Hiding in Plain Sight: Memory-tight Proofs via Randomness Programming

Ashrujit Ghoshal
University of Washington

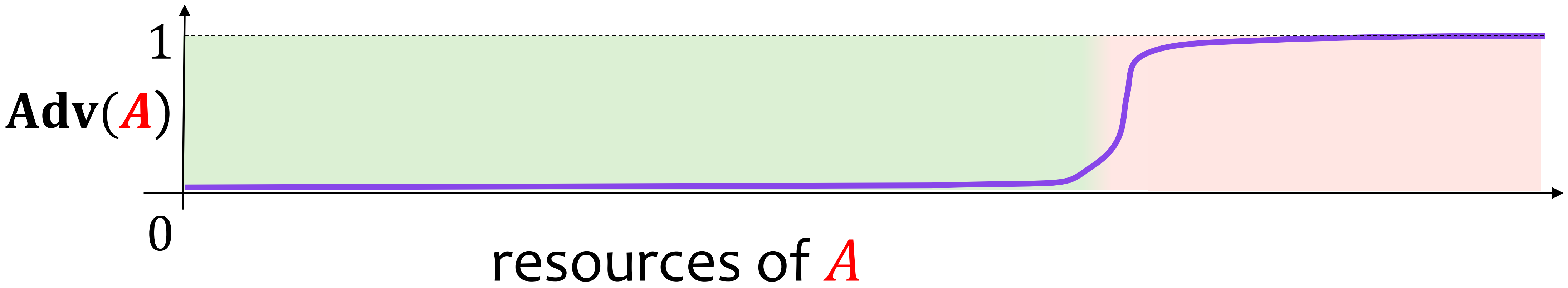
Riddhi Ghosal
UCLA

Joseph Jaeger
Georgia Tech

Stefano Tessaro
University of Washington

Eurocrypt 2022

Concrete security theorems



Traditionally: resources of A = time t

More accurate: resources of A = time t, memory S

Security reductions

classical cryptographic reduction

(T, ε)-hard Π → (T', ε')-secure Σ

wanted
- time tightness $T \approx T'$
- advantage tightness $\varepsilon \approx \varepsilon'$

memory-aware reductions

(T, S, ε)-hard Π → (T', S', ε')-secure Σ

New goal: memory tightness $S \approx S'$

[ACFK17]

Memory-tightness matters

Example: Π = Dlog in 4096-bit prime field

Plausible assumption

$(T = 2^{160}, S = 2^{70}, \varepsilon)$-hard

Π

memory-tight

$(T = 2^{160}, S = 2^{70}, \varepsilon)$-secure

Σ

$(T = 2^{160}, S = 2^{160}, \varepsilon)$-hard

Π

not memory-tight

$(T = 2^{160}, S = 2^{70}, \varepsilon)$-secure

Σ

Known to be false!

Memory-tight reductions are tricky & bizarre!

- Impossibility results [ACFK17,WMHT18,GT20,GJT20]
- Possibility results [ACFK17,Bhattacharya20,GJT20,DGJL21]

Generic impossibility bypassed by specific schemes/settings

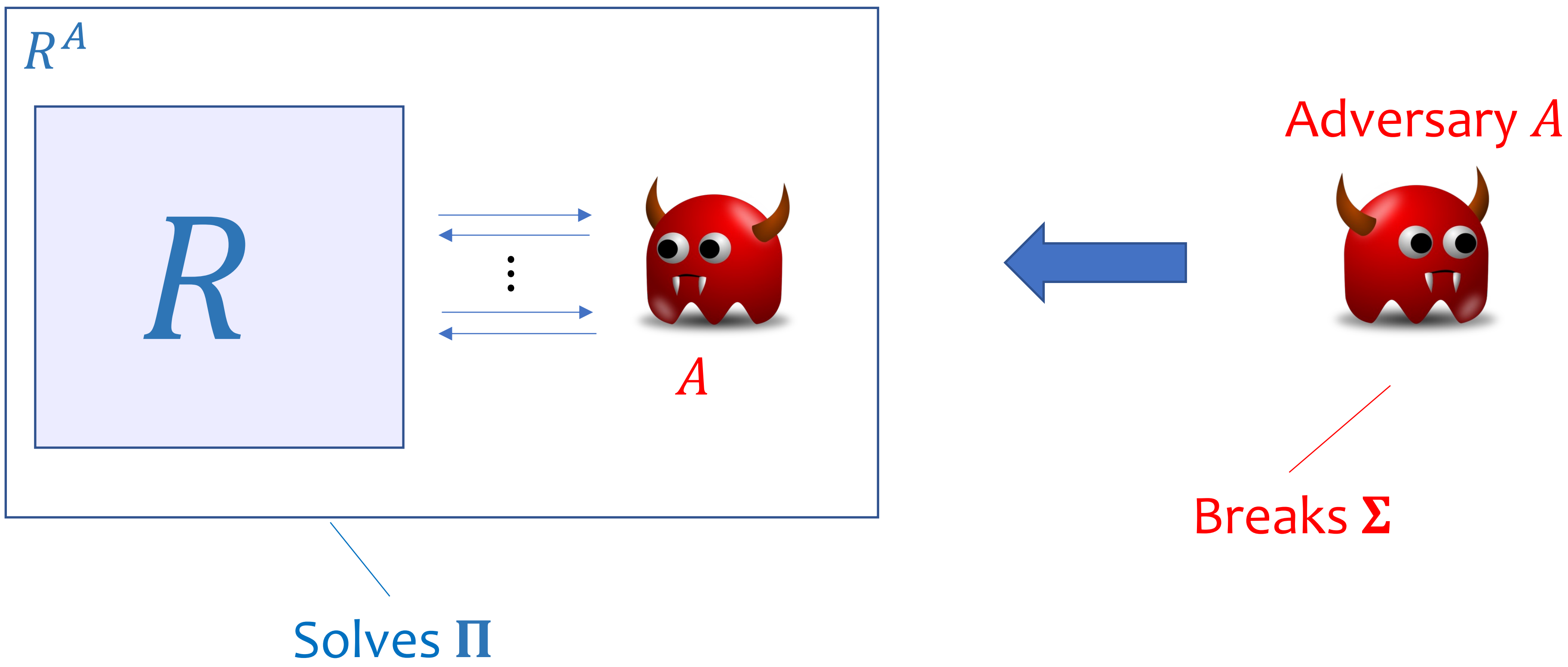
Impossibility bypassed by tweaking schemes

[This work] → Ability to give memory-tight reductions strongly coupled with definitional choices

