DATA PRIVACY AND SECURITY

Prof. Daniele Venturi

Master's Degree in Data Science
Sapienza University of Rome

CIS SAPIENZA

RESEARCH CENTER FOR CYBER INTELLIGENCE
AND INFORMATION SECURITY

CHAPTER 7: Alternative Currencies

Drawbacks of Bitcoin

- PoW perspective
  - High **energy consumption**
  - Advantage for people with **dedicated hardware**
- Transactions perspective
  - Scripts are **not Turing complete**
  - Lack of real **anonymity**

Natural Questions

- PoW without mining in hardware?
- Energy-efficient PoW?
- PoW doing something **useful**?
- PoW without mining pools?
- Cryptocurrency with **real anonymity**?
- Cryptocurrency with **Turing-complete** scripts?
- Other uses of blockchain technologies?

Ethereum

How to Order a Murder?

Carol

Bob

I want Alice murdered

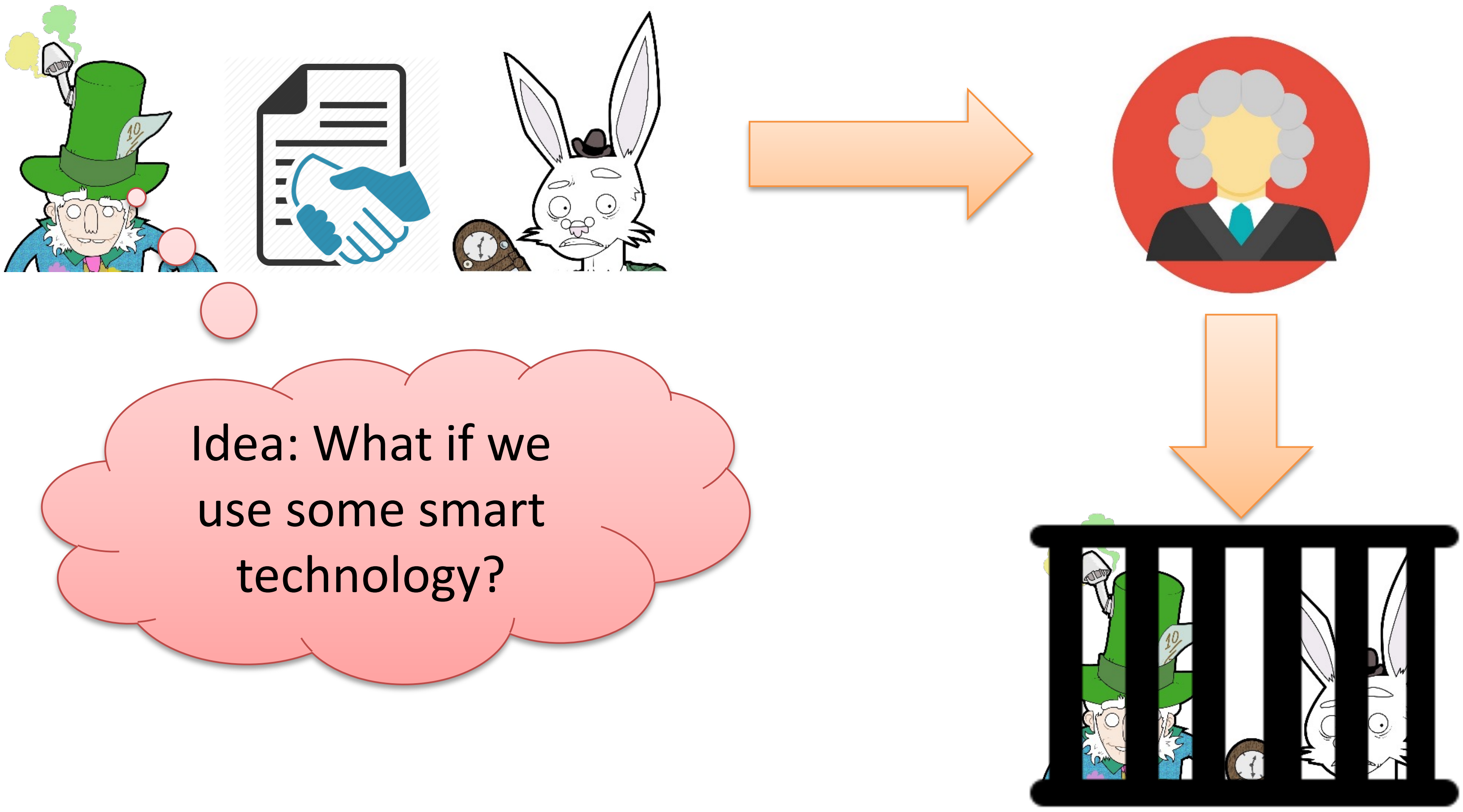
I can do it for 1000000$

So do it, and I will pay you

No, pay first…

No, kill first…

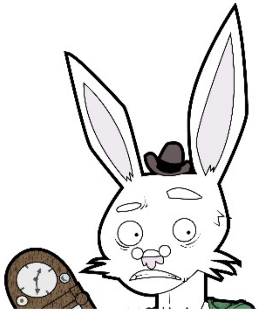
A Bad Solution



Idea: What if we use some smart technology?

Murder Contract

1000 BTC if Bob **provides a proof** that Alice is killed within the next hour



E.g., a signed article from some press agency or an authenticated data feed

Maybe Bob just gets lucky. So add more details, like "using a .44 Magnum Remington gun."

Two Technical Problems

- Such conditions are **impossible** to express using Bitcoin syntax
- A separate contract is needed for every potential hitman
- Solution: Use Ethereum
  - A currency designed for doing **smart contracts**
  - Contracts can be **posted on the blockchain** and give money to anyone who provides a solution
  - Allows to create **arbitrarily complicated contracts**

Promises of Ethereum

- The world computer
- Build **decentralized applications** (DAPPs)
- Trustless & secure smart contracts

Problems with Bitcoin

• Too slow	40 blocks/ 10 minutes
• Not very usable scripting language	Turing complete language
• Only supports transactions	Accounts & Contracts
• Flexible fees	Computations are payed for in gas
• Difficulty adjusted every 1000 blocks	Adjusted in every block
• Mining centralized and uninteresting	POW is memory hard

Ethereum: Some History

January 2014
Vitalik Buterin announces Ethereum

July 2014
Ethereum Presale (18 Mio $)

July 2015
First Release: Frontier

March 2016
Second Release: Homestead

April 2016
Launch of the DAO

May 2016
DAO collects 160 Mio $

June 2016
DAO gets “hacked”

July 2016
First Hard Fork

Ethereum Virtual Machine (EVM)

- Contracts are written in higher-level languages
  - Solidity(Javascipt)
  - Serpent (Python)
  - LLL (Lisp)
- EVM: Low-level, stack-based bytecode language
  - Run by every Ethereum node
  - Contracts need to be compiled before deployment
  - **Turing complete**

