RSAConference2020

San Francisco | February 24 – 28 | Moscone Center

HUMAN ELEMENT

SESSION ID: **CRYP-F03**

Universally Composable Accumulators

Foteini Baldimtsi, Ran Canetti, Sophia Yakoubov

Sophia Yakoubov

Postdoc
Aarhus University
@sophiay135

Our Contributions

- **Accumulators** are used in…
  - (Anonymous) Credentials
  - Cryptocurrencies
  - Group and Ring Signatures
- Definition styles:

	Game-Based Definitions	UC Definitions
Statement	Simple	Complex
Proof of secure use in system	Hard	Easy

Our Contributions

- First UC definition for accumulators
- Proof of equivalence of game-based and UC definitions

	Game-Based Definitions	UC Definitions
Statement	Simple	Complex
Proof of secure use in system	Hard	Easy

Our Contributions

- First UC definition for accumulators
- Proof of equivalence of game-based and UC definitions
  - Best of both worlds!
  - All existing constructions are automatically UC-secure

	Game-Based Definitions	=	UC Definitions
Statement	Simple		Complex
Proof of secure use in system	Hard		Easy

Our Contributions

- First UC definition for accumulators
- Proof of equivalence of game-based and UC definitions
- Demonstration of Composition
  - Modular accumulators
  - Anonymous credentials

	Game-Based Definitions	=	UC Definitions
Statement	Simple		Complex
Proof of secure use in system	Hard		Easy

OUTLINE

- What is an accumulator?
- First UC definition for accumulators
- Proof of equivalence of game-based and UC definitions
- Demonstration of Composition

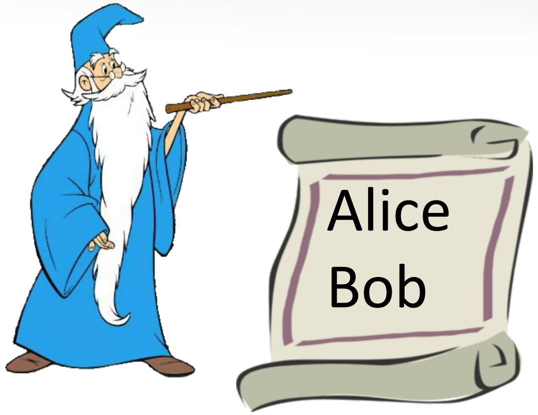
OUTLINE

- What is an accumulator?
- First UC definition for accumulators
- Proof of equivalence of game-based and UC definitions
- Demonstration of Composition

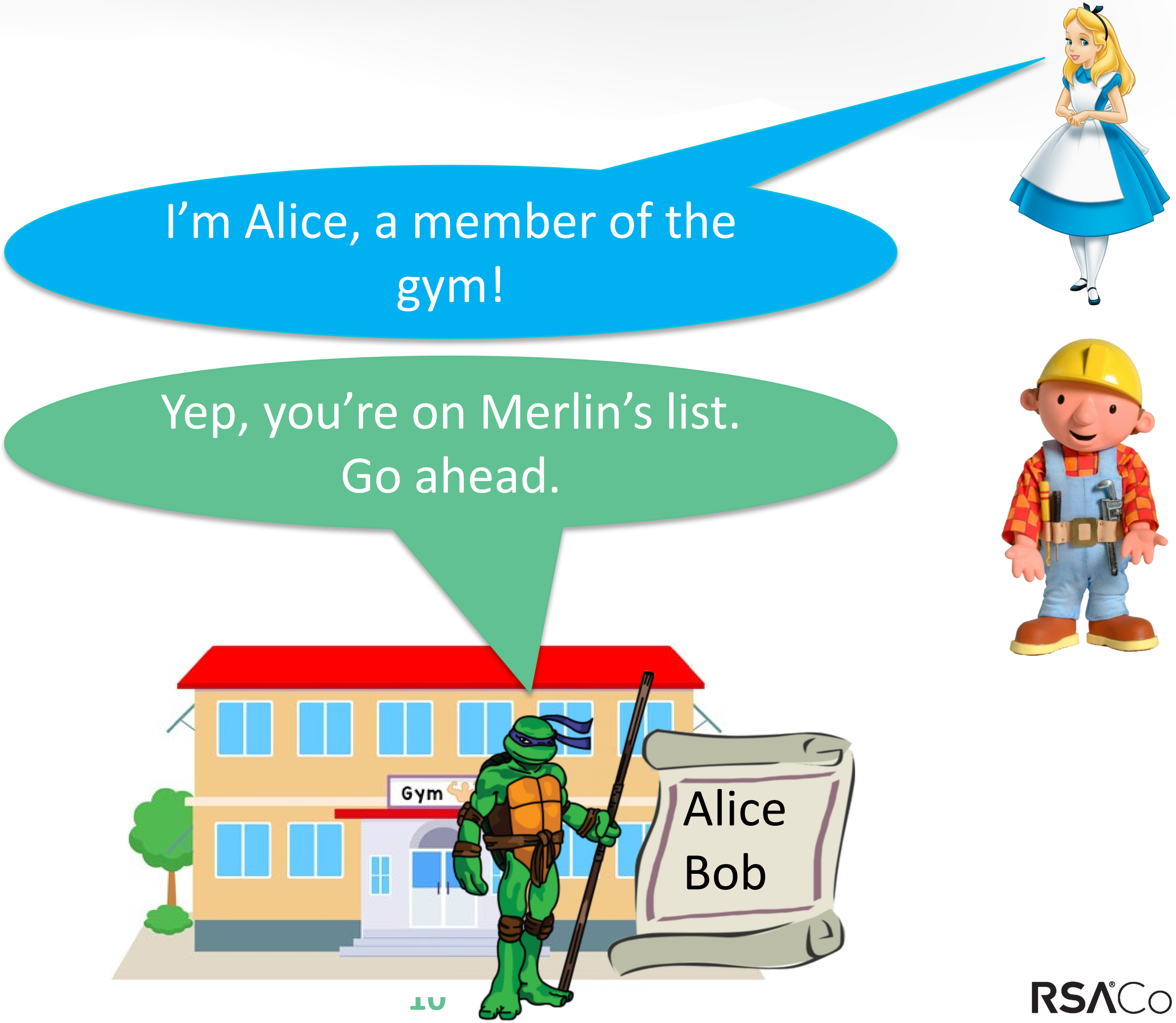
OUTLINE

- What is an accumulator?
- First UC definition for accumulators
- Proof of equivalence of game-based and UC definitions
- Demonstration of Composition

Application: Credentials



Application: Credentials

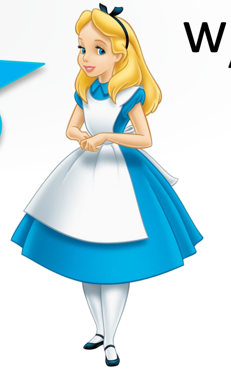


Application: Credentials

Accumulators: a digest of set S

S = {Alice, Bob, …}

Can I have the membership witness for "Alice"?