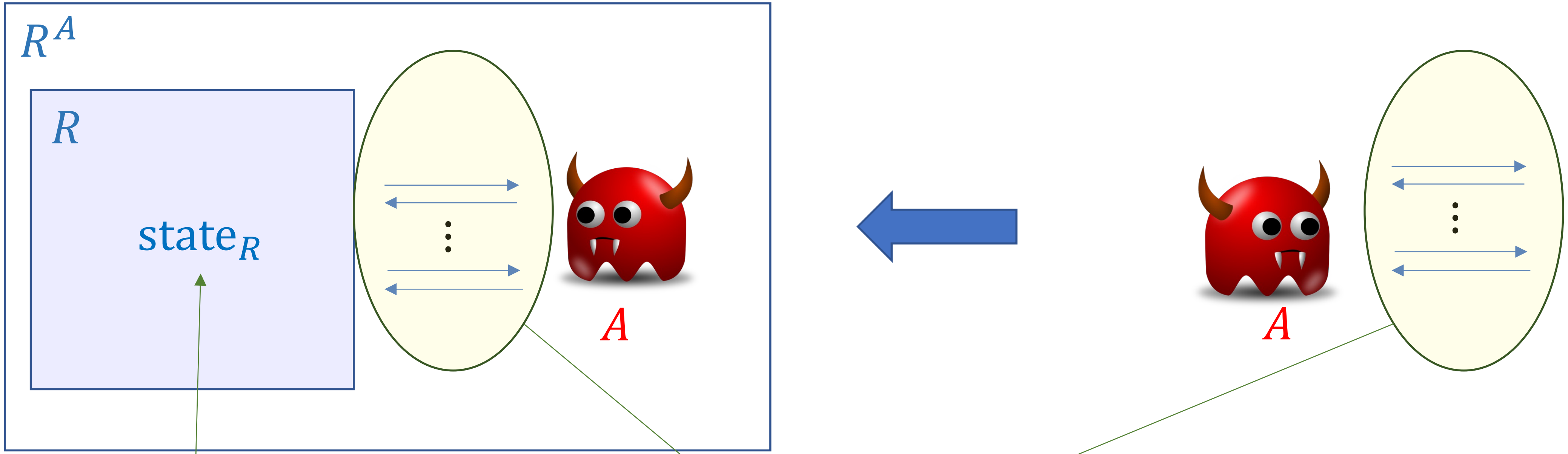
This talk: new class of techniques for memory-tight reductions

Theorem: (T, S, ε)-hard $\mathbf{\Pi}$ $\Rightarrow$ (T', S', ε')-secure $\mathbf{\Sigma}$

Proof:



Memory tightness: mem(R^A) ≈ mem(A)



Need to be indistinguishable to A

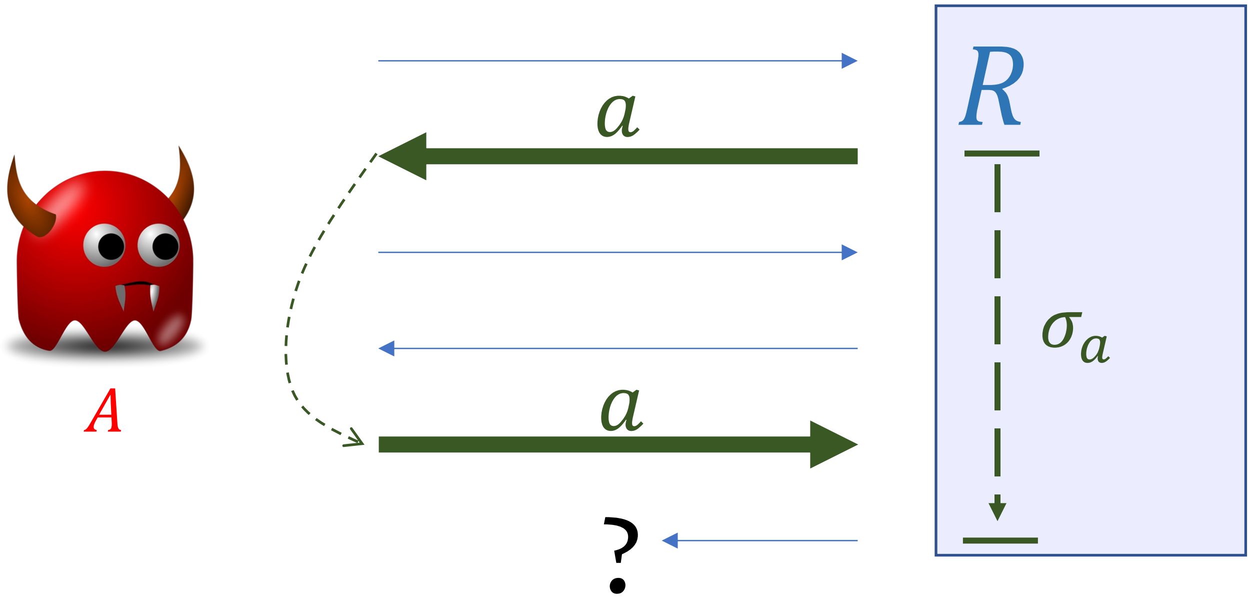
Simulation often requires state

Memory tightness: $|state_R|$ small!

Key observation

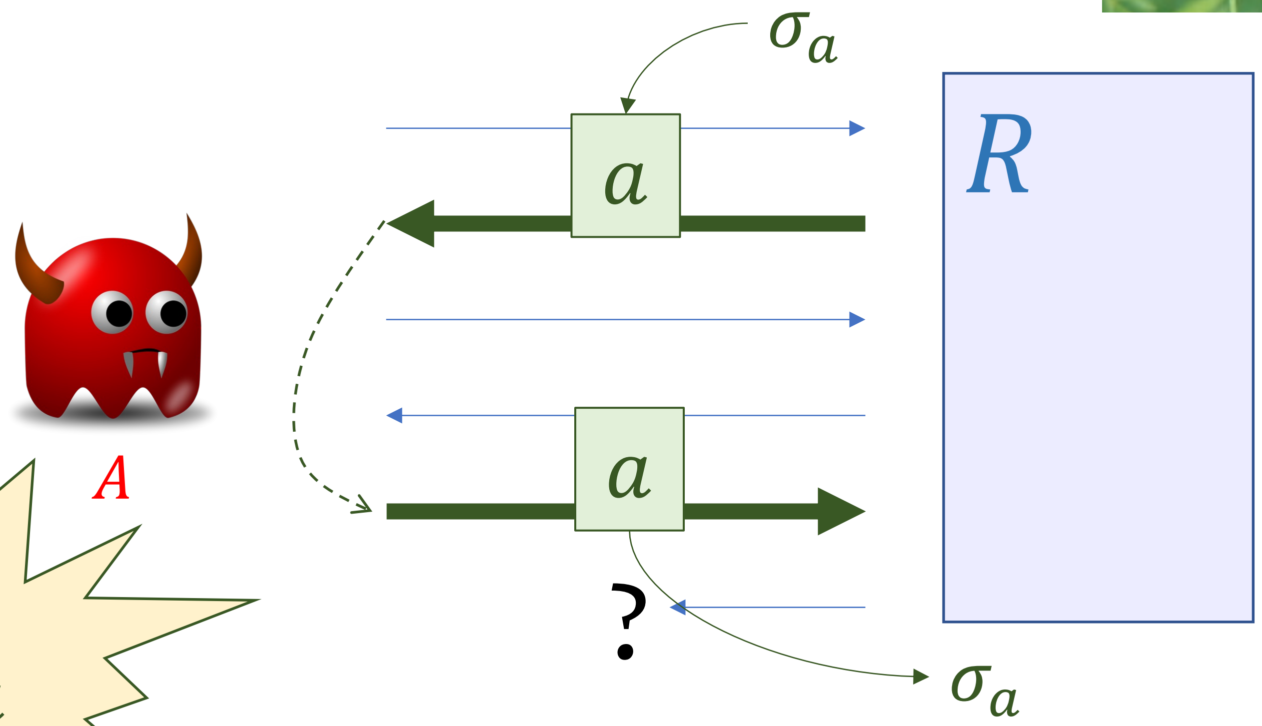
"*For* <u>*some*</u> *reductions, each of* R's answers a to A requires holding some state σ_a *to be used only if* a *is sent back to* R."