Gas

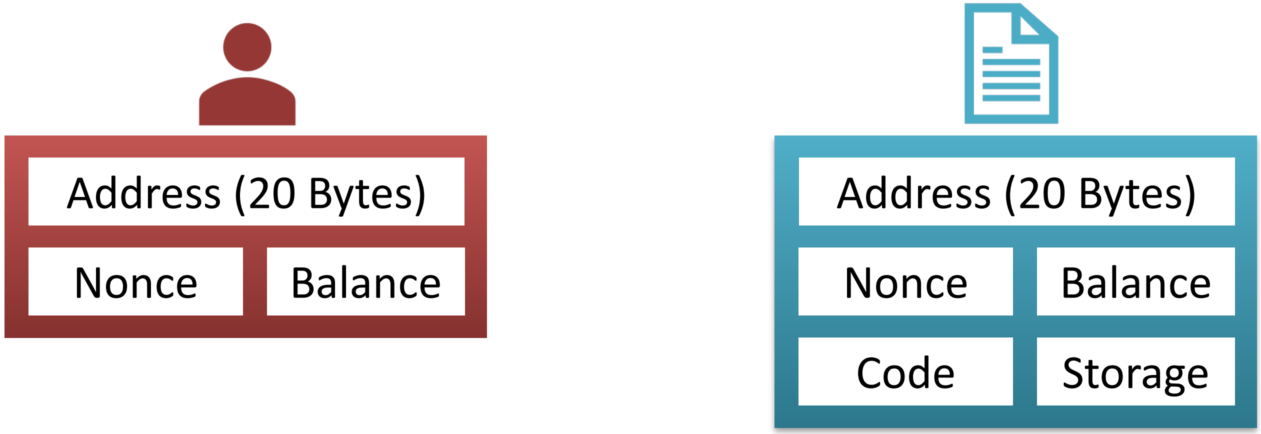
- Users/contracts can run **arbitrary EVM code**
- Every EVM operation has a certain **cost (gas)**

OP Code	Gas	Description
0x01 ADD	3	Add two values
0x06 MOD	5	Modulo Operation
0x20 SHA3	30	Calculate Keccak-256 of a value
0xf0 CREATE	40	Create a new EOA/ contract address

  - If execution requires more gas than the user sent, all changes are reverted but fee goes to the miner
- The gas price is determined by **free market**

Accounts

- Basic building block of the Ethereum blockchain
- An account can either be **externally owned (EOA)** or a **contract account**



Address (20 Bytes)

Nonce | Balance

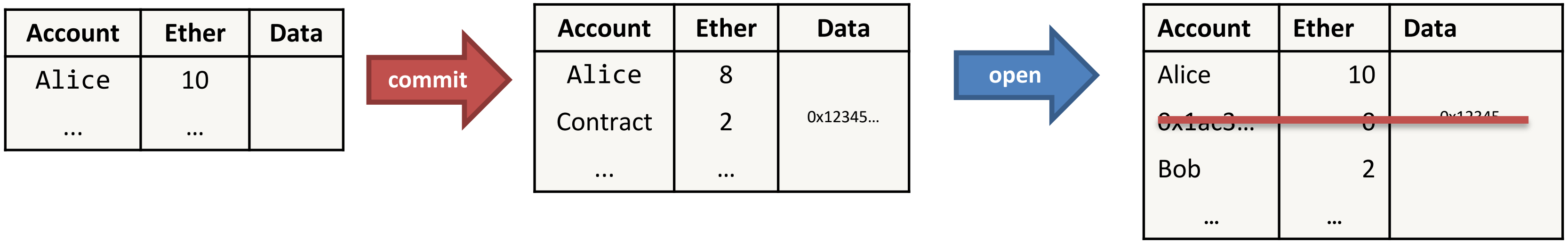
Address (20 Bytes)

Nonce | Balance

Code | Storage

State

- Additionally to the blockchain Ethereum has a concept of **state**



Account	Ether	Data
Alice	10	
…	…	

commit

Account	Ether	Data
Alice	8	
Contract	2	0x12345…
…	…	

open

Account	Ether	Data
Alice	10	
~~0x1ac3…~~	~~0~~	~~0x12345~~
Bob	2	
…	…	

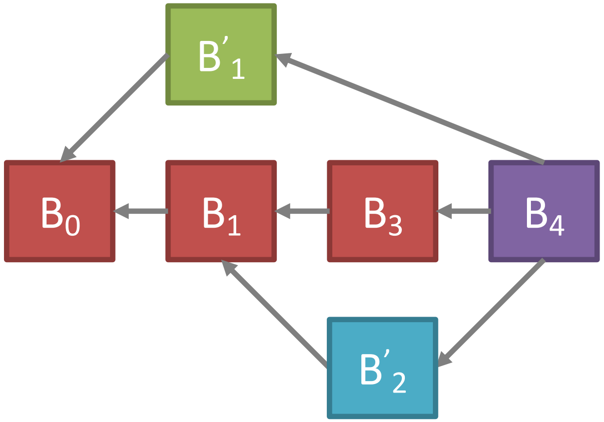
- State can be computed from the blockchain
- Transactions **change the state**

Ethereum Blockchain

- Block creation in Ethereum is approx. 15 sec
  - Problem: **Orphan blocks**
- An orphan, or stale block:
  - Happens if 2 blocks are found at the same time
  - In Bitcoin: Only one block is accepted into the blockchain
  - In Ethereum: Orphans can be included in the blockchain as **uncles**
- Ethereum uses a modification of the **GHOST protocol**

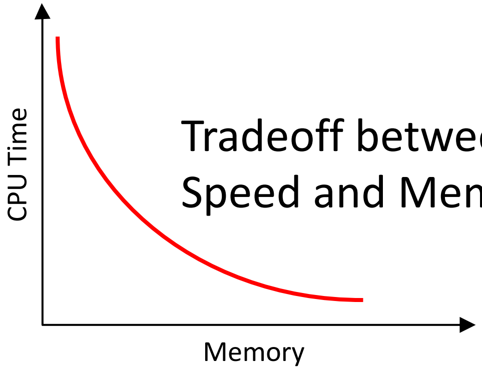
GHOST Protocol

- Goal: Neutralize network lag/centralization
  - A miner gets 12.5% of block reward for every orphan
  - Uncles cannot be older than **7 blocks**
  - Max. **2 uncles** allowed **per block**



Ethash

- Ethereum's PoW Algorithm (Ethash) is believed to be **memory hard**
- Generate a Directed Acylic Graph every 30000 blocks (approx. 5.2 days)
  - – Needs to be precomputed
  - – Computing PoW **requires lookups** in the DAG
  - – Not needed for verification



Tradeoff between CPU Speed and Memory

Comparison with Bitcoin

- Language
  - Script vs **EVM**
- Data
  - Blockchain vs **blockchain + state**
  - Unspent transactions vs **accounts**
- Unit
  - Bitcoin vs **Ether**
  - Transaction fees vs **gas**

