98) works even for more complicated paddings (ISO/IEC 9796-2)
- Attack only works for small $\boldsymbol{n}$! (Complexity depends on $\boldsymbol{l}$ and probability that an $\boldsymbol{n}$-bit number is $\boldsymbol{y}$-smooth).
- But using SHA-1 ($\boldsymbol{n = 160}$) the attack takes much less than $\boldsymbol{2^{50}}$ operations!

There are many ways to make mistakes...

(Similar attacks apply to encryption!)

That's why we want security reductions

The Random Oracle Model

Standard model vs. idealized model

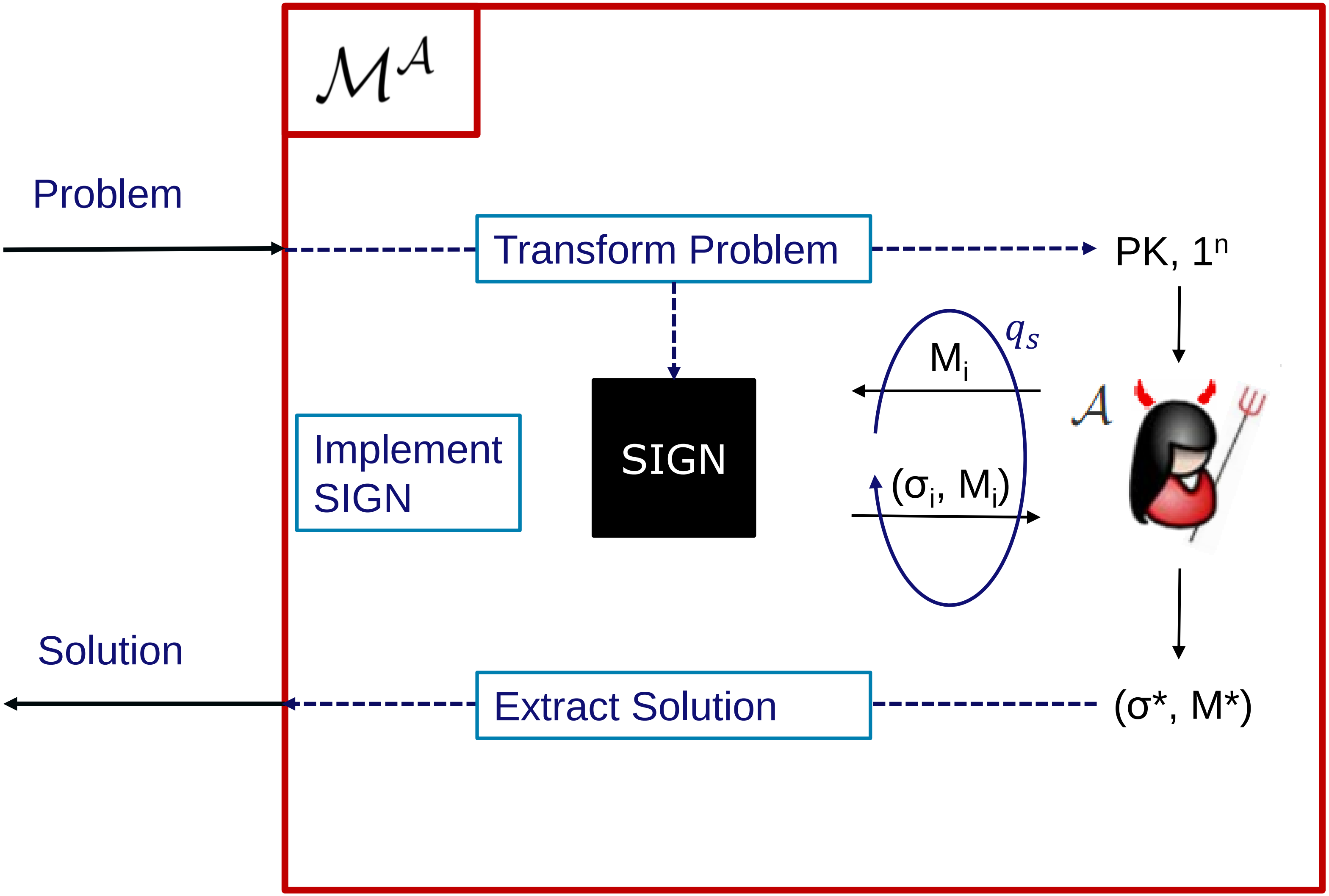
Standard model:

Assume building block has property P (e.g., collision resistance). Use property in reduction.

Idealized model:

Assume a building block behaves perfectly (e.g. hash function behaves like truely random function). Replace building block by an oracle in reduction.