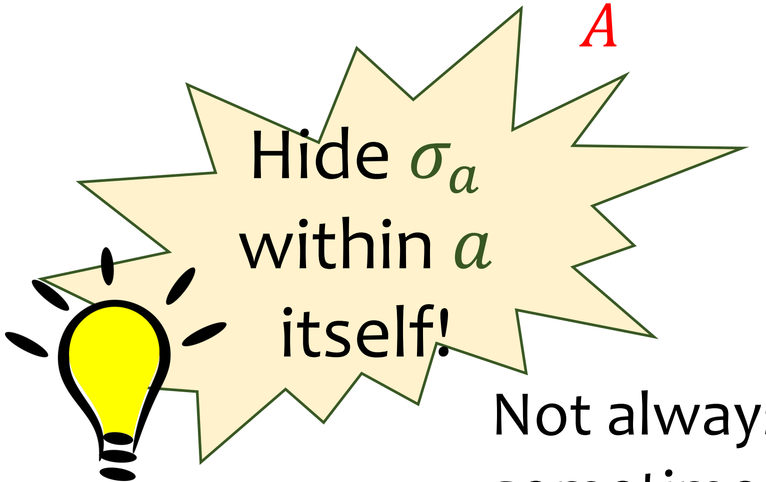
[This work]



How can we avoid storing the state σ_a???

Idea: hiding in plain sight! [This work]





Not always possible, but sometimes a has enough redundancy!

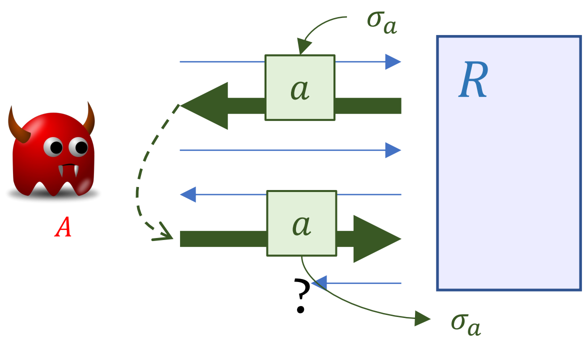
This talk: three techniques

1. Efficient tagging
2. Inefficient tagging
3. Message encoding

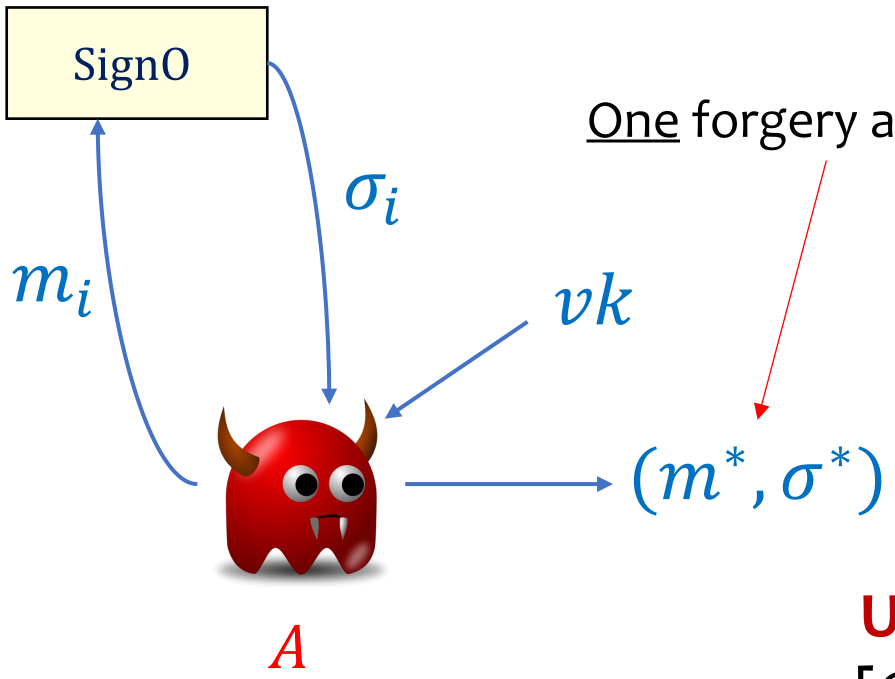
$\sigma_a \in \{0,1\}$, *recoverable in time* $O(1)$

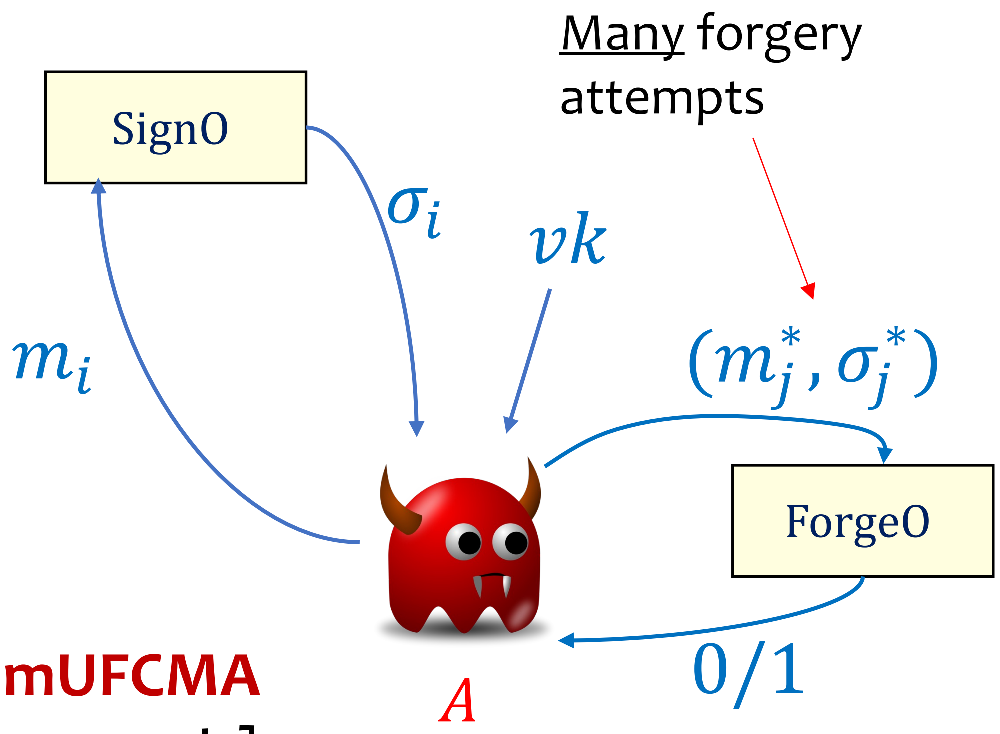
$\sigma_a \in \{0,1\}$, *recoverable in time* $\omega(1)$