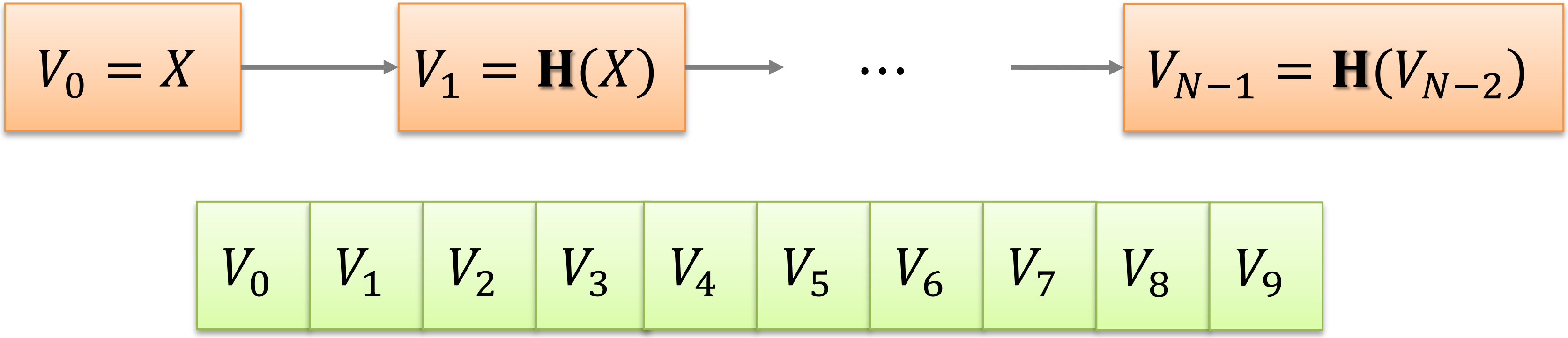
Litecoin

Litecoin

- Released in October 2011 by Charles Lee
- Replaces SHA256 with **scrypt** hash function
  - C. Percival. "Strong key derivation in sequential memory-hard functions." 2009
- Main idea: Make a function whose computation requires a **lot of memory**
  - So it's hard to implement in hardware
  - Proposed to counter **offline password guessing**
  - Market Cap ≈ 2 billion EUR (1 LTC ≈ 30 EUR)

The scrypt function

- Initialization phase:



$V_0 = X \rightarrow V_1 = \mathbf{H}(X) \rightarrow \cdots \rightarrow V_{N-1} = \mathbf{H}(V_{N-2})$

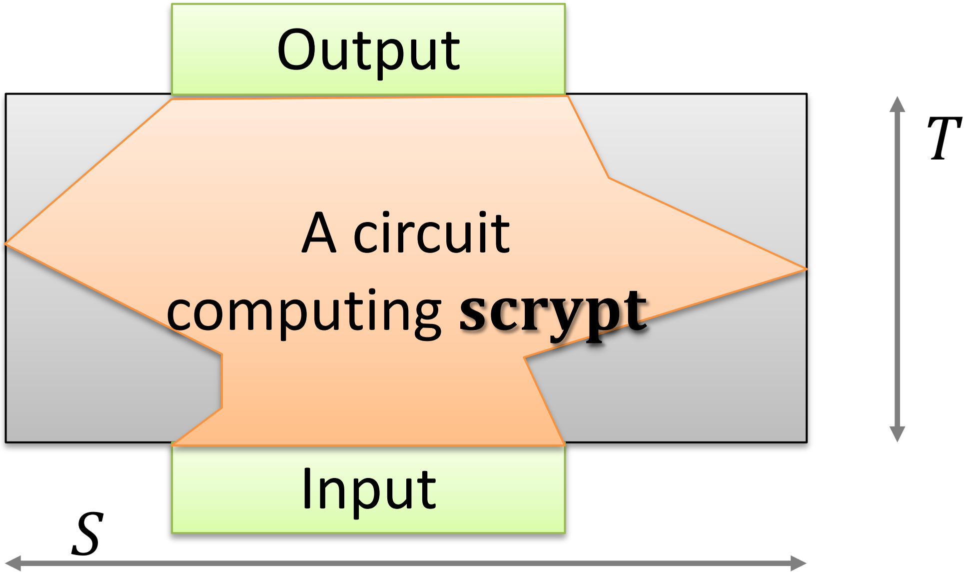
V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9

- Second phase:

```
$Y = \mathbf{H}(V_{N-1})$
For $i = 0, \ldots, N-1$
    $j := Y \bmod N$
    $Y := \mathbf{H}(Y \oplus V_j)$
Output $Y$
```

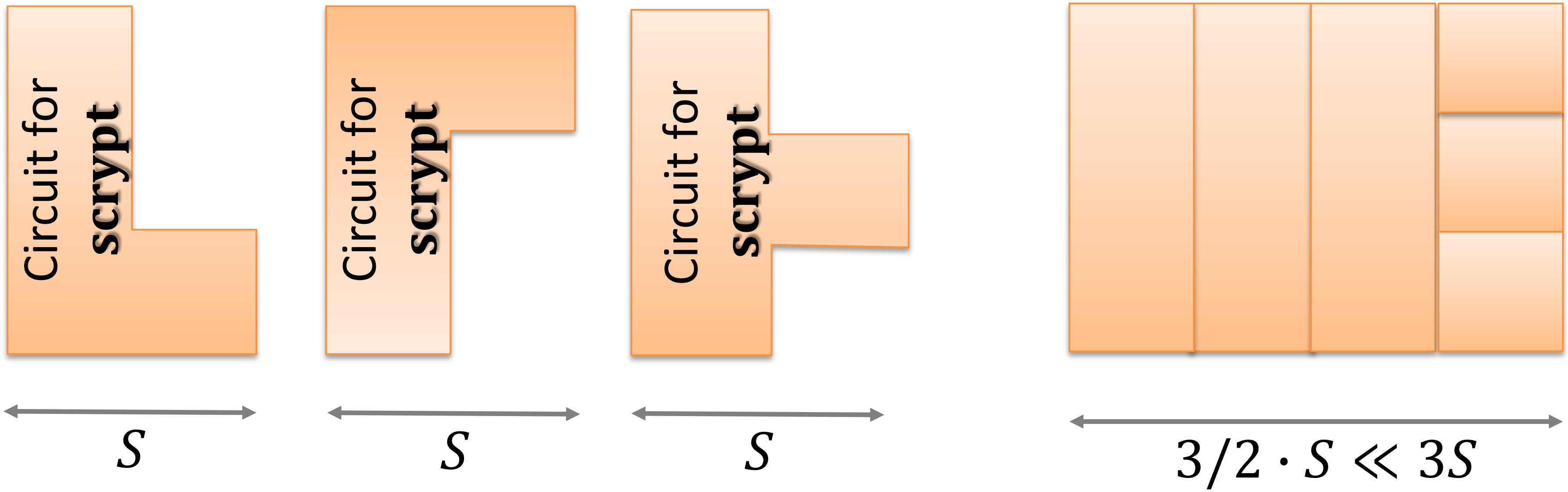
The Result of Percival

- It can be **computed** in time $O(N)$
- To compute it one needs time T and **maximum space** S such that $S \cdot T \in \Omega(N^2)$
  - Even on a **parallel** machines