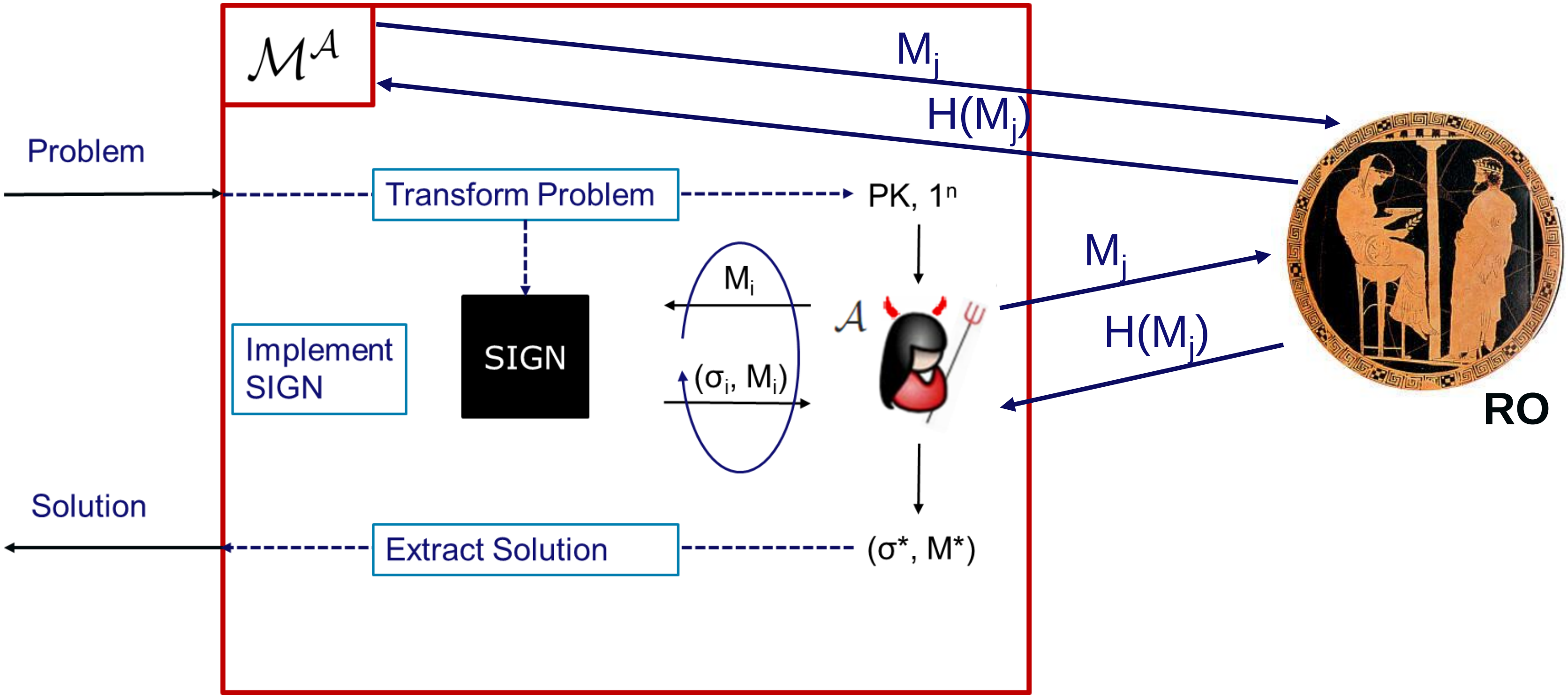
Reduction



$\mathcal{M}^{\mathcal{A}}$
Problem
Transform Problem
PK, 1^n
q_s
M_i
Implement SIGN
SIGN
(σ_i, M_i)
$\mathcal{A}$
Solution
Extract Solution
(σ^*, M^*)

Random Oracle Model (ROM)

- **Idealized Model**
- **Perfectly Random Function**



How to implement RO? (In theory)

"Lazy Sampling":

- Keep list of $(\boldsymbol{x_i}, \boldsymbol{y_i})$
- Given $\boldsymbol{M_j}$, search for $\boldsymbol{x_i} = \boldsymbol{M_j}$
- If $\boldsymbol{x_i} = \boldsymbol{M_j}$ exists, return $\boldsymbol{y_i}$
- Else sample new $\boldsymbol{y}$ from Range, using uniform distribution
- Add $(\boldsymbol{M_j}, \boldsymbol{y})$ to table
- Return $\boldsymbol{y}$

RO

TU/e Technische Universiteit Eindhoven University of Technology

ROM security

- Take scheme that uses cryptographic hash
- For proof, replace hash by RO
  - Different flavors:
    Random function vs. Programmable RO

➢ Heuristic security argument

➢ Allows to verify construction

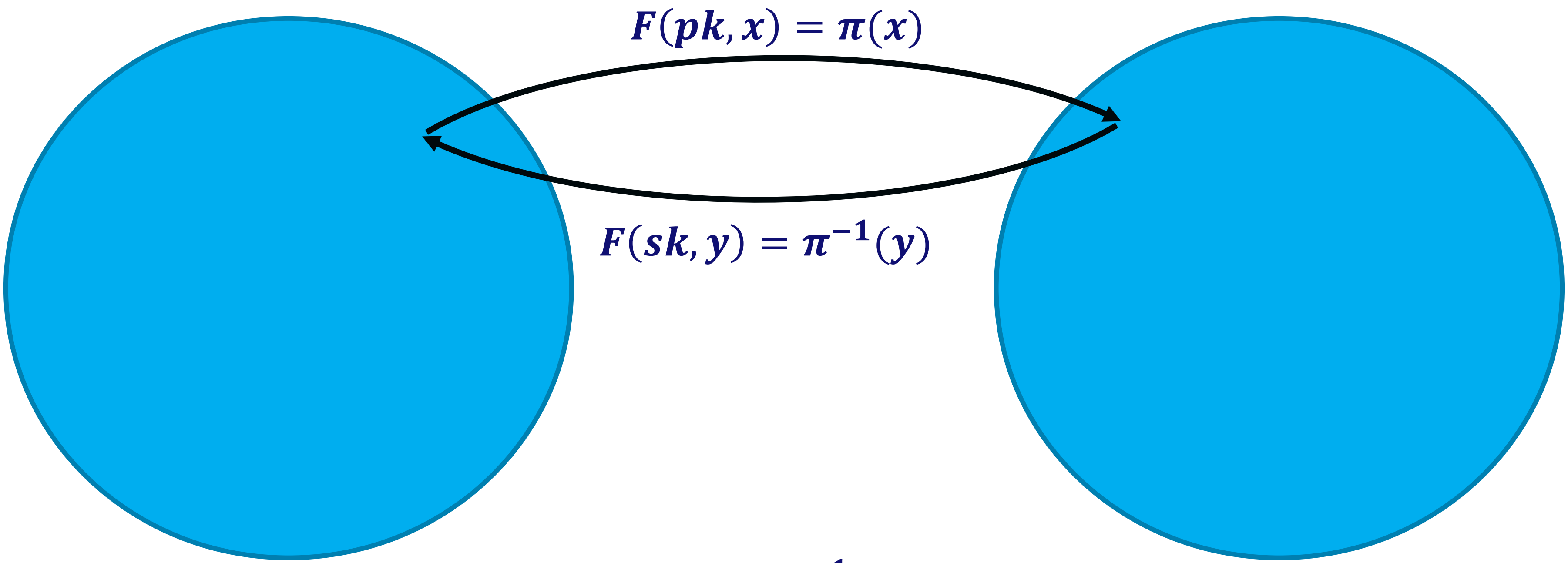
➢ Worked for ”Natural schemes” so far

➢ However: Artificial counter examples exist!