Bounded-length σ_a, *recoverable in time* $O(1)$



Digital signatures vs memory-tightness



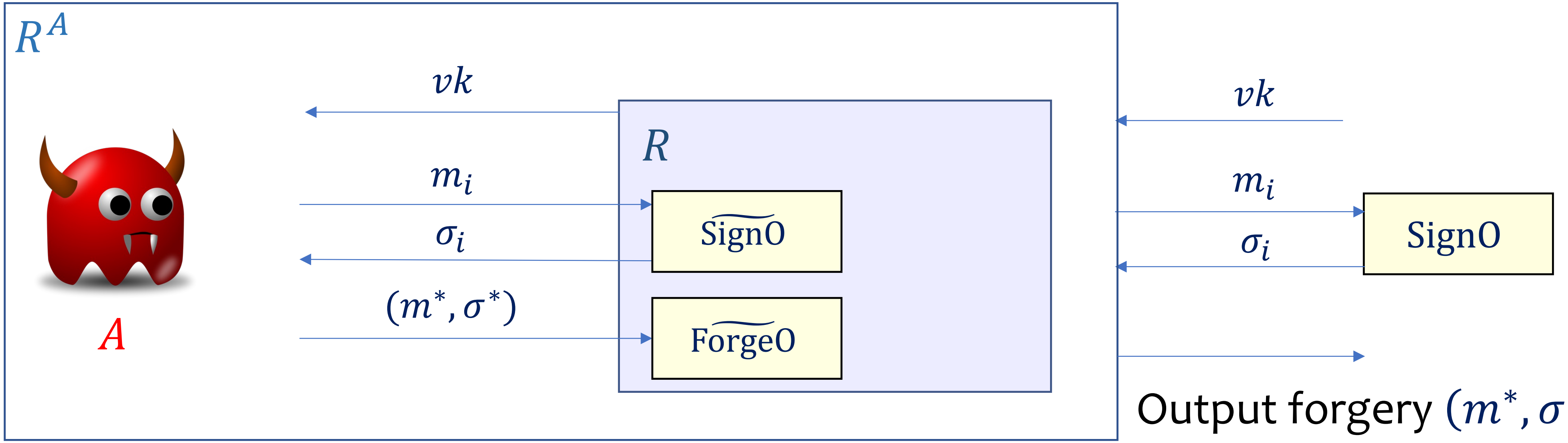


UFCMA $\Leftrightarrow$ **mUFCMA**
[Cryptography 101 result]

Theorem. [ACFK17] Reduction **UFCMA ⇒ mUFCMA** cannot be both memory- and advantage-tight!

Let's see why …

Let us recall the UFCMA ⇒ mUFCMA reduction



Output forgery (m^*, σ^*) iff

1. σ^* valid for m^*
2. m^* is "fresh"

Efficient tagging!

Option I: R… prior m_i's $\rightarrow$ **not memory-tight**

Option II: Guess if fresh $\rightarrow$ **not advantage-tight**

We use efficient tagging to obtain the following:

DS

UFCMA secure digital signature scheme

generic transform →

RDS

mUFCMA secure digital signature scheme

Memory & advantage tight!

<u>RDS.Sign(sk, m)</u>

$r \overset{\$}{\leftarrow} \{0,1\}^{\ell}$

$\sigma \leftarrow$ DS.Sign($sk, (m||r)$)

Return (σ, r)

<u>RDS.Ver($vk, m, (\sigma, r)$)</u>

Return DS. Ver($vk, (m||r), \sigma$)

Generalizes PFDH [Coron01]

Idea: Reduction will add tag in r to identify non-fresh query

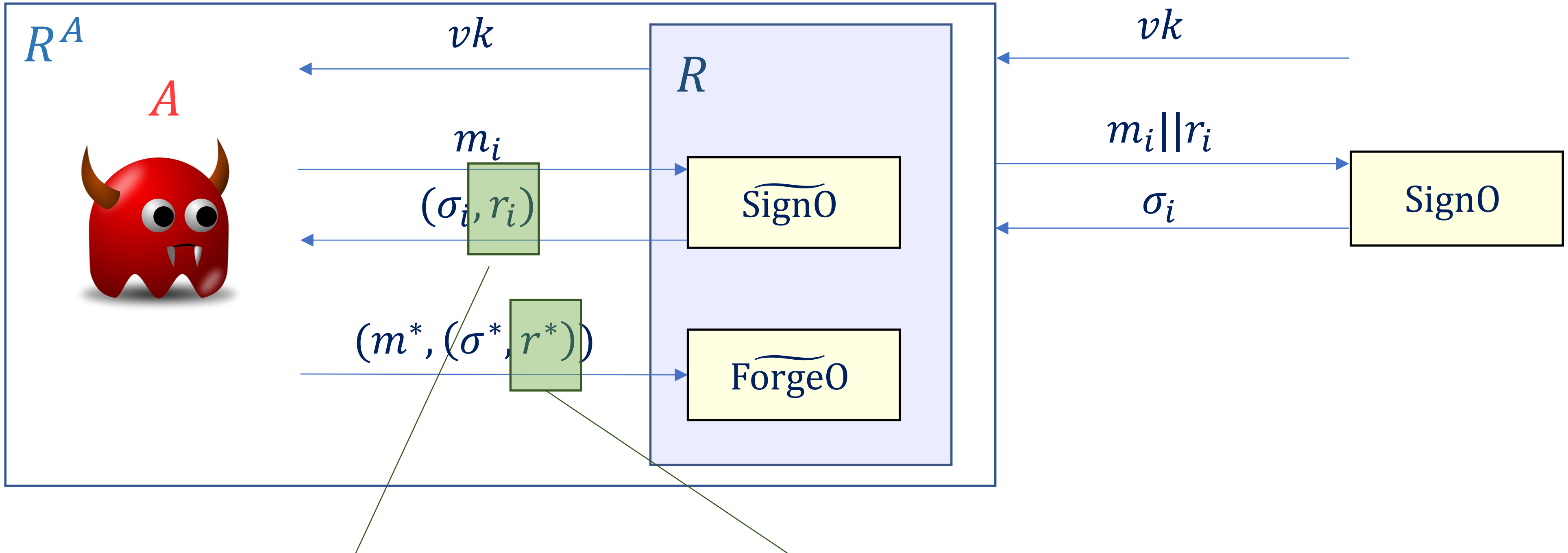
Theorem. [This work]

UFCMA secure DS ⇒ mUFCMA secure RDS, memory/advantage-tightly

[DGJL21] (concurrent work) for certain DS,
strong UFCMA* secure DS ⇒ strong mUFCMA secure RDS, memory/advantage-tightly

Key idea

RDS.Sign(sk, m)
$r \overset{\$}{\leftarrow} \{0,1\}^{\ell}$
$\sigma \leftarrow$ DS.Sign$(sk, (m||r))$
Return (σ, r)



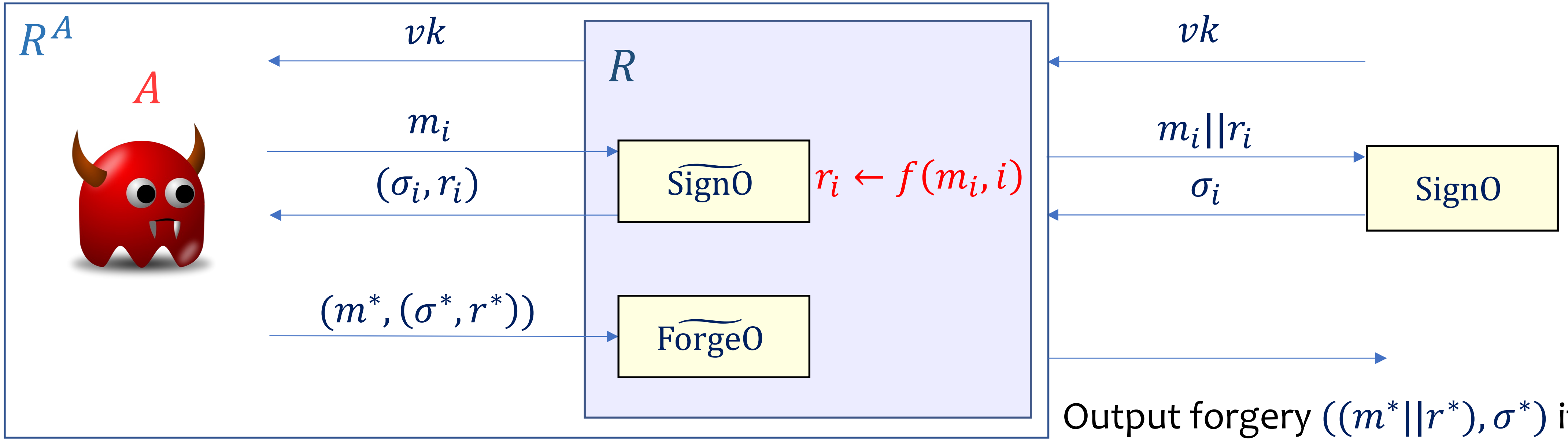
Idea 1: hide the info that m_i is not "fresh" in r_i