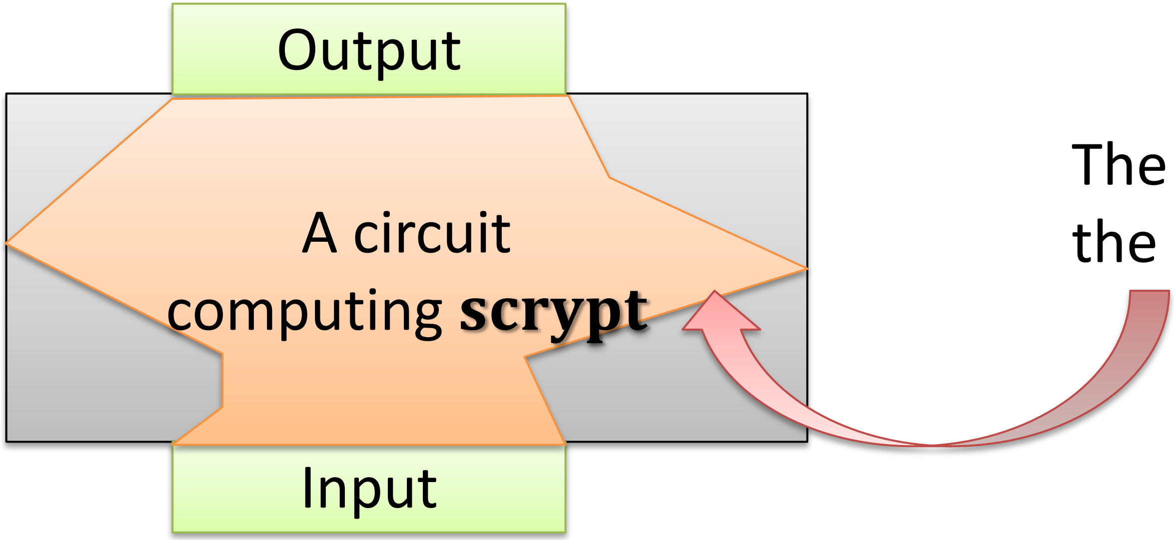
Observation by Alwen-Serbinenko

- Not a very strong bound
- Adversary computing **scrypt** in parallel can **amortize space**



Cumulative Memory Complexity

- The right definition: Sum of memory **actually used** at each point in time



The area in the picture

- Alwen et al. (2016): **scrypt** is **maximally memory hard**

Proofs of Stake

Proofs of Stake (1/2)

- Bitcoin can be seen as running a **lottery**
  - Probability of winning proportional to fraction of computing power
  - The winner is in charge of **proposing the next block**
- Main idea: Make the probability of winning **proportional to the money** (or stake) associated to each public key
  - I.e., shares of coins ≈ voting power

Proofs of Stake (2/2)

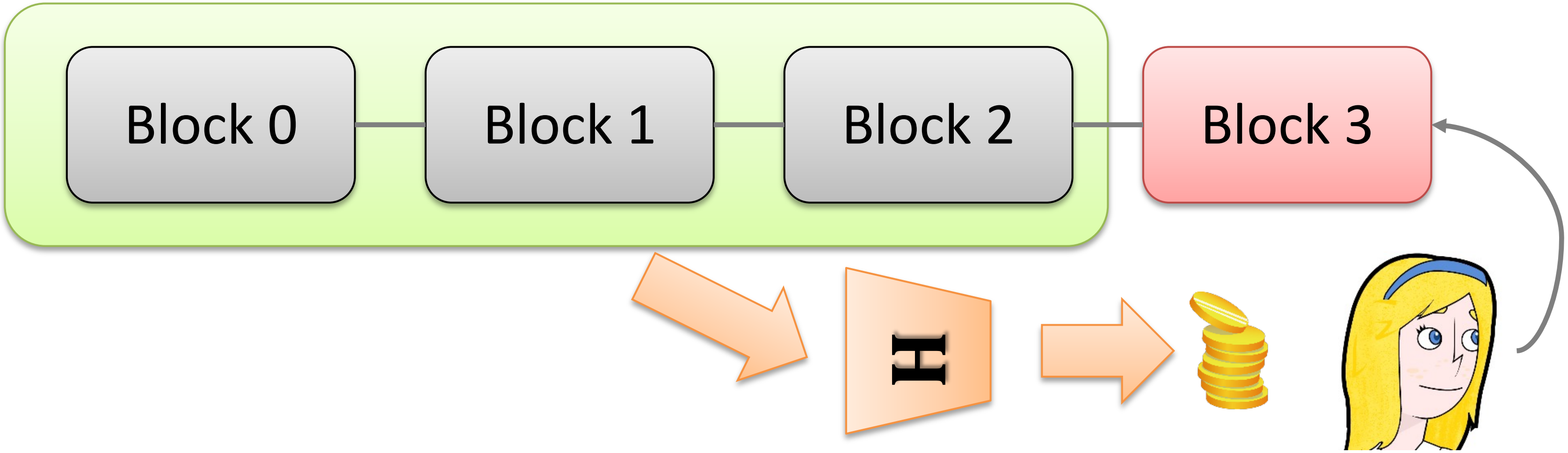
- People who have the money are naturally interested in the **stability** of the currency
- **Assumption:** Honest Majority of Money
  - Money can be used in particular to buy computational power!

Challenges

- How to prevent mining on **many chains**?
  - Since **little computational effort** is required, stakeholders might work **simultaneously** on different chains ("There is nothing at stake!")
- How to prevent **grinding**?
  - The attacker can try to **influence the lottery** to improve its chance of being the leader
- How to distribute initial money?
- How to **incentivize** coin owners to extend the chain?