Full Domain Hash Signature Scheme

Technische Universiteit Eindhoven University of Technology

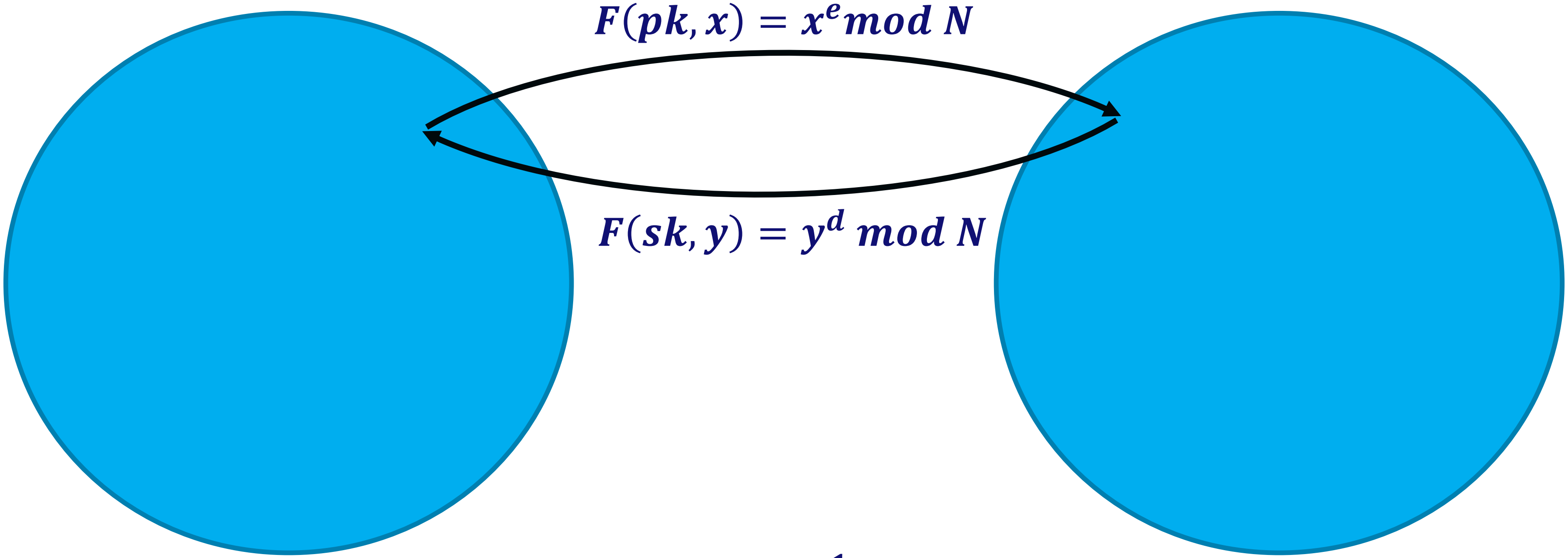
Trapdoor (One-way) Permutation



Computing $\pi^{-1}(y)$
without knowledge of **sk**
computationally hard

RSA Trapdoor (One-way) Permutation

$(N, e, d) \leftarrow \mathbf{GenRSA}(1^k)$; $pk = (N, e)$; $sk = d$



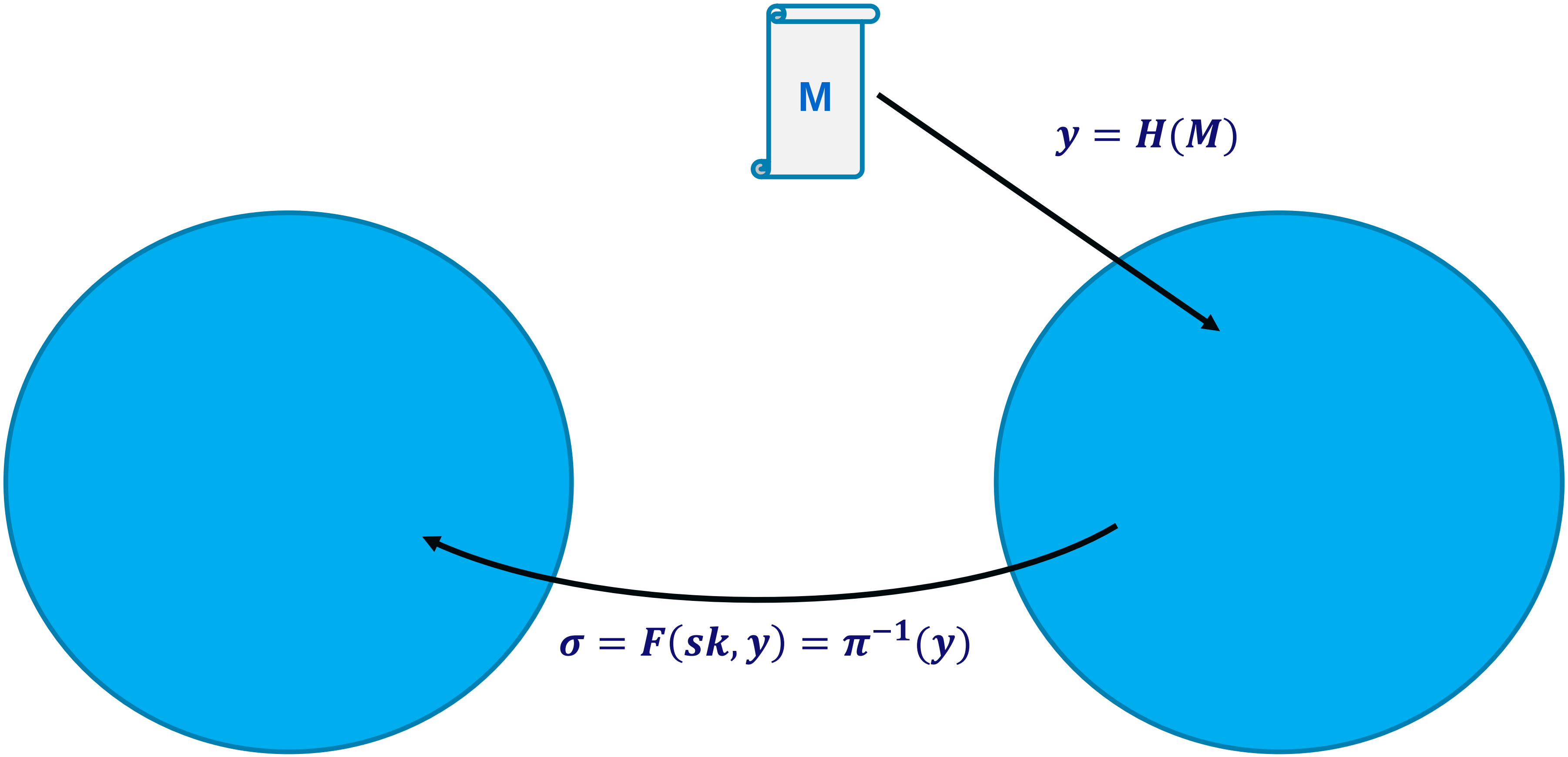
$$F(pk, x) = x^e \mathit{mod}\ N$$

$$F(sk, y) = y^d\ \mathit{mod}\ N$$

Computing $\pi^{-1}(y)$
without knowledge of **sk**
computationally hard if
RSA Assumption holds

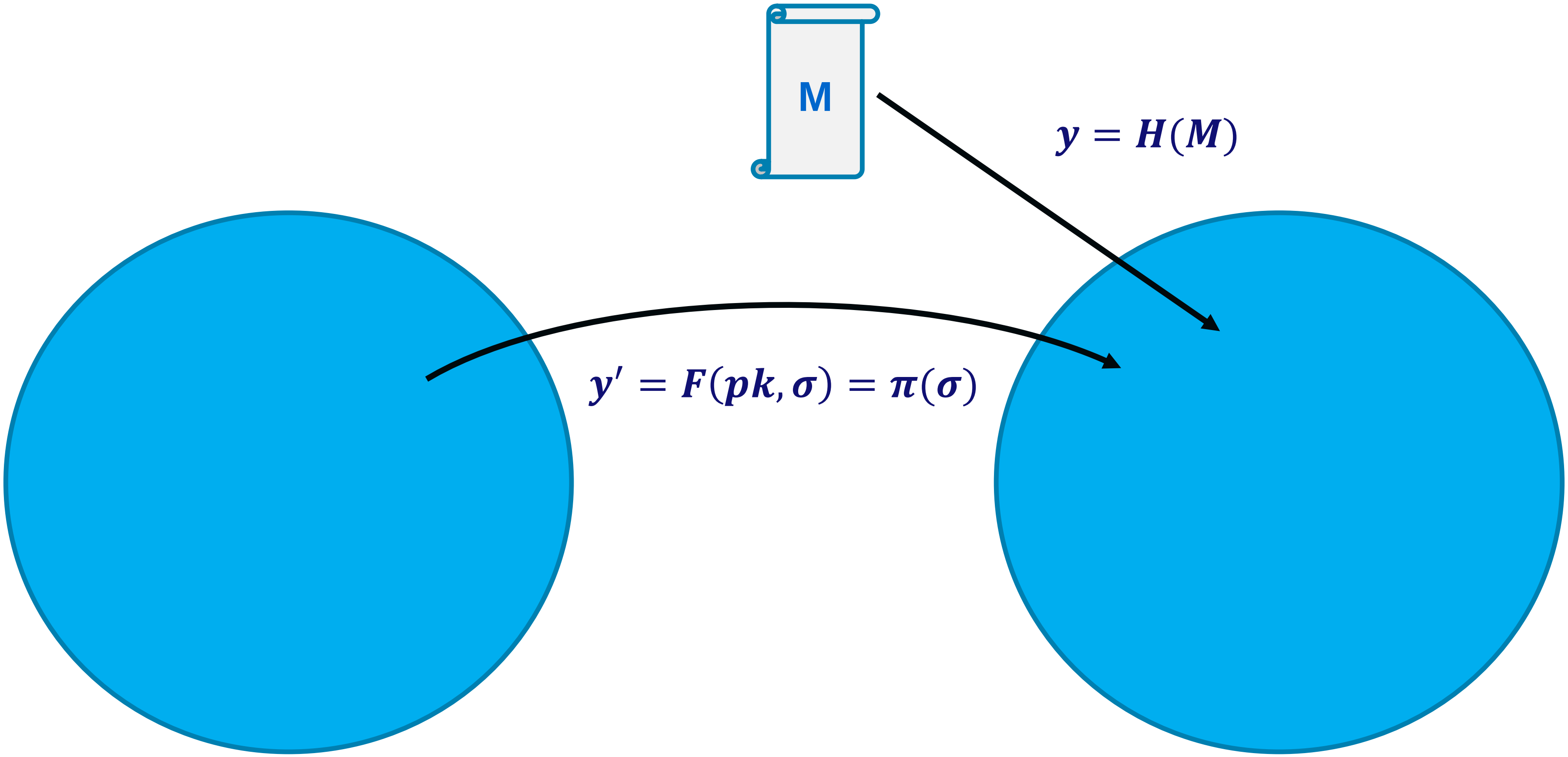
Technische Universiteit
Eindhoven
University of Technology

Generic FDH: Sign



$$\begin{aligned} \sigma &= Sign(sk, M) \\ &= \pi^{-1}(H(M)) \\ &= F(sk, H(M)) \end{aligned}$$

Generic FDH: Verify



$$Verify(pk, M, \sigma):$$

check $y = H(M) == \pi(\sigma) = F(pk, \sigma) = y'$

Technische Universiteit Eindhoven University of Technology

RSA-PFDH

- Randomized FDH

- Simplified RSA-PSS
  - Standardized in PKCS #1 v2

    (slightly different randomization)

- Tight Reduction from RSA Assumption in ROM

RSA-PFDH

Assume Hashfunction $H: \{0,1\}^* \rightarrow \mathbb{Z}_N^*$

$\mathbf{KeyGen}(1^k)$: Run $(N, e, d) \leftarrow \mathbf{GenRSA}(1^k)$.