Idea 2: Use the hidden info in r^* to determine whether to output forgery

Concretely: efficient tagging

$f : \mathcal{M} \times [q] \rightarrow \{0,1\}^{\ell}$

1) Random
2) Tweakable
3) Injective



Output forgery $((m^* || r^*), \sigma^*)$ iff

1. (σ^*, r^*) is valid signature for m^*
2. $f^{-1}(m^*, r^*) \notin [q]$

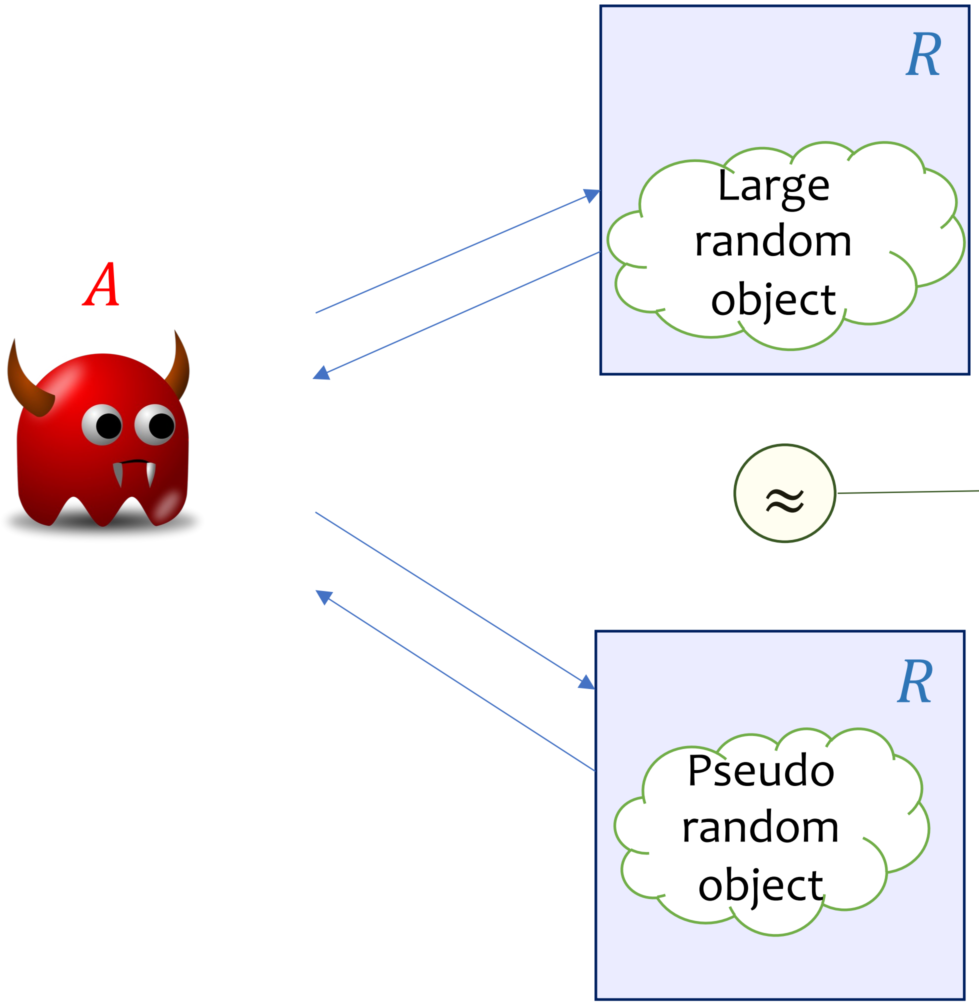
\# (signing queries)

Suppose (σ^*, r^*), is a valid signature for m^*

If (m^*, r^*) queried to SignO
$\Rightarrow \exists i \in [q], (m^*, r^*) = (m_i, r_i)$
$\Rightarrow f^{-1}(m^*, r^*) = i \in [q]$

If (m^*, r^*) not queried to SignO
$\Rightarrow \forall i \in [q], (m^*, r^*) \neq (m_i, r_i)$
$\Rightarrow f^{-1}(m^*, r^*) \notin [q]$ w.h.p

$S_{R^A} \approx S_A + S_f$ → how large is this?



≈ → invoking pseudorandomness

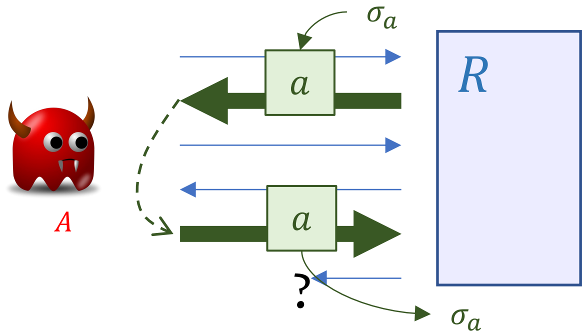
Instantiating pseudorandom object requires **little memory**, e.g., tweakable injective PRF from a blockcipher (CMC) [HR03]

This talk: three techniques

1. Efficient tagging
2. Inefficient tagging
3. Message encoding

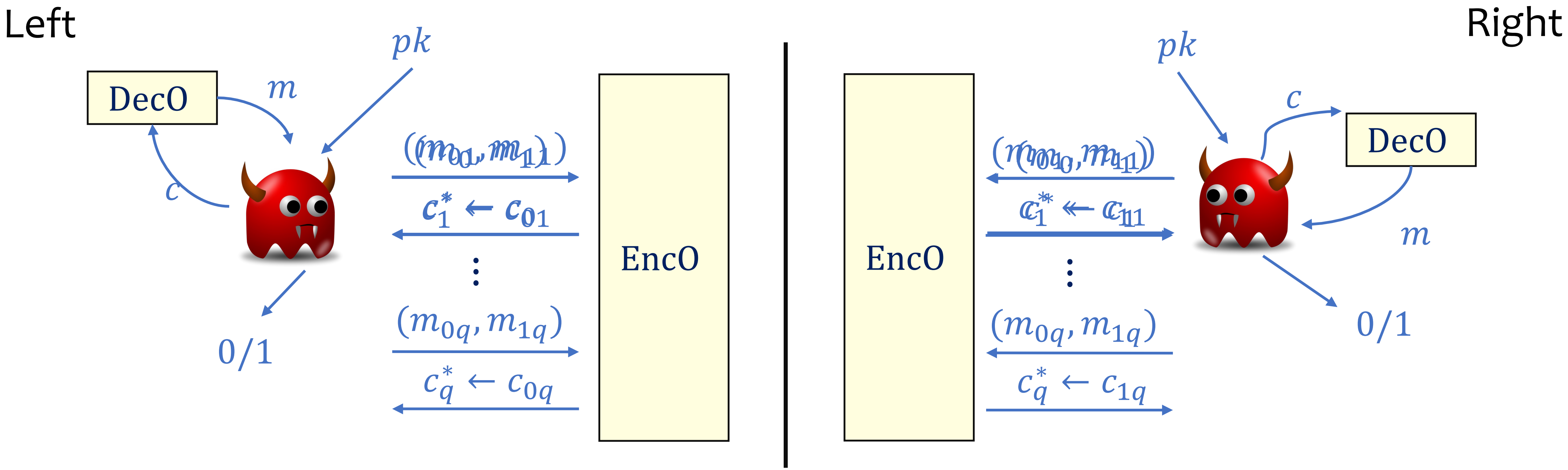
$\sigma_a \in \{0,1\}$, *recoverable in time* $O(1)$