Witness w_A

Application: Credentials

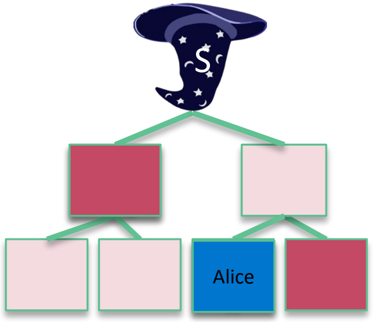
w_A

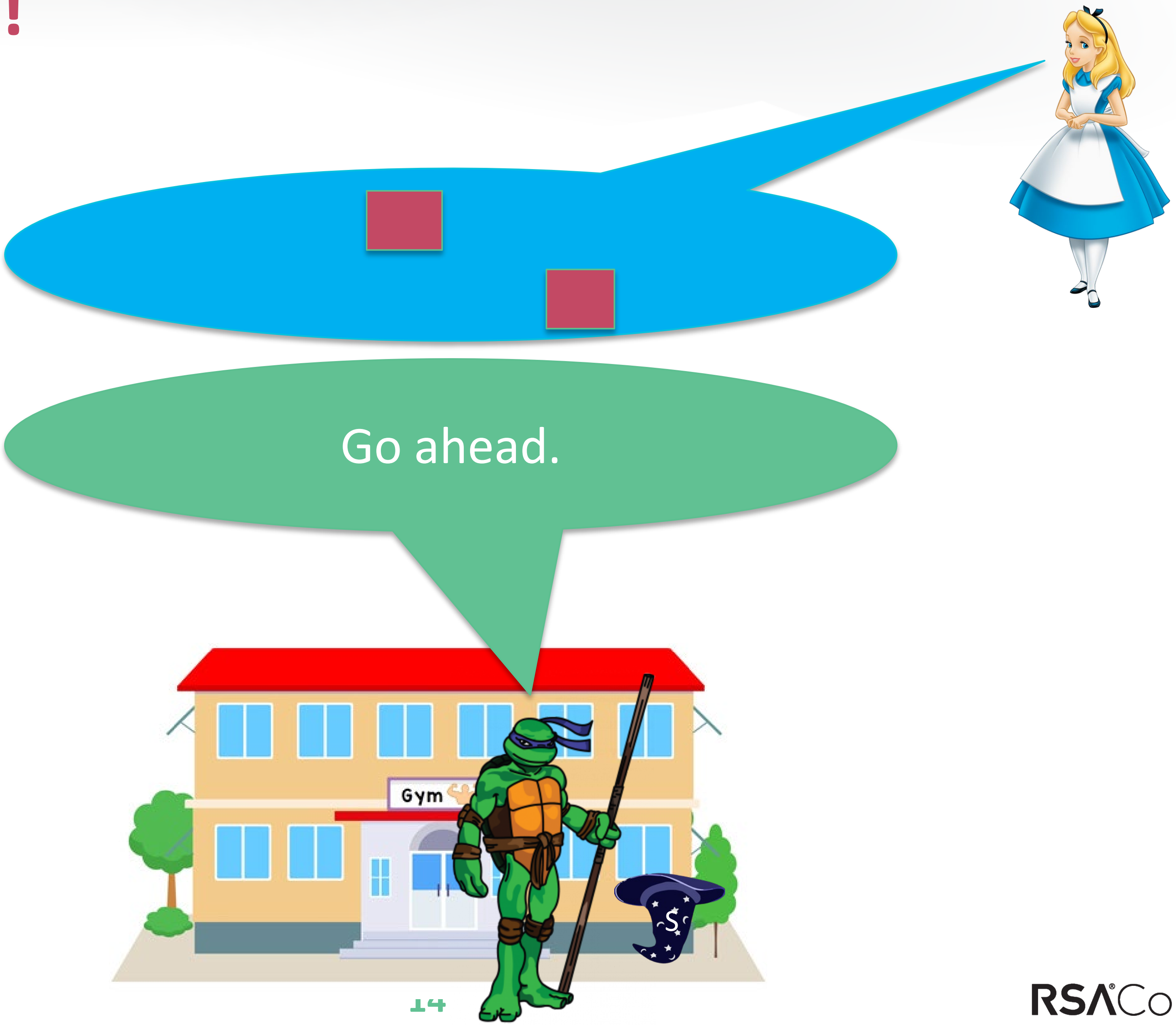
I'm Alice, a member of the gym! w_A

So you are. Go ahead.

Everything is an Accumulator!