Return (pk, sk) with $pk = (N, e), sk = d$.

$\mathbf{Sign}(sk, M)$: Sample $r \overset{\$}{\leftarrow} U_\kappa$; Compute $\mathbf{y} = H(r||M)$

Return $\boldsymbol{\sigma} = (r, y^d \bmod N)$

$\mathbf{Verify}(pk, M, \sigma)$: Return **1** iff $\sigma^e \bmod N == H(r||M)$

RSA-PFDH Security

If the RSA Assumption holds, RSA-PFDH is existentially unforgeable under adaptive chosen message attacks in the ROM.

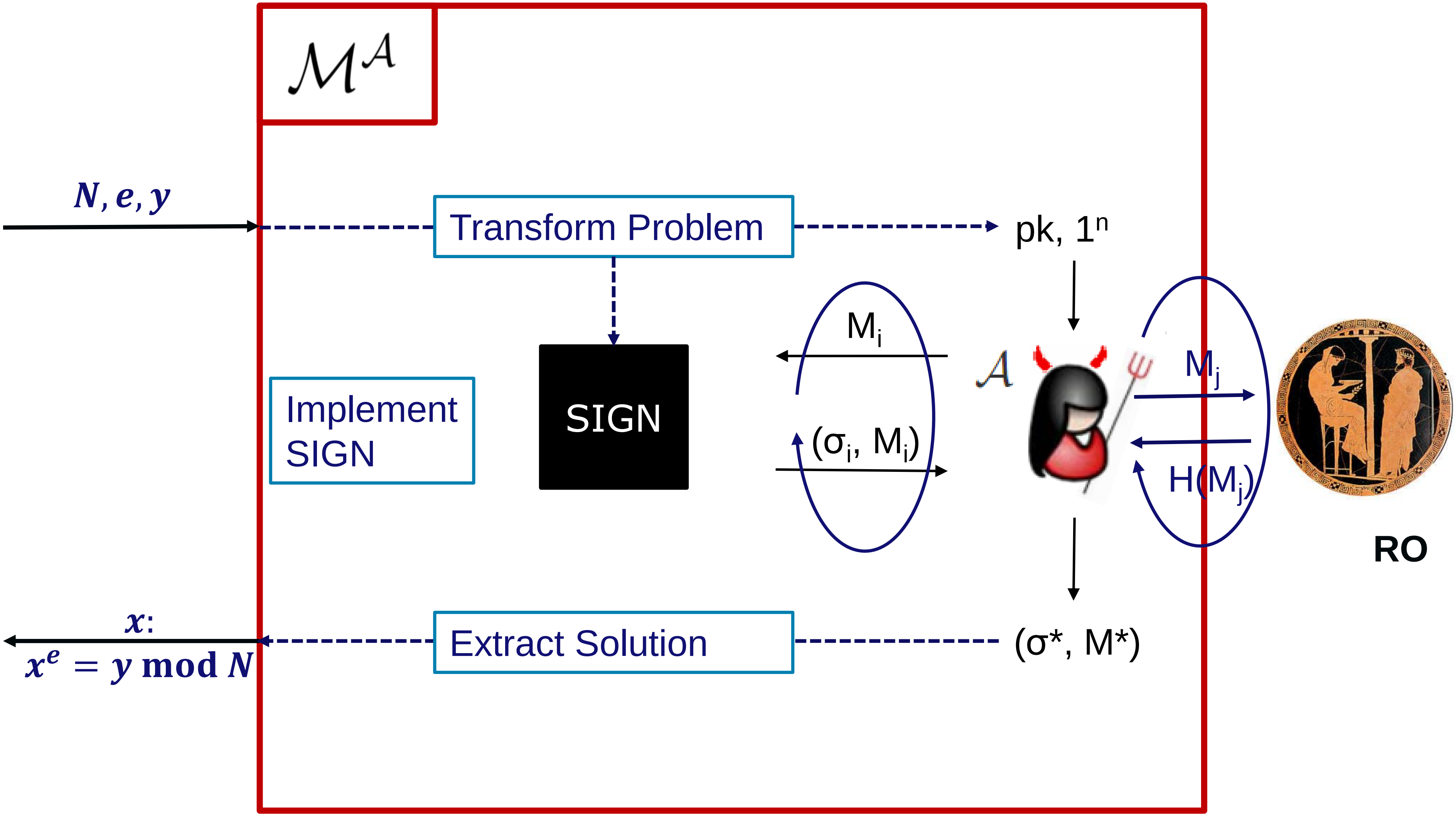
Proof

TODO:

Show that any forger $\mathbf{A}$ against RSA-PFDH can be used to break the RSA Assumption with approx. the same time and success probability.