$\sigma_a \in \{0,1\}$, *recoverable in time* $\omega(1)$



Left-or-right CCA for PKE

mCCA



DecO returns ⊥ if c_i^* is queried

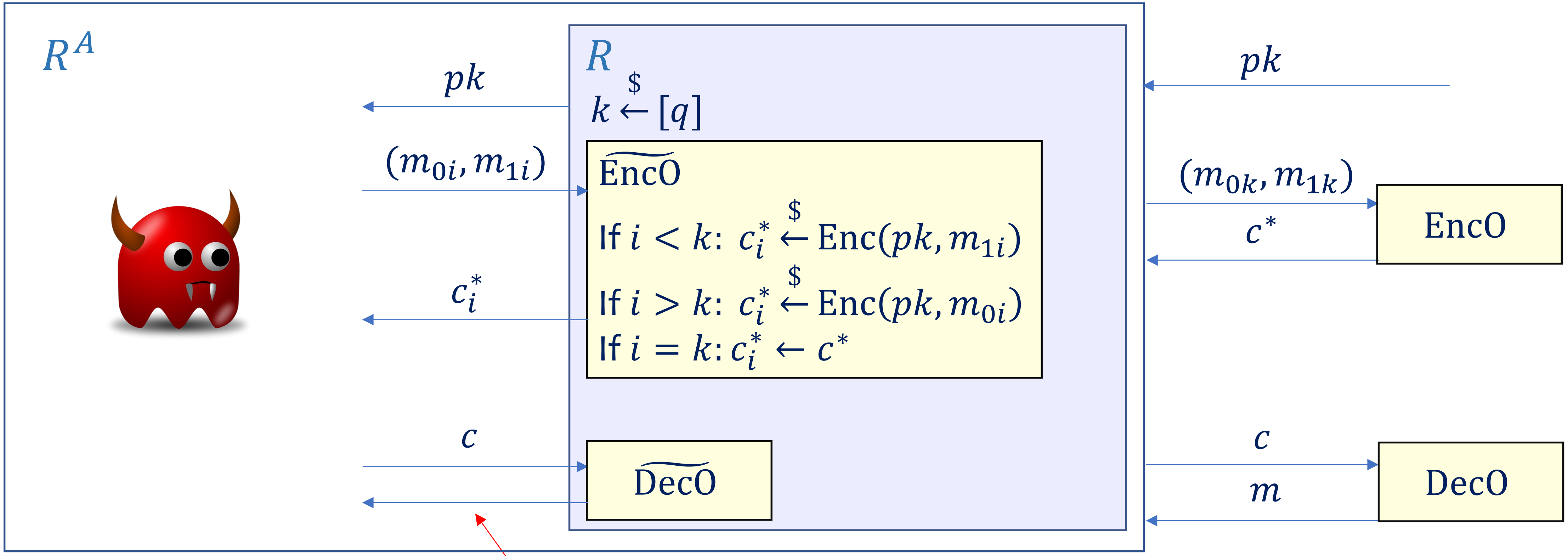
1CCA ⇒ mCCA

not memory-tight

Let's see why …

(erronously claimed memory-tight in [ACFK17])

Let us recall the 1CCA ⇒ mCCA reduction

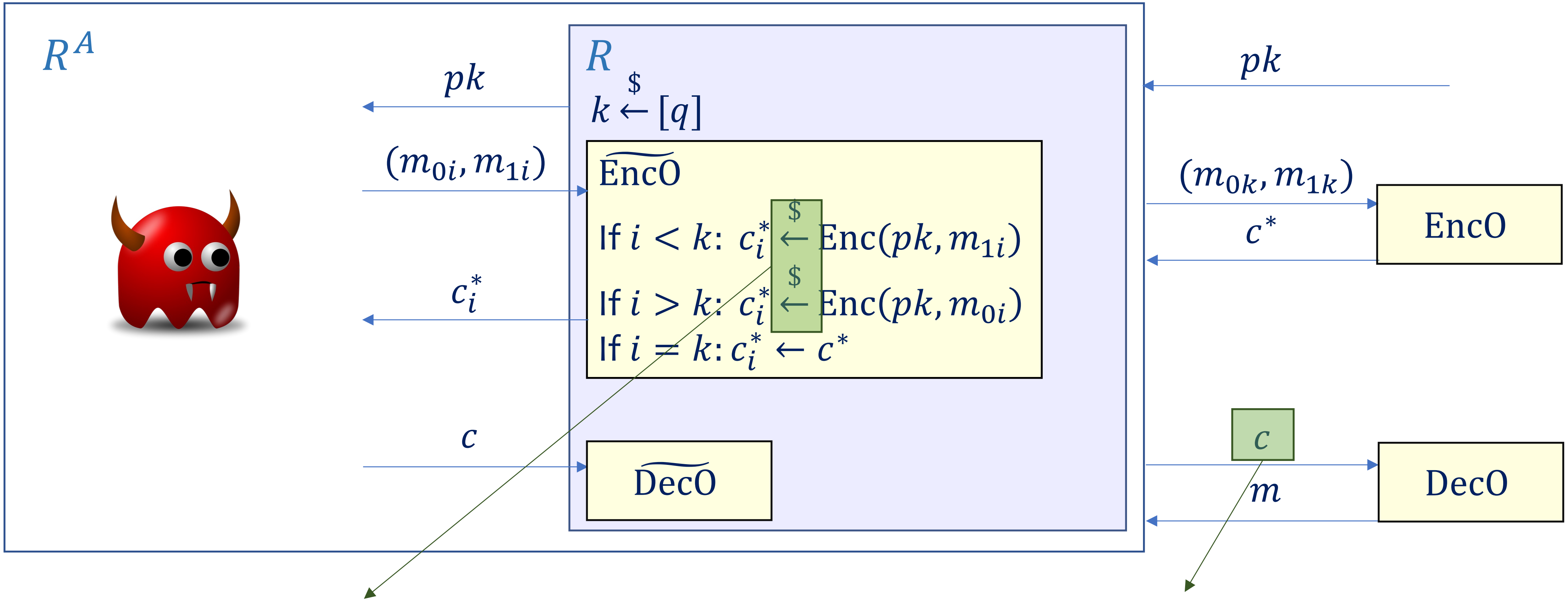


Need to return ⊥ if $c = c_i^*$ for some i, m otherwise

Solution: ... c_i^*'s $\rightarrow$ **not memory-tight**

Inefficient tagging!