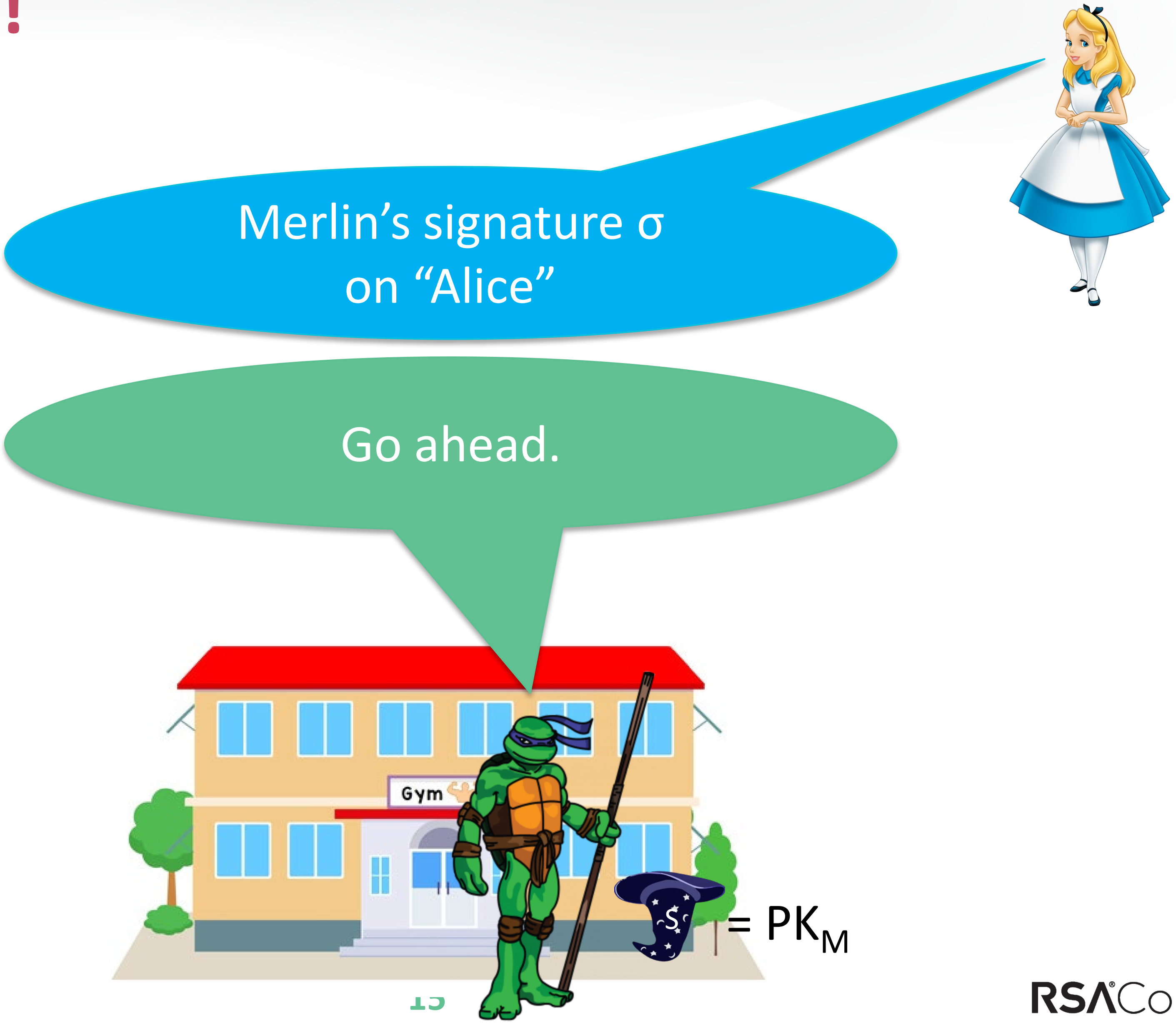
- Merkle Trees





Everything is an Accumulator!

- Merkle Trees
- Digital Signatures

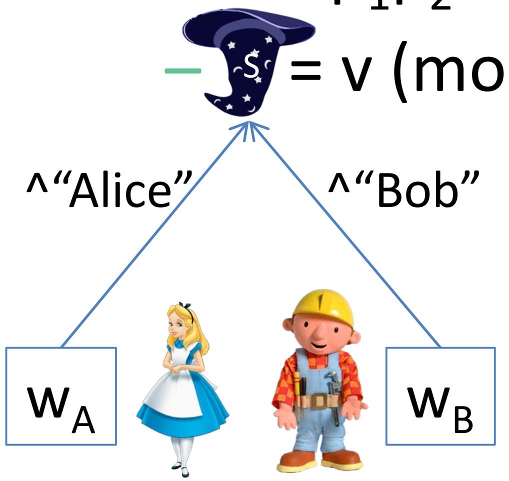


Everything is an Accumulator!

	Strong	Add
Merkle Trees	✓	X
Digital Signatures	X	✓

Everything is an Accumulator!

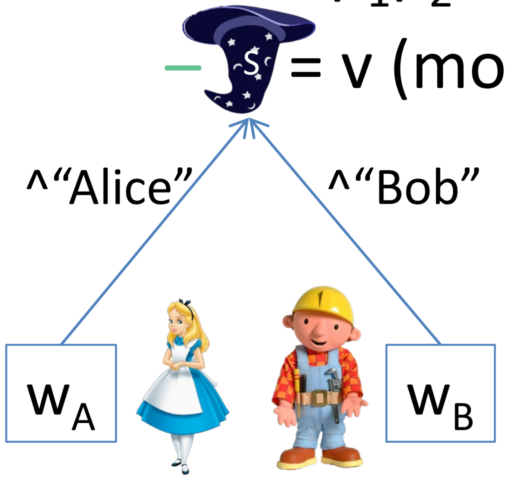
	Strong	Add
Merkle Trees	✓	X
Digital Signatures	X	✓

- RSA Accumulator
  - p_1, p_2 secret primes
  - $n = p_1 p_2$
  -  = v (mod n)

Everything is an Accumulator!

	Strong	Add
Merkle Trees	✓	X
Digital Signatures	X	✓
RSA Accumulator	X	

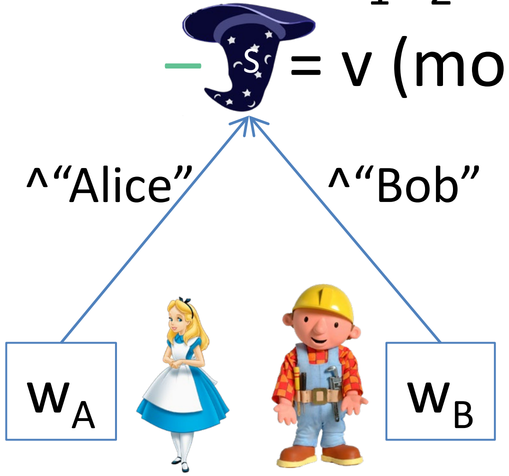
- Merkle Trees
- Digital Signatures
- RSA Accumulator
  - p_1, p_2 secret primes
  - $n = p_1 p_2$
  - $= v \pmod{n}$