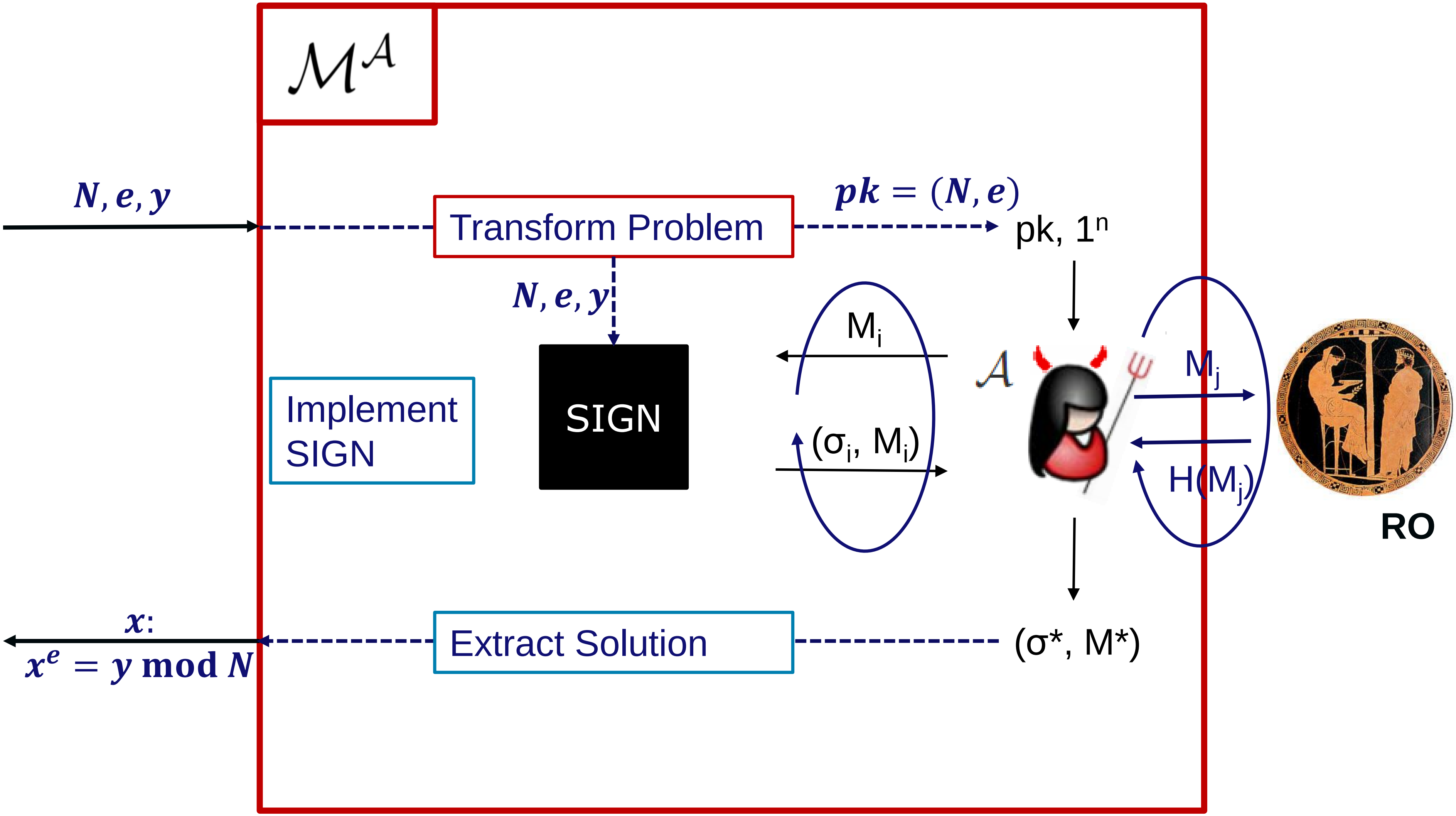
”Given a forger $\mathbf{A}$ against RSA-PFDH with success probability $\boldsymbol{\varepsilon}$, we construct an oracle Machine $\mathbf{M}^{\mathbf{A}}$ against the RSA Assumption that succeeds with probability $\boldsymbol{\varepsilon/4}$.”

Reduction



$\mathcal{M}^{\mathcal{A}}$
N, e, y
Transform Problem
pk, 1ⁿ
Implement SIGN
SIGN
M_i
(σ_i, M_i)
$\mathcal{A}$
M_j
$H(M_j)$
RO
x:
$x^e = y \bmod N$
Extract Solution
(σ*, M*)