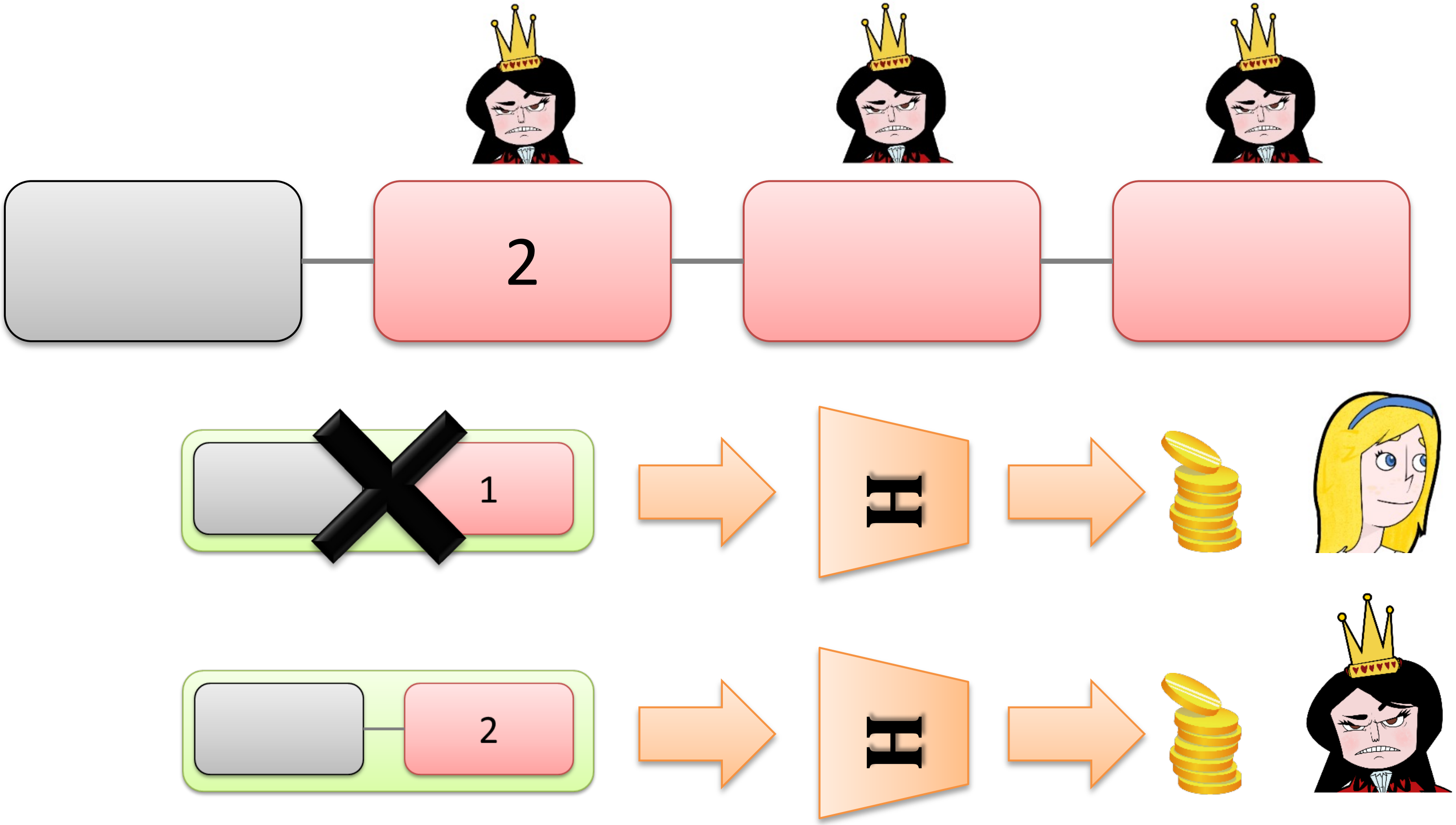
Grinding

- Running the lottery requires **randomness**
- Simple idea: **Hash the blockchain** and use the outcome to select a random coin which corresponds to the winner
  - Assume for simplicity each public key owns 1 coin



Rejection Sampling

- Assume that at **some point** the attacker is elected as **the leader**



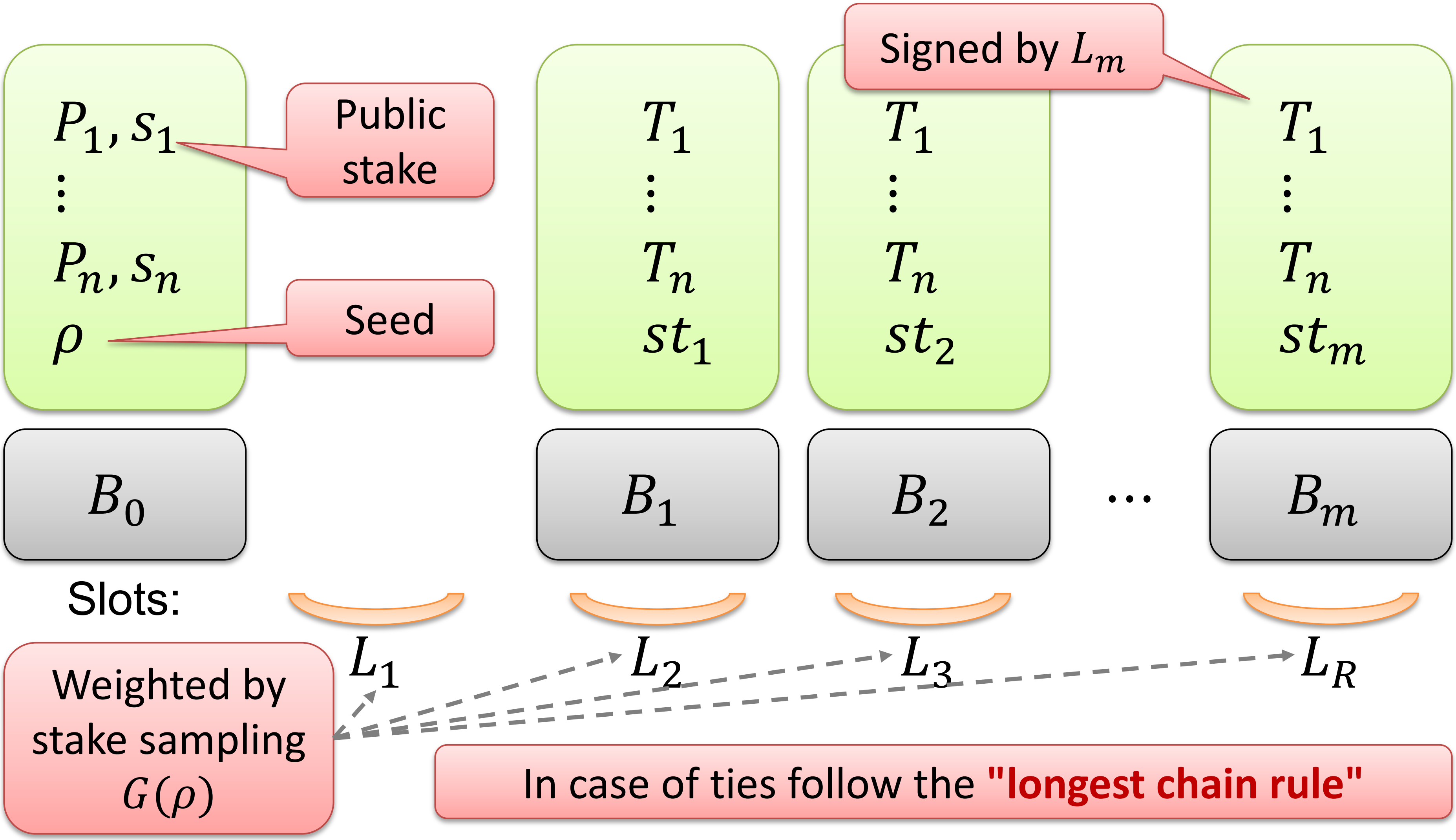
PoS Blockchains with Provable Guarantees

- Ouroboros (Kiayias et al., 2017)
  - Generate **clean randomness** using cryptography
- Snow White (Bentov et al., 2019) and Ouroboros Praos (David et al., 2018)
  - Use **hashing** in a careful manner
- Algorand (Chen and Micali, 2017)
  - Also based on **hashing** but follows a completely **different approach**

Ouroboros: Synchronous Setting

- Time is divided in rounds (also called slots)
  - Messages sent to honest parties **are delivered** by the end of the slot
- Messages sent through a diffusion mechanism
- The attacker is rushing and may
  - Spoof/Inject/Re-order messages
- Assumptions
  - Adversary controls **minority of stake** and subject to **corruption delay**
  - Stake shifts at **bounded rate**