Key idea

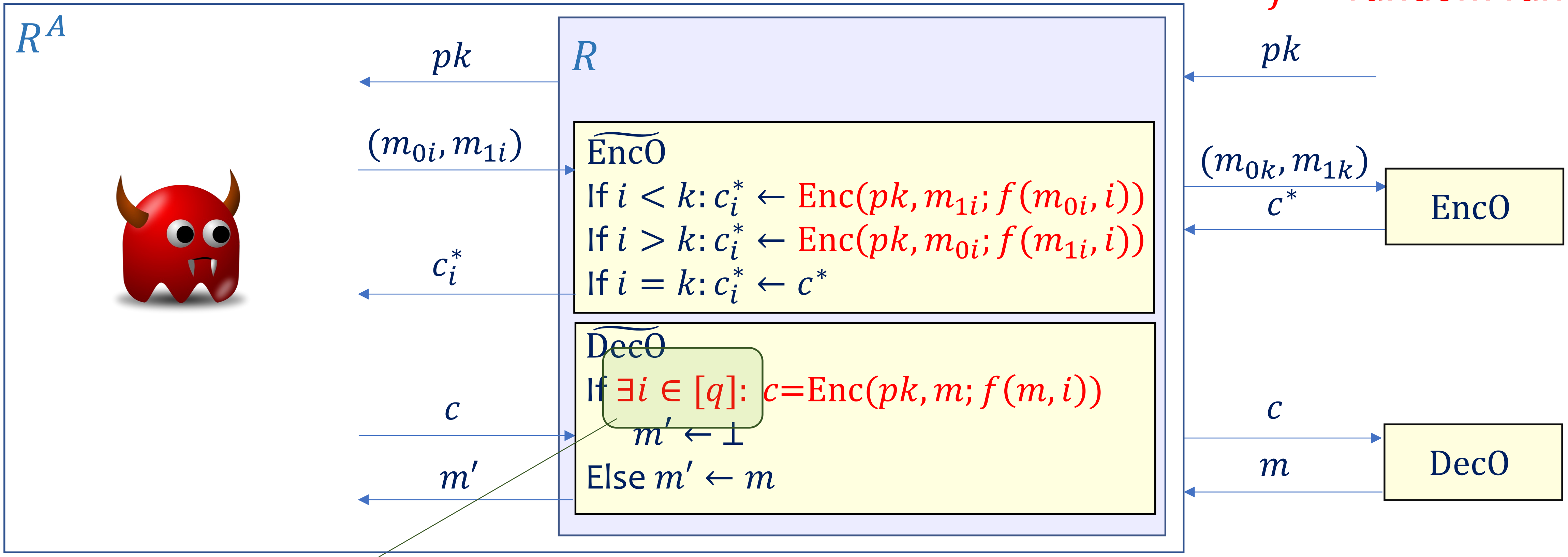


Idea 1: use randomness which is determined by the message and i

Idea 2: To figure out whether c is a challenge ciphertext, re-encrypt m using randomness corresponding m and i for each i

Concretely: inefficient tagging

f = random function



not time-tight

$\exists i$: $c = c_i^* \Rightarrow c = \mathrm{Enc}(pk, m; f(m, i))$

o.w. w.h.p. $c \neq \mathrm{Enc}(pk, m; f(m, i))$

Why is inefficient tagging enough?

It can be better to have memory-tightness over time-tightness for many problems

Lattices, RSA/Factoring, finite field DLP, …

What if I really also want time-tightness, though?

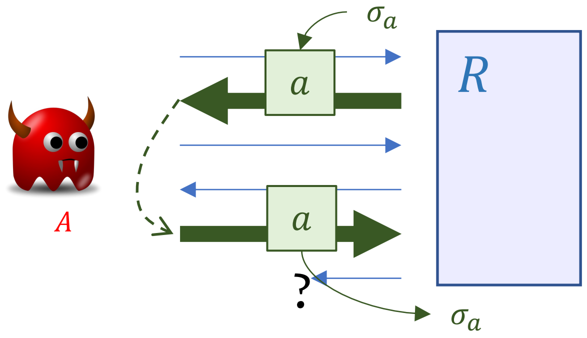
Change the definition!

This talk: three techniques

1. Efficient tagging
2. Inefficient tagging
3. Message encoding

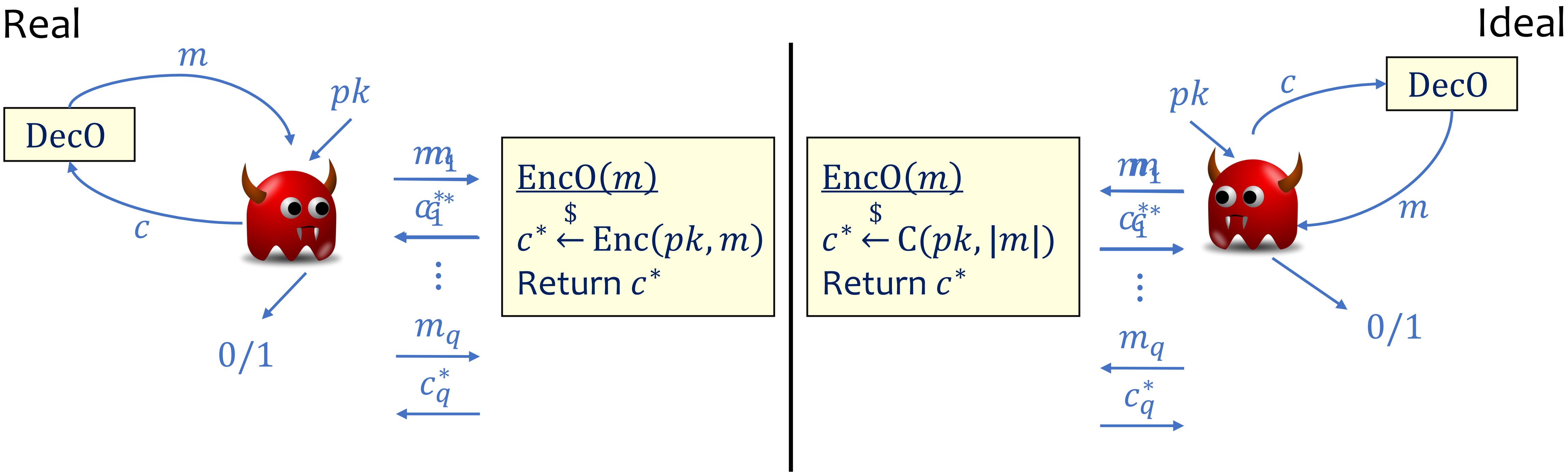
$\sigma_a \in \{0,1\}$, *recoverable in time* $\omega(1)$