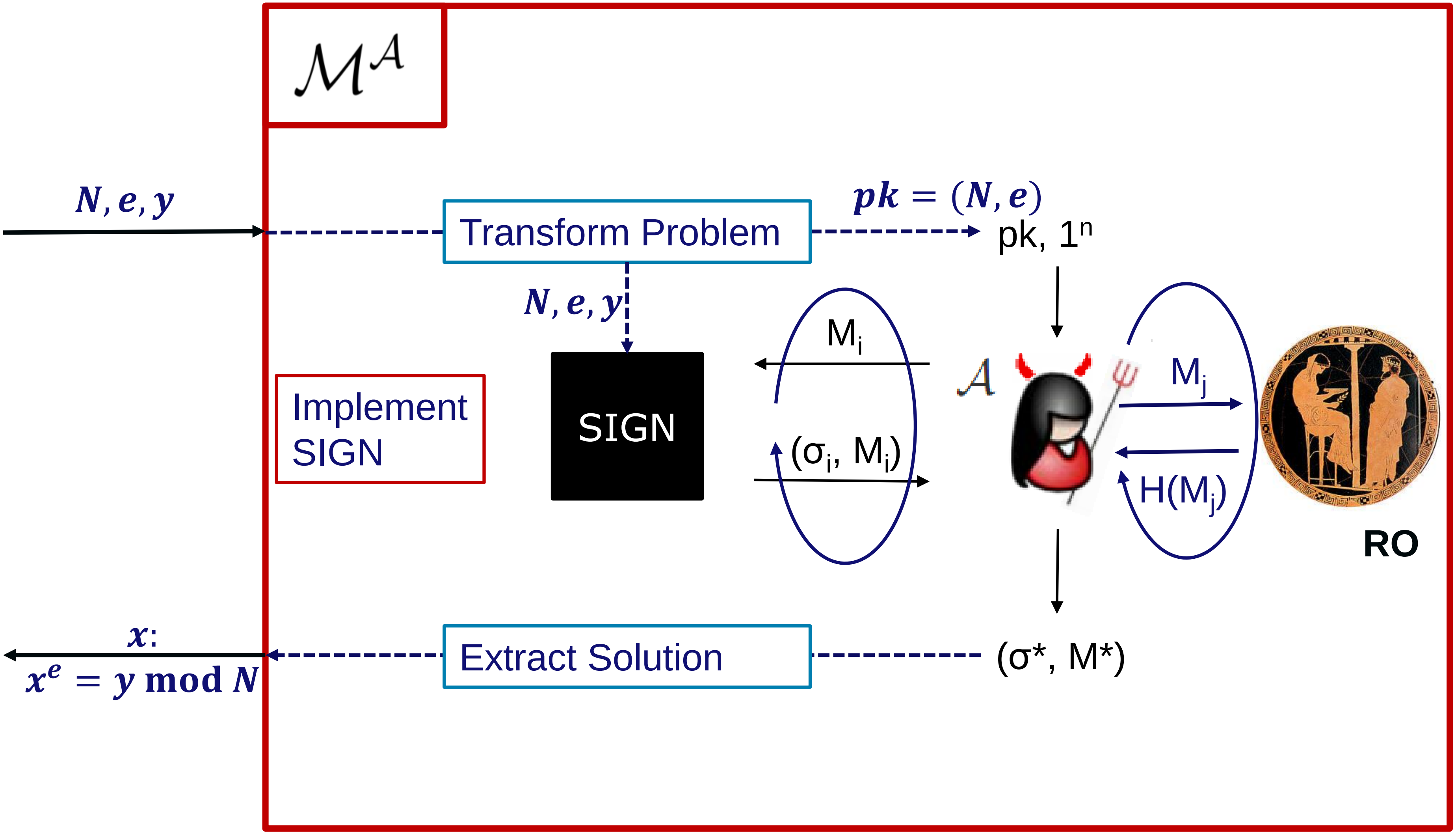
Reduction: Transform Problem



The RO trick

Simulate RO such that you can answer SIGN-queries.

Reduction: Implement SIGN



Implement SIGN – Simulate RO

- Keep table of tripples $(*,*,*)$
- W

1.
2.

z^d together with x allows to extract y^d

3. If $r \in L_M$ then let i be such that $r = r_{M,i}$. Choose random $x_{M,i} \in \mathbb{Z}_N^*$ and return the answer $\mathbf{z} = x_{M,i}{}^e \mathbf{mod}\ N$. Store $(r||M, x_{M,i}, z)$ in the table. (RP)
4. If $r \notin L_M$, choose random $x \in \mathbb{Z}_N^*$ and return the answer $z = yx^e \mathbf{mod}\ N$. Store $(r||M, x, z)$ in the table.

Implement SIGN

- When **A** requests some message $\boldsymbol{M}$ to be signed for the $\boldsymbol{i}$ th time:
  - let $\boldsymbol{r_{M,i}}$ be the $\boldsymbol{i}$ th value in $\boldsymbol{L_M}$ and
  - compute $\mathbf{z} = \mathbf{H}(\boldsymbol{r_{M,i}}||\boldsymbol{M})$ using **RO.**
  - Let $(\boldsymbol{r}||\boldsymbol{M}, \boldsymbol{x_{M,i}}, \boldsymbol{z})$ be the corresponding entry in the **RO** table.
  - Output signature $(\boldsymbol{r_{M,i}}, \boldsymbol{x_{M,i}})$**.**

Observation

- All **SIGN** queries can be answered!
- **SIGN** queries are answered using hash

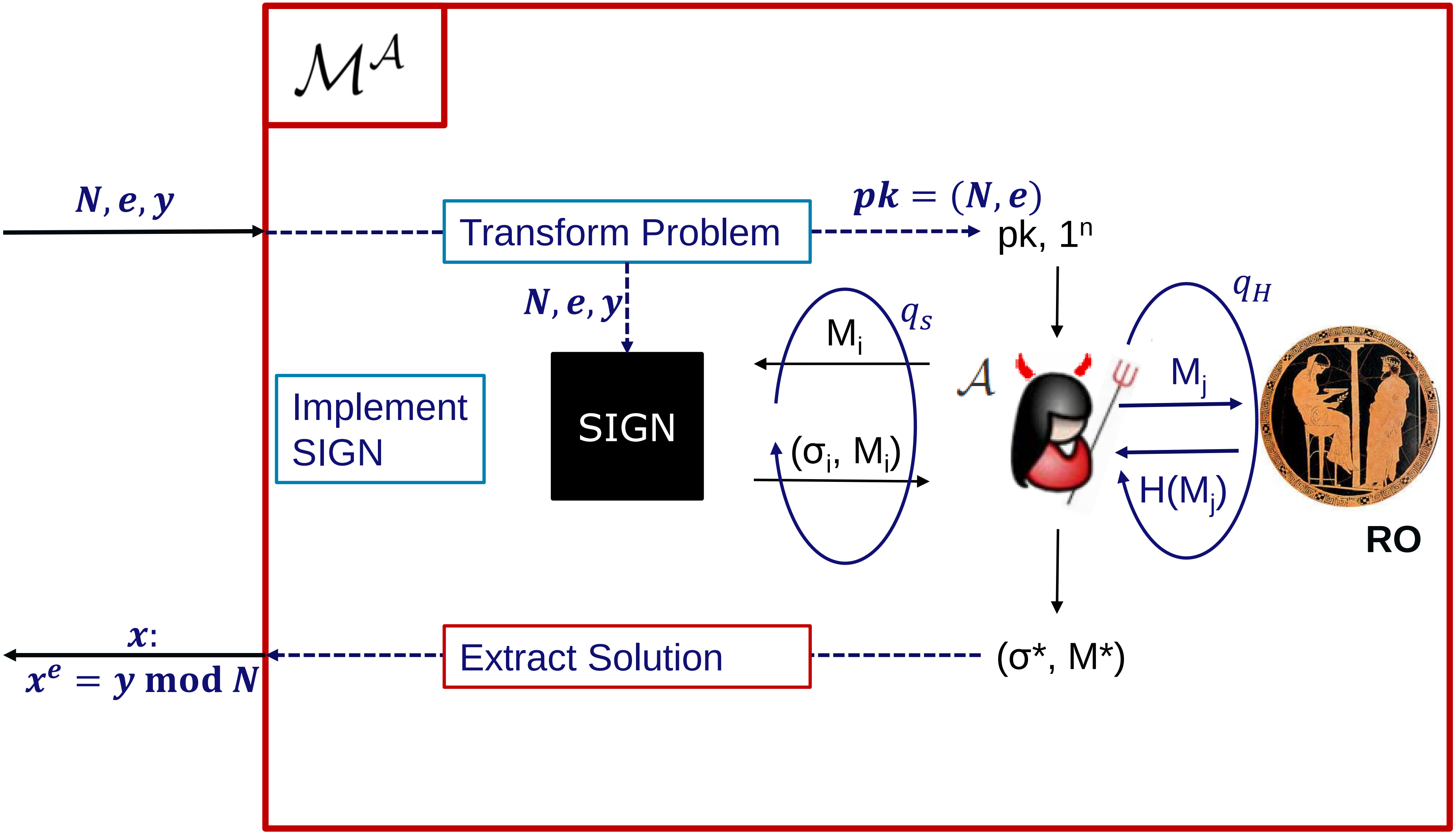
$$\mathbf{H}(r_{M,i}||M) = \mathbf{z} = x_{M,i}{}^{e} \mathbf{mod}\ N$$

  - Signature $(r_{M,i}, x_{M,i})$ known by programming **RO**
- All other hash queries are answered with

$$\mathbf{H}(r||M) = z = yx^{e} \mathbf{mod}\ N$$

  (with high probability).
  - Signature not known!
  - **BUT:** Allows to extract solution from forgery!
- Note: Any **A** with non-negl. success probability has to query **RO** for digest of forgery message!