Ouroboros: Static Stake



P_1, s_1
$\vdots$
P_n, s_n
ρ

Public stake

Seed

B_0

T_1
$\vdots$
T_n
st_1

B_1

Signed by L_m

T_1
$\vdots$
T_n
st_2

B_2

$\cdots$

T_1
$\vdots$
T_n
st_m

B_m

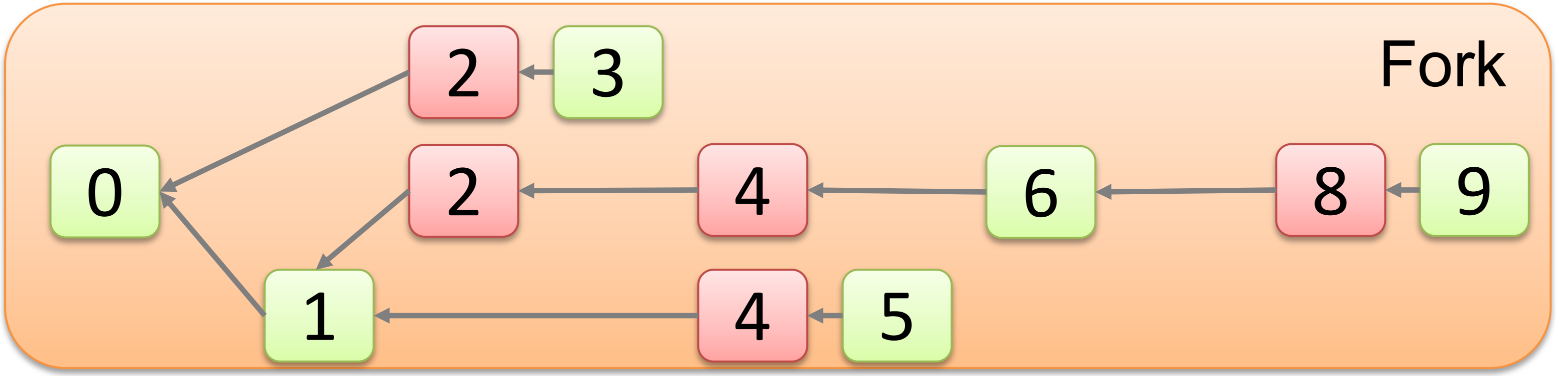
Slots:

L_1 L_2 L_3 L_R

Weighted by stake sampling $G(\rho)$

In case of ties follow the **"longest chain rule"**

Example Dynamics

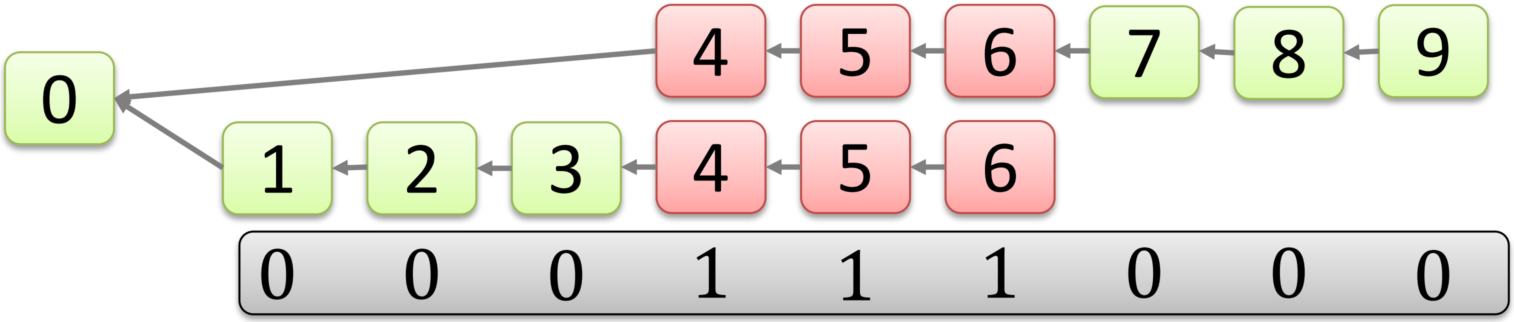


Characteristic string (Bernoulli w.p. $1/2 - \varepsilon$)

- Attacker's advantages (w.r.t. PoW)
  - Sees leaders scheduling **ahead of time**
  - It can generate **multiple different blocks** for the same slot at any time and without any cost

Forkable Strings (1/2)

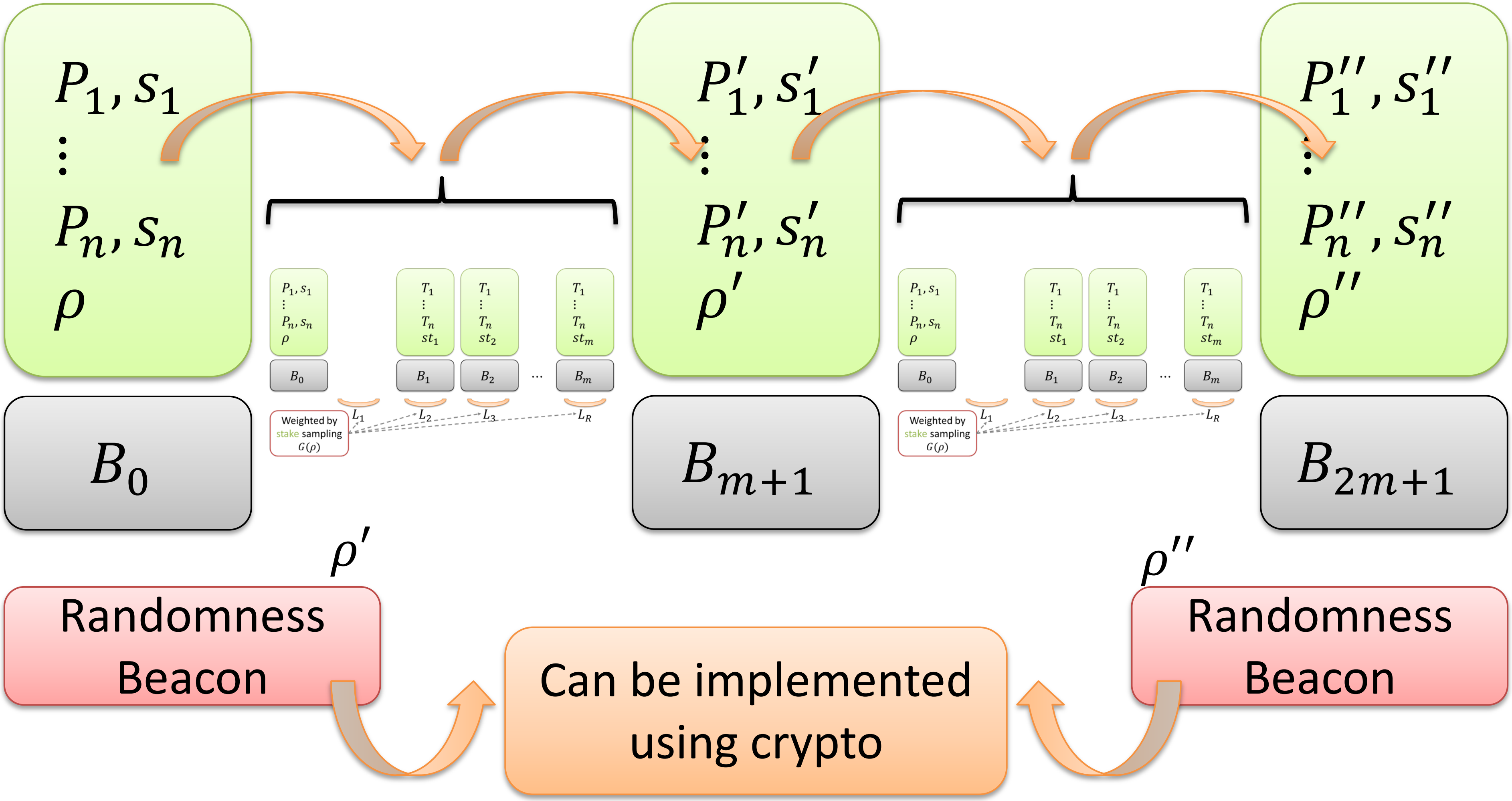
- **Extreme case:** Two **disjoint** paths with the **same maximum length**
  - Call **forkable** a characteristic string where this happens