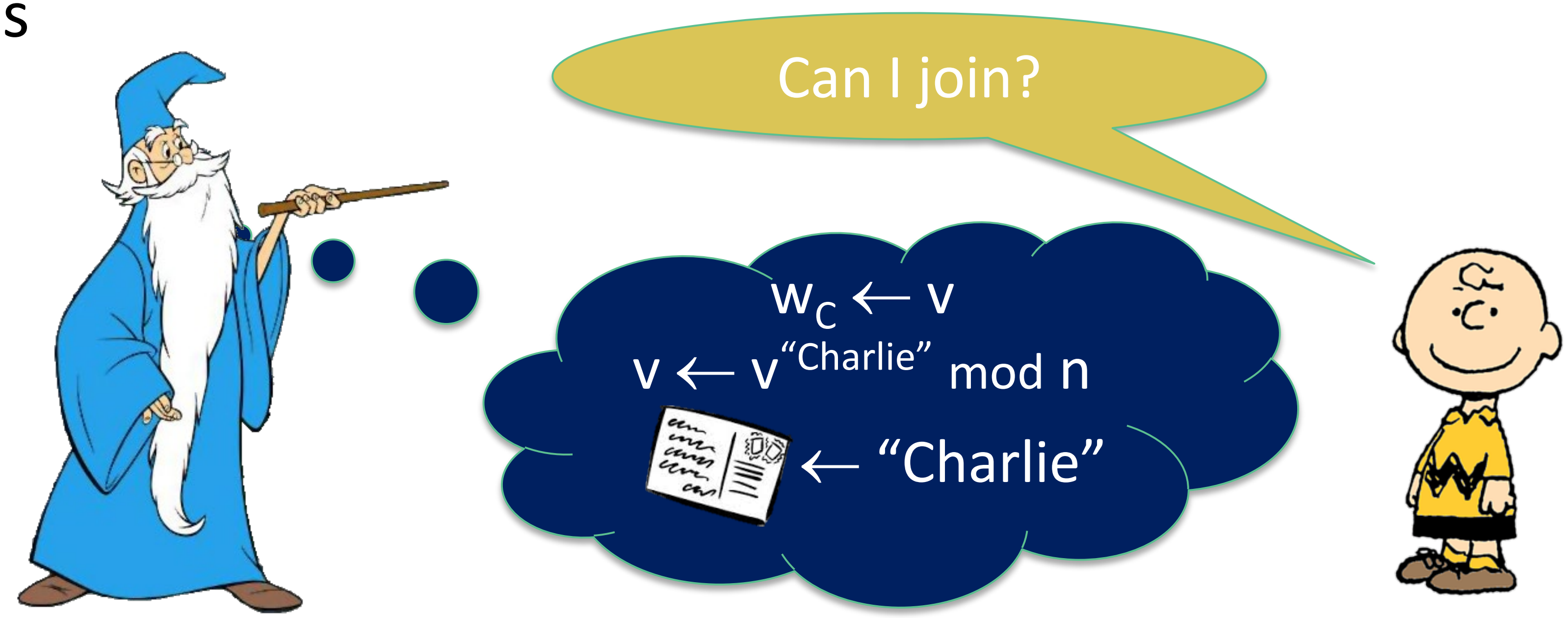


Everything is an Accumulator!

	Strong	Add	Del	Hiding Update Message
Merkle Trees	✓	✗	✗	-
Digital Signatures	✗	✓	✗	✓
RSA Accumulator	✗	✓	✓	✗

- Merkle Trees
- Digital Signatures
- RSA Accumulator
  - p_1, p_2 secret primes
  - $n = p_1 p_2$
  - = v (mod n)





Everything is an Accumulator!

	Strong	Add	Del	Hiding Update Message	Proofs of Non-Membership
Merkle Trees	✓	✗	✗	-	✗
Digital Signatures	✗	✓	✗	✓	✗
RSA Accumulator	✗	✓	✓	✗	✓

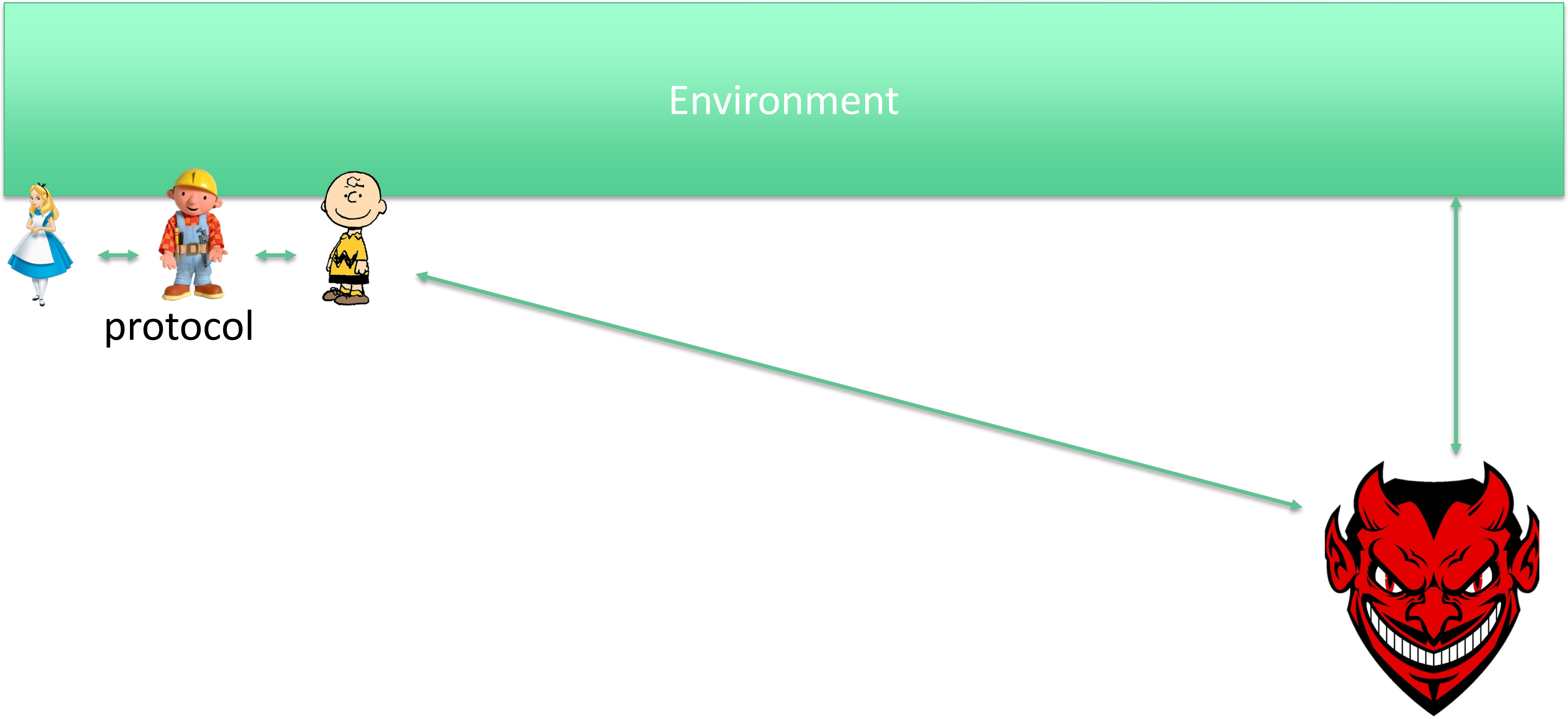
There are many other interesting accumulator properties!