Bounded length σ_a, *recoverable in time* $O(1)$



Real-or-random CCA for PKE

m$CCA

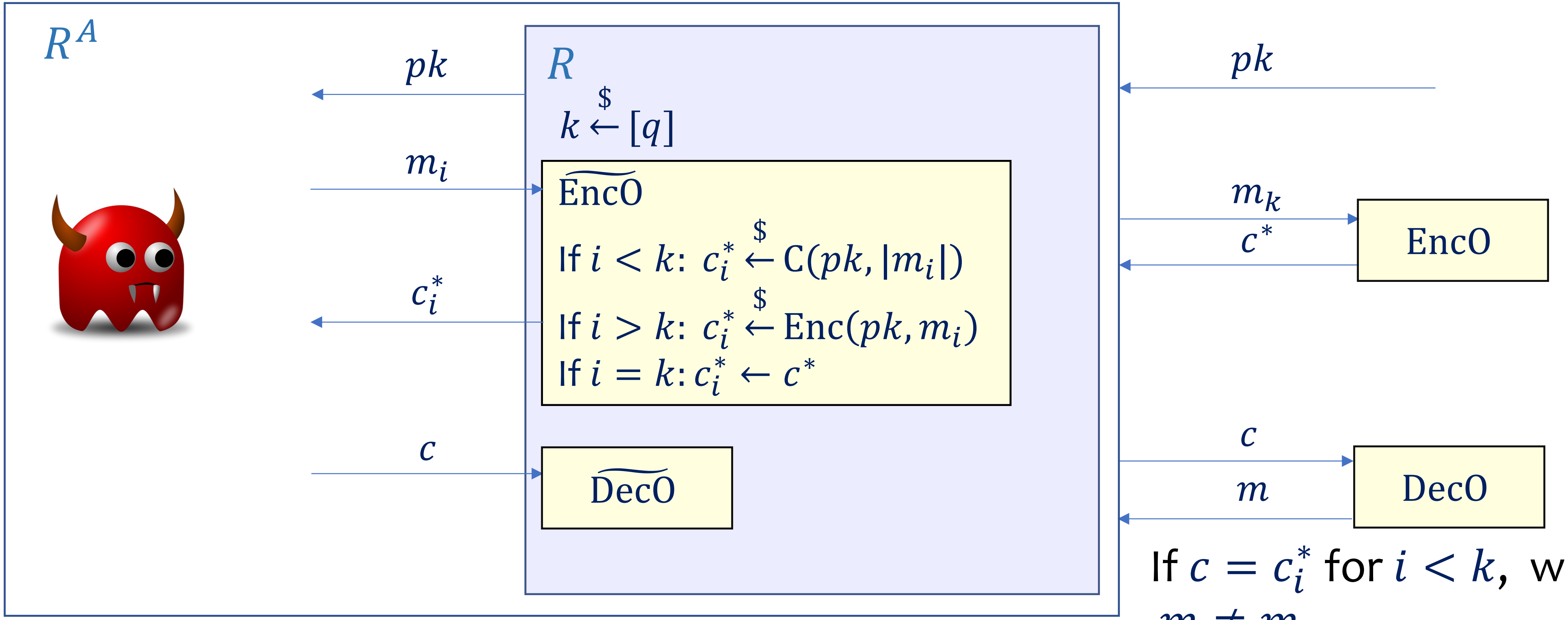


DecO returns m_i iff c_i^* is queried, actual decryption o.w.

1$CCA $\Rightarrow$ m$CCA

not memory-tight

1$CCA ⇒ m$CCA reduction

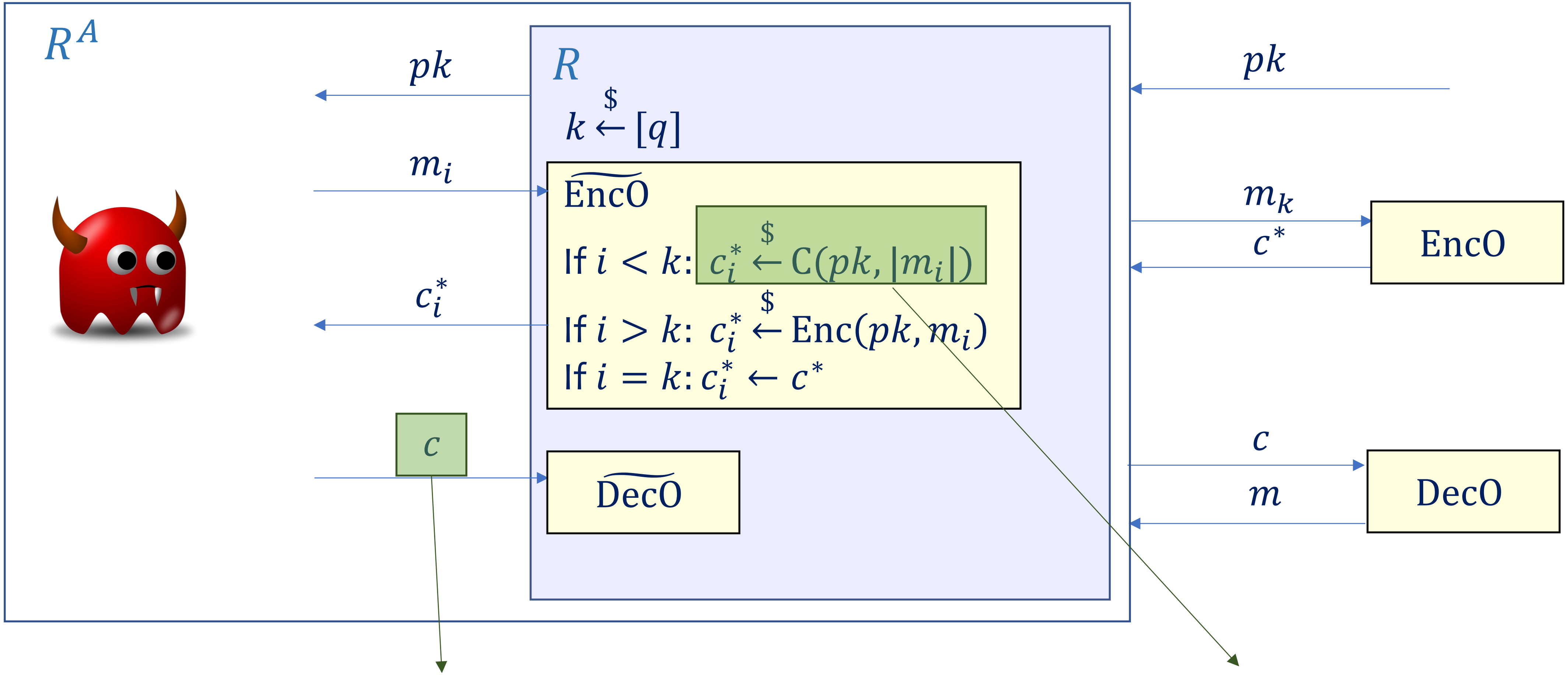


If $c = c_i^*$ for $i < k$, w.h.p $m \neq m_i$

Solution: Reme… $, c_i^*)$ for $i < k$ → **not memory-tight**

Message encoding!

Key idea



Idea 2: decode c – if the decoded answer is of the "right" form, return decoded message, o.w. use DecO

Idea 1: encode m_i into c_i^* for $i < k$

Definitions matter!

Depending on which definition of IND-CCA we use …

the memory-tight reduction for single CCA ⇒ multi-CCA

- may be time-tight
- may not be time-tight

Lesson: Quality of memory-tight reduction strongly related to definitional choices

Other results

- **Memory-tight** AE security for **Encrypt-then-PRF**
  - Bypasses impossibility of [GJT20]
- **Generalize memory-tight mUFCMA** result for RDS
  - Captures signature used in **TLS 1.3**
- **Time, memory, advantage-tight** direct reduction of mUFCMA security of **RSA-PFDH** to RSA

Conclusions

- Ability to give memory-tight reductions strongly couples with definitional choices
- Impossibility results [ACFK17,WMHT18,GT20,GJT20] do not preclude positive results for specific schemes

Open problems

- More new general techniques for memory-tightness beyond [ACFK17,Bhattacharya20,GJT20,DJKL21] and this work
- Understanding the “right” definitional choices in the memory-restricted setting

Paper: https://eprint.iacr.org/2021/1409