Reduction: Extract Solution



Reduction: Extract Solution

- If **A** outputs a forgery $(\boldsymbol{M}^*, (\boldsymbol{r}^*, \boldsymbol{\sigma}^*))$:
  - If $\boldsymbol{r}^* \in \boldsymbol{L}_{\boldsymbol{M}^*}$ abort.
  - Else, let $(\boldsymbol{r}^* || \boldsymbol{M}^*, \boldsymbol{x}, \boldsymbol{z})$ be the corresponding entry of the table.
  - Output $\frac{\boldsymbol{\sigma}^*}{\boldsymbol{x}} \bmod \boldsymbol{N}$.

- Note:

$$\left(\frac{\boldsymbol{\sigma}^*}{\boldsymbol{x}}\right)^{\boldsymbol{e}} = \frac{\boldsymbol{\sigma}^{*\boldsymbol{e}}}{\boldsymbol{x}^{\boldsymbol{e}}} = \frac{\boldsymbol{H}(\boldsymbol{r}^* || \boldsymbol{M}^*)}{\boldsymbol{x}^{\boldsymbol{e}}} = \frac{\boldsymbol{y}\boldsymbol{x}^{\boldsymbol{e}}}{\boldsymbol{x}^{\boldsymbol{e}}} = \boldsymbol{y} \bmod \boldsymbol{N}$$

$$\Longrightarrow \frac{\boldsymbol{\sigma}^*}{\boldsymbol{x}} = \sqrt[\boldsymbol{e}]{\boldsymbol{y}} \bmod \boldsymbol{N}$$

Analysis

- **Transform Problem:**
  - Succeeds always
  - Generates exactly matching distribution
- **Implement SIGN / RO:**
  - Succeeds always (we choose $\boldsymbol{r}$)
  - Generates exactly matching distribution:
    - **RO**: Outputs are uniform in $\mathbb{Z}_N^*$
    - **SIGN**: Follows from **RO**
- **Extract Solution:**
  - Succeeds iff **A** succeeds AND
  - $\boldsymbol{r}^* \notin \boldsymbol{L}_{\boldsymbol{M}^*} \Rightarrow \mathbf{p} = \mathbf{Pr}[\boldsymbol{r}^* \notin \boldsymbol{L}_{\boldsymbol{M}^*}] = (\mathbf{1} - \mathbf{2}^{-\boldsymbol{\kappa}})^{\boldsymbol{q}_s}$
    Setting $\boldsymbol{\kappa} = \mathbf{log}_2\, \boldsymbol{q}_s$: $\boldsymbol{p} \geq \frac{1}{4}$ assuming $\boldsymbol{q}_s \geq \mathbf{2}$

What have we shown?

- We can turn any forger $\mathbf{A}$ against RSA-PFDH with success probability $\boldsymbol{\varepsilon}$ into an algorithm $\mathbf{M}^{\mathbf{A}}$ that solves the RSA problem with probability $\boldsymbol{\varepsilon/4}$.

- **In reverse:**
If there exists no efficient algorithm to solve the RSA problem with probability $\geq \boldsymbol{\varepsilon}$ then there exists no efficient forger against RSA-PFDH that succeeds with probability $\geq \mathbf{4}\boldsymbol{\varepsilon}$.

- As proof is in ROM we have to add
"... As long as the used hash function behaves like a RO."

Technische Universiteit Eindhoven University of Technology

How to implement RO? (In practice)

Formerly:

- Use hash function + PRG
- E.g. SHA2 in couter mode /
- SHA2-HMAC in counter mode keyed with SHA2(M)

Today:

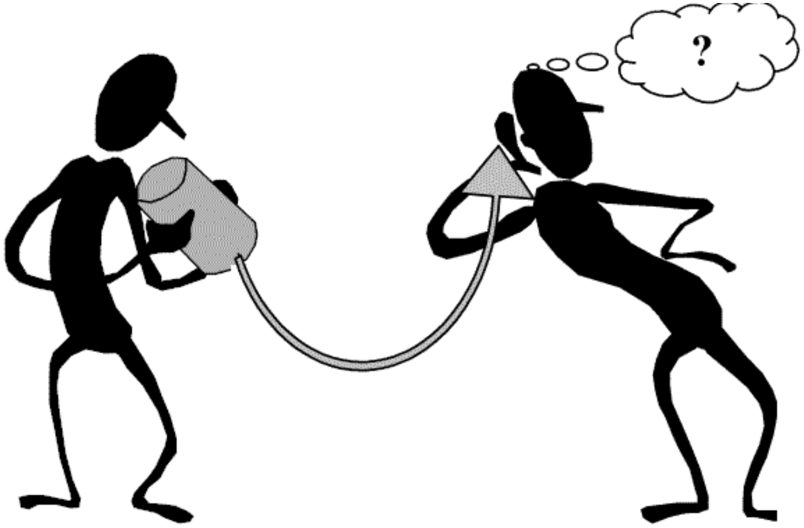
- Use XOF
- E.g. SHAKE128

RO

Conclusion

- Ad Hoc constructions problematic
  - Blinding / Index Calculus
- Proofs (even in ROM) allow to check construction
- There is one standardized RSA Sig with proof
- Similar situation for DSA (ROM proof)

Thank you!
Questions?



Technische Universiteit Eindhoven University of Technology