OUTLINE

- What is an accumulator?
- First UC definition for accumulators
- Proof of equivalence of game-based and UC definitions
- Demonstration of Composition

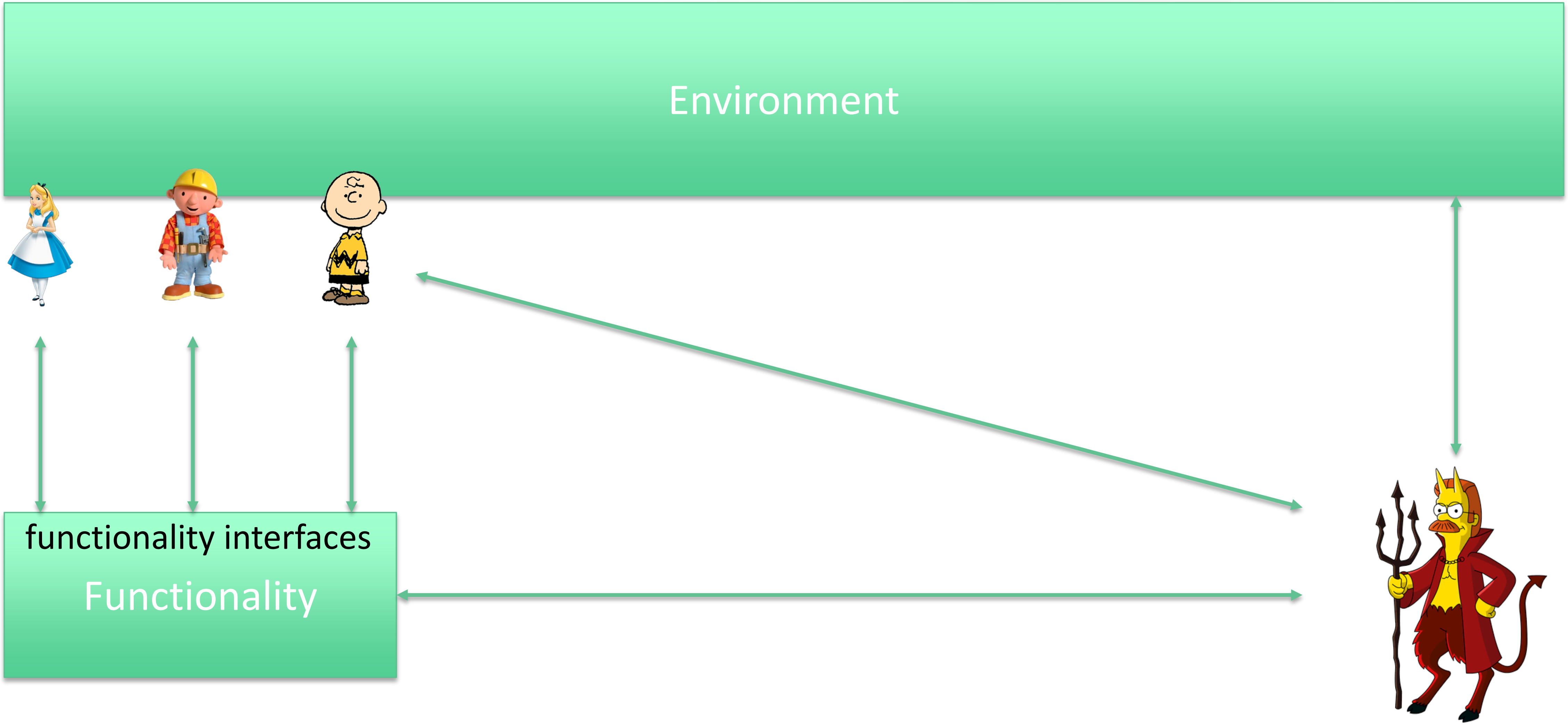
Universal Composability

Real World

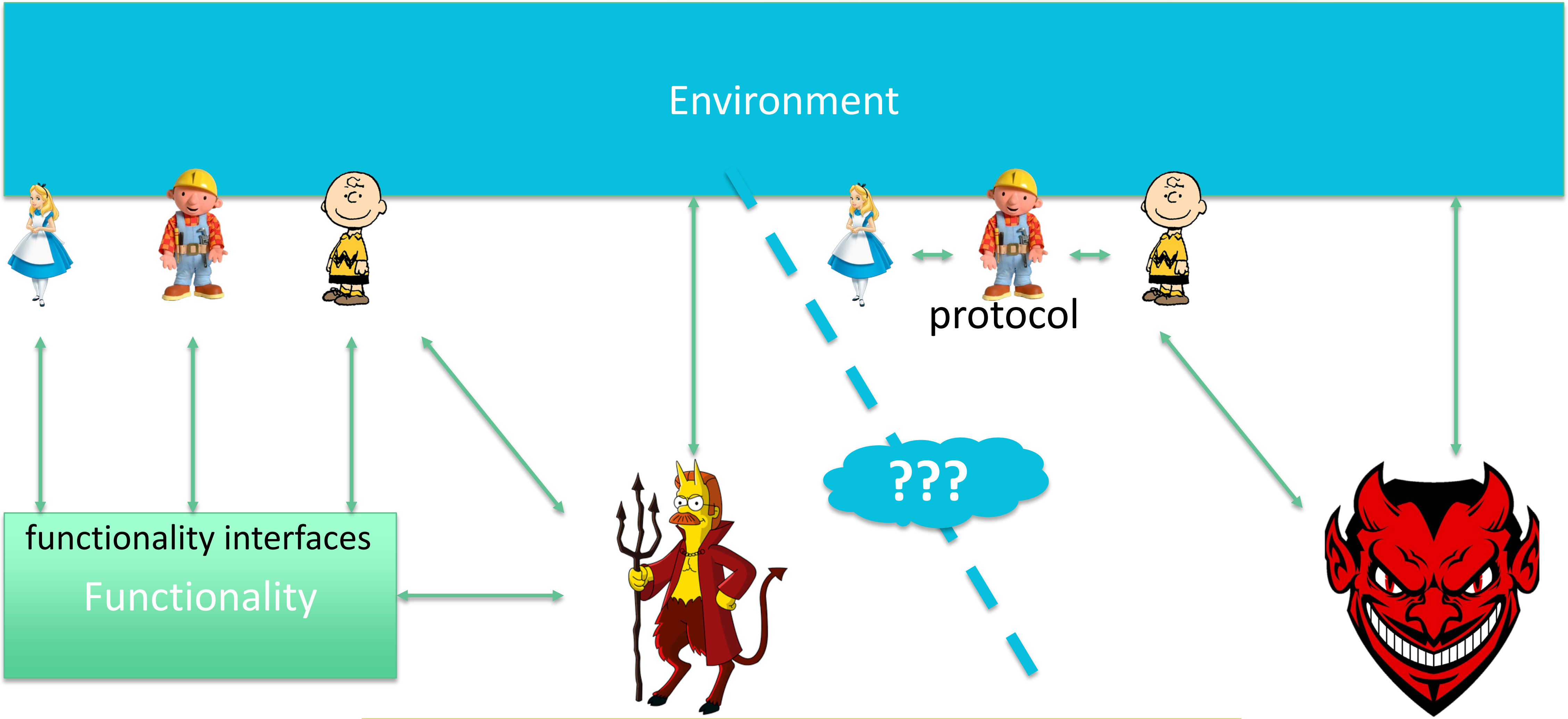


Universal Composability

Ideal World



Universal Composability



New UC definition: describe what the functionality does

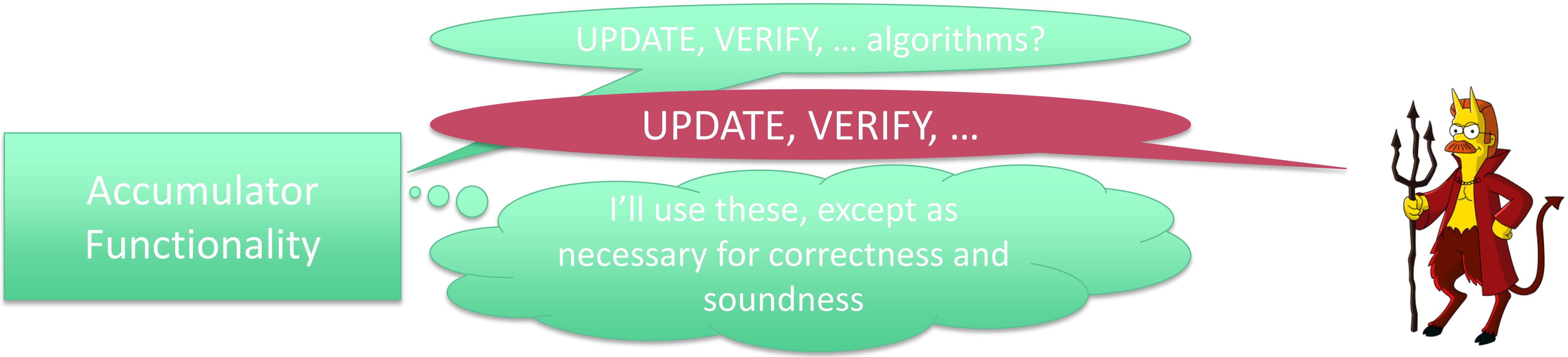
UC Signature Definition

- Requesting **algorithms** from the adversary ensures VERIFY consistency
- This is intuitive: if the functionality never has to substitute algorithms' outputs…
  - The algorithms satisfy correctness and soundness (classical definitions)
  - This is indistinguishable from a real execution (UC security)

UC Accumulator Definition

Same principle as for signatures!

- Requesting **algorithms** from the adversary ensures VERIFY consistency
- This is intuitive: if the functionality never has to substitute algorithms' outputs…
  - The algorithms satisfy correctness and soundness (classical definitions)
  - This is indistinguishable from a real execution (UC security)



1. GEN: Upon getting (GEN, sid, S_0) as first activation from $\mathcal{AM}$ ...
   (a) Initialize an operation counter $t = 0$.
   (b) Initialize an empty list $\mathbf{A}$. This list will be used to keep track of all accumulator states.
   (c) Initialize an empty map $\mathbf{S}$, and set $\mathbf{S}[0] = S_0$. (If S_0 was not provided, use $\emptyset$.) This map will be used to map operation counters to current accumulated sets.
   (d) Send (GEN, sid) to Adversary $\mathcal{A}_{\mathsf{Ideal}}$.
   (e) Get (ALGORITHMS, sid, (Gen, Update, WitCreate, WitUp, VerStatus, VerGen, VerUpdate)) from Adversary $\mathcal{A}_{\mathsf{Ideal}}$. This includes all of the accumulator algorithms; their expected input output behavior is described in Figure 1. All of them should be polynomial-time; we restrict the verification algorithms VerStatus, VerGen, VerUpdate to be deterministic.
   (f) Run $(sk, a_0, m_0, \mathsf{v}) \leftarrow \mathsf{Gen}(1^\lambda, S_0)$.
   (g) Verify that $\mathsf{VerGen}(S_0, a_0, \mathsf{v}) = 1$. If not, output $\perp$ to $\mathcal{AM}$ and halt. (This ensures strength.) Otherwise, continue.
   (h) Store sk, m_0; add a_0 to $\mathbf{A}$.
   (i) Output (ALGORITHMS, sid, S_0, (Gen, Update, WitCreate, WitUp, VerStatus, VerGen, VerUpdate) to $\mathcal{AM}$.
2. UPDATE: Upon getting (UPDATE, sid, Op, x) from $\mathcal{AM}$ ...
   (a) Increment the operation counter: $t = t + 1$.
   (b) Set $\mathbf{S}[t] = \mathbf{S}[t-1]$.
   (c) Run $(a_t, m_t, w_t^x, \mathsf{upmsg}_t, \mathsf{v}_t) \leftarrow \mathsf{Update}(\mathsf{Op}, sk, a_{t-1}, m_{t-1}, x)$.
   (d) If Op = Add:
      i. Verify that $\mathsf{VerStatus}(\mathsf{in}, a, x, w_t) = 1$. If not, output $\perp$ to $\mathcal{AM}$ and halt. (This ensures correctness.) Otherwise, continue.
      ii. If $x \notin \mathbf{S}[t]$, add x to $\mathbf{S}[t]$.
   (e) If Op = Del:
      i. Verify that $\mathsf{VerStatus}(\mathsf{out}, a, x, w_t) = 1$. If not, output $\perp$ to $\mathcal{AM}$ and halt. (This ensures negative correctness.) Otherwise, continue.
      ii. If $x \in \mathbf{S}[t]$, remove x from $\mathbf{S}[t]$.
   (f) Verify that $\mathsf{VerUpdate}(\mathsf{Op}, a_{t-1}, a_t, x, \mathsf{v}_t) = 1$. If not, output $\perp$ to $\mathcal{AM}$ and halt. (This ensures strength.) Otherwise, continue.
   (g) Store m_t, upmsg_t; add a_t to $\mathbf{A}$.
   (h) Output (UPDATE, sid, Op, a_t, x, w_t, upmsg_t) to $\mathcal{AM}$.
3. WITCREATE: Upon getting (WITCREATE, sid, stts, x) from $\mathcal{AM}$ ...
   (a) Run $w \leftarrow \mathsf{WitCreate}(\mathsf{stts}, sk, a_t, m_t, x, (\mathsf{upmsg}_1, \ldots, \mathsf{upmsg}_t))$
   (b) If stts = in:
      If $x \in \mathbf{S}[t]$, verify that $\mathsf{VerStatus}(\mathsf{in}, a_t, x, w) = 1$. If not, output $\perp$ to $\mathcal{AM}$ and halt. (This ensures creation-correctness.) Otherwise, continue.
   (c) If stts = out:
      If $x \notin \mathbf{S}[t]$, verify that $\mathsf{VerStatus}(\mathsf{out}, a_t, x, w) = 1$. If not, output $\perp$ to $\mathcal{AM}$ and halt. (This ensures negative-creation-correctness.) Otherwise, continue.
   (d) Output (WITNESS, sid, stts, x, w) to $\mathcal{AM}$.
4. WITUP: Upon getting (WITUP, sid, stts, a_{old}, a_{new}, x, w_{old}, $(\mathsf{upmsg}_{old+1}, \ldots, \mathsf{upmsg}_{new})$) from any party $\mathcal{H}$ ...
   (a) Run $w_{new} \leftarrow \mathsf{WitUp}(\mathsf{stts}, x, w_{old}, (\mathsf{upmsg}_{old+1}, \ldots, \mathsf{upmsg}_{new}))$
   (b) If $a_{old} \in \mathbf{A}$, $a_{new} \in \mathbf{A}$ and $old < new$:
      i. If stts = in, $\mathsf{VerStatus}(\mathsf{in}, a_{old}, x, w_{old}) = 1$, $x \in \mathbf{S}[t]$ for $t \in [old, \ldots, new]$, $\mathsf{upmsg}_{old+1}, \ldots, \mathsf{upmsg}_{new}$ match the stored values and $\mathsf{VerStatus}(\mathsf{in}, a_{new}, x, w_{new}) = 0$, output $\perp$ to $\mathcal{P}$ and halt. (This ensures correctness.) Otherwise, continue.
      ii. If stts = out, $\mathsf{VerStatus}(\mathsf{out}, a_{old}, x, w_{old}) = 1$, $x \notin \mathbf{S}[t]$ for $t \in [old, \ldots, new]$, $\mathsf{upmsg}_{old+1}, \ldots, \mathsf{upmsg}_{new}$ match the stored values and $\mathsf{VerStatus}(\mathsf{out}, a_{new}, x, w_{new}) = 0$, output $\perp$ to $\mathcal{P}$ and halt. (This ensures negative correctness.) Otherwise, continue.
   (c) Output (UPDATEDWITNESS, sid, stts, a_{old}, a_{new}, x, w_{old}, $(\mathsf{upmsg}_{old+1}, \ldots, \mathsf{upmsg}_{new})$, w_{new}) to $\mathcal{H}$.