Forkable Strings (2/2)

- **Theorem:** No string of density $\leq 1/3$ is forkable and all strings of density $\geq 1/2$ are forkable
  - But we want resilience against $1/2 - \varepsilon$ corruptions
- **Theorem:** Draw $w = (w_1, \ldots, w_n)$ from the Binomial distribution with parameter $1/2 - \varepsilon$. Then $\mathbb{P}[w \text{ is forkable}] \leq e^{-\Omega(n)}$

Ouroboros: Dynamic Stake



P_1, s_1
$\vdots$
P_n, s_n
ρ

B_0

P'_1, s'_1
$\vdots$
P'_n, s'_n
ρ'

B_{m+1}

P''_1, s''_1
$\vdots$
P''_n, s''_n
ρ''

B_{2m+1}

ρ'

Randomness Beacon

Can be implemented using crypto

ρ''

Randomness Beacon

The Final Result

Theorem. Assuming a **delay on adaptive corruptions** of $2R - 4k$ slots, Ouroboros satisfies:

Common Prefix	Chain Quality	Chain Growth
k	s	$\tau \geq 1/2\,;s \geq 2k$

- Incentives:
  - A **reward mechanism** is introduced for which Ouroboros can be proven to yield an **approximate Nash equilibrium**
  - In contrast Bitcoin is not incentive compatible!

Algorand

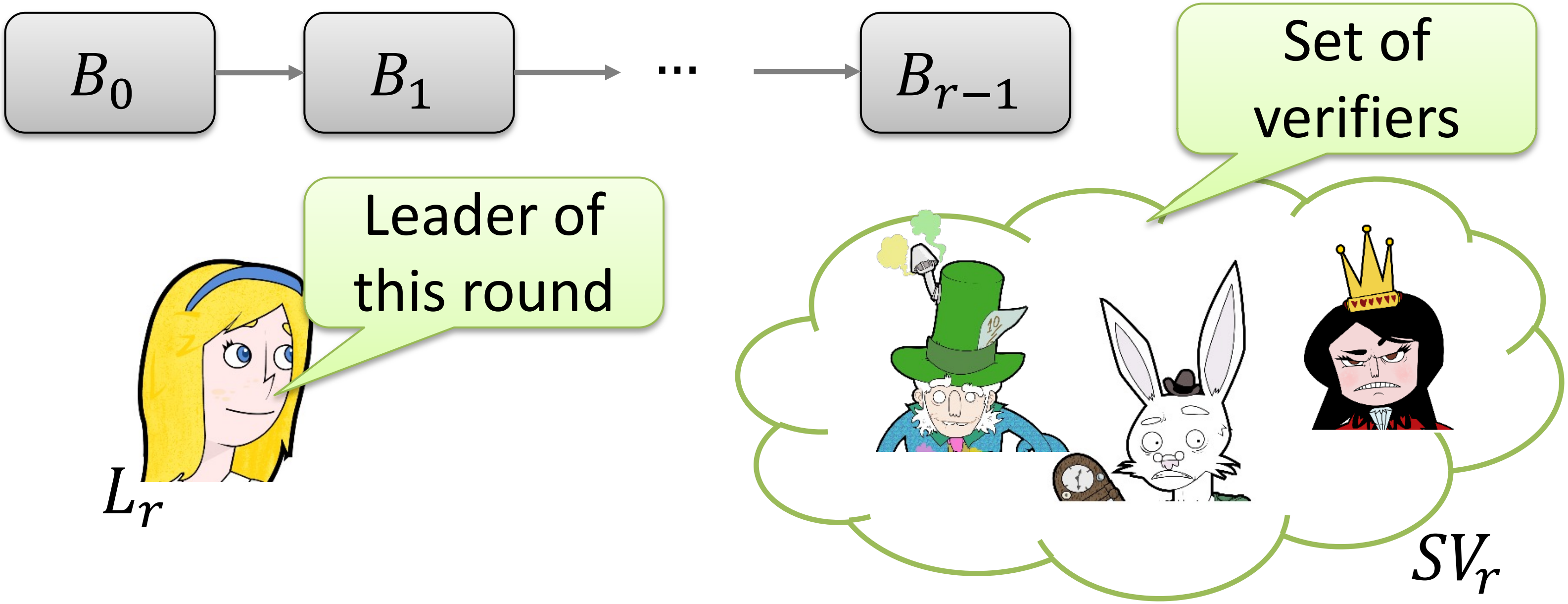
- Developed by a team led by Silvio Micali
- Main goals:
  - Truly distributed and **no concentration of power** (all users are equal)
  - Green (**no waste** of computation)
  - **No forks** (except with probability, say, 10^{-18})
  - Scalability (bottleneck is network latency)

Adversarial Model

- The adversary can **immediately corrupt** any honest user he wants
  - Perfect coordination among corrupted users
- Communication model
  - Message gossipping over complete, asynchronous network (attacker sees all good-to-bad messages)
  - Message sent by honest user reaches 95% of honest users (with some latency)
- Assumptions
  - **Honest majority of stake** and **bounded** stake shifts

Sortition

- In each round different users are selected
  - **Leader:** Assembles and propagates the next block
  - **Set of verifiers:** Need to reach agreement on the block proposed by the (possibly dishonest) leader



B_0 → B_1 → … → B_{r-1}

Set of verifiers

Leader of this round

L_r

SV_r

Secret Cryptographic Sortition (1/4)

- Sortition needs to be **automatic** and **random**
  - Main idea: Use a special quantity Q_r associated to the last block B_{r-1}
  - Hard for the adversary to **predict** who the leader is
- Problem: If the outcome L_r, SV_r is **publicly verifiable**, the adversary can corrupt all users
  - Make the outcome **secret**
  - Each user obtains a **credential** allowing him to prove he was selected as part of L_r, SV_r

Secret Cryptographic Sortition (2/4)