Fig. 4. Ideal Functionality $\mathcal{F}_{\mathsf{ACC}}$ for Accumulators With Explicit Verification Algorithm

See the paper for the full functionality ☺

1. VERSTATUS: Upon getting (VERSTATUS, sid, stts, a, VerStatus$'$, x, w) from any party $\mathcal{P}$ ...
   (a) If VerStatus$'$ = VerStatus and there exists a t such that $a = a_t \in \mathbf{A}$:
      i. Let t be the largest such number.
      ii. If stts = in:
         A. If $\mathcal{AM}$ not corrupted, $x \notin \mathbf{S}[t]$ and $\mathsf{VerStatus}(\mathsf{in}, a_t, x, w) = 1$, output $\perp$ to $\mathcal{P}$ and halt. (This ensures collision-freeness.) Otherwise, continue.
         B. Set $\phi = \mathsf{VerStatus}(\mathsf{in}, a_t, x, w)$.
      iii. If stts = out:
         A. If $\mathcal{AM}$ not corrupted, $x \in \mathbf{S}[t]$ and $\mathsf{VerStatus}(\mathsf{out}, a_t, x, w) = 1$, output $\perp$ to $\mathcal{P}$ and halt. (This ensures negative collision-freeness.) Otherwise, continue.
         B. Set $\phi = \mathsf{VerStatus}(\mathsf{out}, a_t, x, w)$.
   (b) Otherwise, set $\phi = \mathsf{VerStatus}'(\mathsf{stts}, a, x, w)$.
   (c) Output (VERIFIED, sid, stts, a, VerStatus$'$, x, w, ϕ) to $\mathcal{P}$.
2. VERGEN: Upon getting (VERGEN, sid, S, a, v, VerGen$'$) from any party $\mathcal{P}$ ...
   (a) Set $\phi = \mathsf{VerGen}'(S, a, \mathsf{v})$.
   (b) Output (VERIFIED, sid, S, a, v, VerGen$'$, ϕ) to $\mathcal{P}$.
3. VERUPDATE: Upon getting (VERUPDATE, sid, Op, a, a', x, v_t, VerUpdate$'$) from any party $\mathcal{P}$ ...
   (a) Set $\phi = \mathsf{VerUpdate}'(\mathsf{Op}, a, a', x, \mathsf{v}_t)$.
   (b) Output (VERIFIED, sid, Op, a, a', x, v_t, VerUpdate$'$, ϕ) to $\mathcal{P}$.

Fig. 5. Ideal Functionality $\mathcal{F}_{\mathsf{ACC}}$ Interfaces for Third Parties

OUTLINE

- What is an accumulator?
- First UC definition for accumulators
- Proof of equivalence of game-based and UC definitions
- Demonstration of Composition
  - Modular accumulators
  - Anonymous credentials

OUTLINE

- What is an accumulator?
- First UC definition for accumulators
- Proof of equivalence of game-based and UC definitions
- Demonstration of Composition
  - Modular accumulators
  - Anonymous credentials

Everything is an Accumulator!

	Strong	Add	Del	Hiding Update Message	Proofs of Non-Membership
• Merkle Trees	✓	✗	✗	-	✗
• Digital Signatures	✗	✓	✗	✓	✗
• RSA Accumulator	✗	✓	✓	✗	✓

Everything is an Accumulator!

	Strong	Add	Del	Hiding Update Message	Proofs of Non-Membership
Merkle Trees	✓	✗	✗	-	✗
Digital Signatures	✗	✓	✗	✓	✗
RSA Accumulator	✗	✓	✓	✗	✓
WANT		✓	✓	✓	

Composition: Braavos

Adding "Alice":
Pick random R

("Alice", R) R

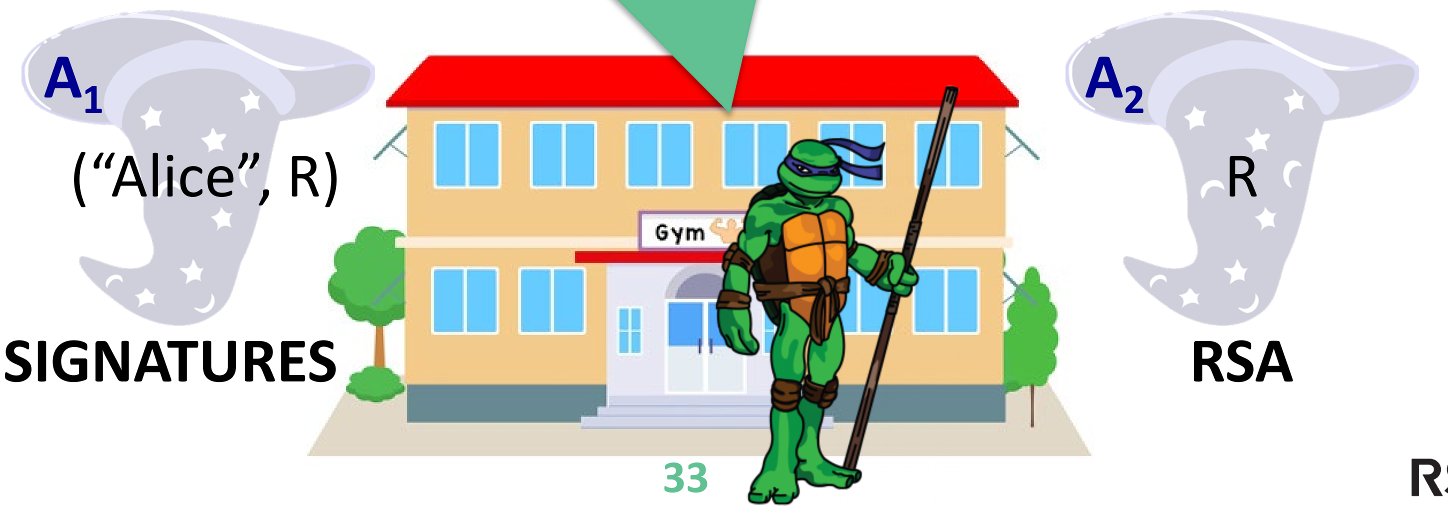
SIGNATURES

RSA

Composition: Braavos

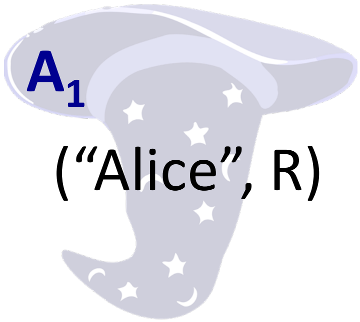
R, witness w_1 that (R, "Alice") in A_1, witness w_2 that R in A_2

Go ahead.

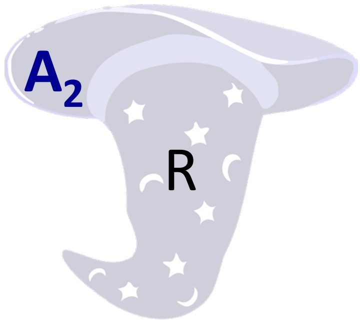


Composition: Braavos

Deleting “Alice”:



SIGNATURES



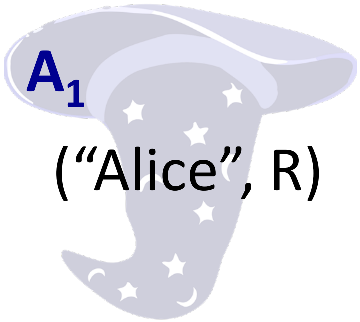
RSA

Composition: Braavos

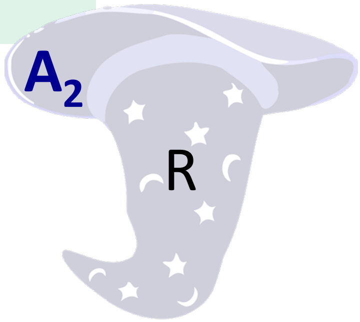
	Strong	Add	Del	Hiding Update Message	Proofs of Non-Membership
• Merkle Trees	✓	✗	✗	-	✗
• Digital Signatures	✗	✓	✗	✓	✗
• RSA Accumulator	✗	✓	✓	✗	✓

We can achieve better efficiency by using CL-RSA-B instead

• Braavos		✓	✓	✓	



SIGNATURES



RSA

OUTLINE

- What is an accumulator?
- First UC definition for accumulators
- Proof of equivalence of game-based and UC definitions
- Demonstration of Composition
  - Modular accumulators
  - Anonymous credentials

Composition: Anonymous Credentials

- We can build revocable anonymous credentials by combining…
  - An hiding update message (HUM) accumulator, e.g. Braavos
  - Zero knowledge proofs

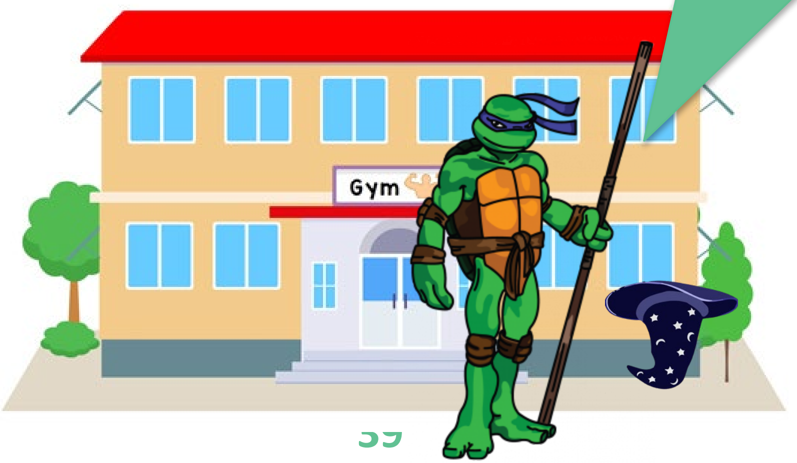
Composition: Anonymous Credentials

Zero knowledge proof of knowledge of sk_A, pk_A, w such that…

- sk_A matches pk_A
- pk_A is in the accumulator

sk_A, pk_A

Go ahead.



Hiding update messages ensure that no-one learns who gets added or deleted!

OUTLINE

- What is an accumulator?
- First UC definition for accumulators
- Proof of equivalence of game-based and UC definitions
- Demonstration of Composition