- Unique signatures: Every message has **only one** valid signature (even under malicious pk)
  - Let $\mathbf{sig}_i(m) = \mathbf{S}(sk_i, \mathbf{H}(m))$ for hash function $\mathbf{H}$ and auxiliary signature algorithm $\mathbf{S}$, and $\mathbf{SIG}_i(m) = (i, m, \mathbf{sig}_i(m))$
- Both the set of verifiers and the leader are **selected randomly** between the users already in the system k rounds before r

Secret Cryptographic Sortition (3/4)

- The leader of round r is the user i for which

$$.\,\mathbf{H}\left(\mathbf{SIG}_i(r, 1, Q_{r-1})\right) \leq p$$

  - The quantity $\mathbf{H}\left(\mathbf{SIG}_i(r, 1, Q_{r-1})\right)$ is **uniquely** associated to (i, r)
  - Only user i can verify that he is the leader, but given credentials $\sigma_i^r = \mathbf{SIG}_i(r, i, Q_{r-1}))$ **everybody** can check i is the leader
  - Probability p so that **at least one** potential leader is honest

Secret Cryptographic Sortition (1/3)

- Set of verifiers for step s of round r:

$$.\,\mathbf{H}\big(\mathbf{SIG}_i(r, s, Q_{r-1})\big) \leq p'$$

  - **Only** user i can check he is elected but given $\sigma_i^{r,s} = (\mathbf{SIG}_i(r, s, Q_{r-1}))$ everybody can check that
  - Verifier $i \in SV_{r,s}$ sends message $m_i^{r,s}$ including $\sigma_i^{r,s}$
  - Probability p' chosen so that **at least** $2/3$ of the verifiers are honest (proportional to the stake)

Byzantine Agreement

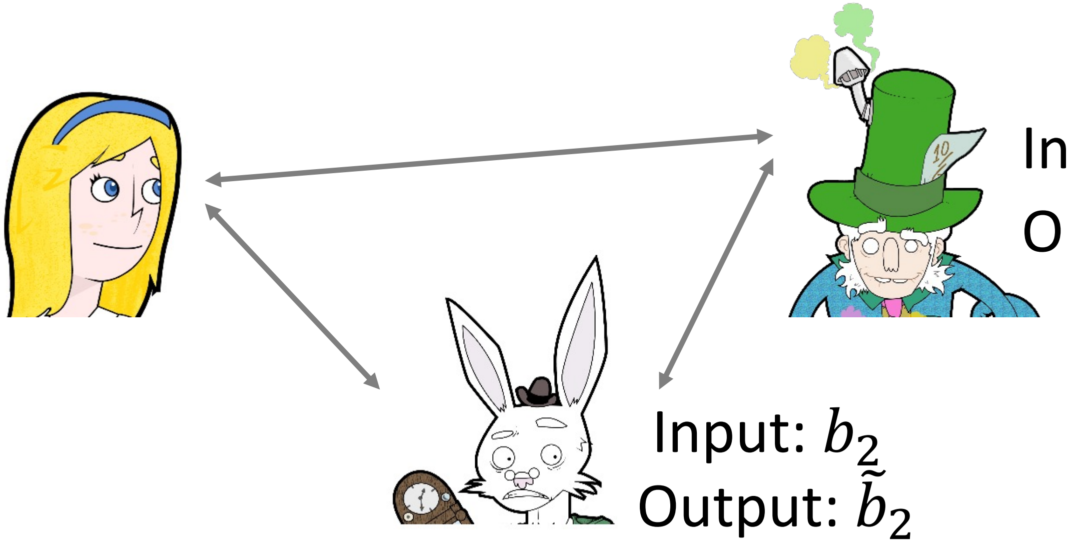
- After the leader is selected it propagates the proposed block to the verifiers in SV_i
  - The verifiers need to agree on the proposed block
  - This is achieved via a protocol for so-called **Byzantine Agreement (BA)**
  - M. Pease, R. Shostak, L. Lamport. "Reaching agreement in the presence of faults." 1980
- The agreed upon block is then **certified** via digital signatures and **propagated** to the network

The Byzantine Generals Problem

- Generals need to decide to attack/retreat
- If some attack and some not they lose (and get killed by the Sultan)
- Main problem: **Cheaters**
  - Can trick honest generals
- **Classical setting:** Number of parties is **fixed**, and parties are connected by **point-wise bidirectional channels**

Problem Statement

- Total of n parties connected by p2p network
- Maximum $t < n$ parties are malicious
- **Input:** Each party P_i inputs bit b_i
- **Output:** Each party P_i outputs bit $\tilde{b}_i$



Input: b_1
Output: $\tilde{b}_1$

Input: b_3
Output: $\tilde{b}_3$

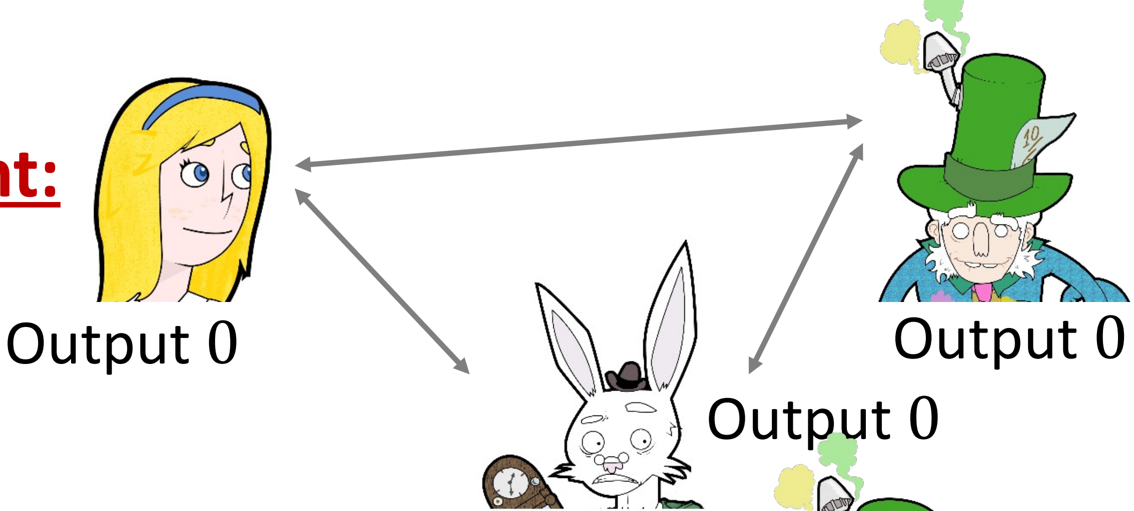
Input: b_2
Output: $\tilde{b}_2$

Security Definition

- **Termination:** Protocol terminates after **finitely many** rounds
  - Typically poly(n) (optimal is constant)
- **Agreement:** All honest parties agree on the **same output**
  - I.e., if P_i, P_j are both honest we have $\tilde{b}_i = \tilde{b}_j$
- **Consistency:** If initial values of honest players are identical, they decide on **that value**
  - I.e., if $b_i = b$ for all honest P_i, each of them outputs $\tilde{b}_i = b$

Observations

- Trivial to achieve consistency or agreement **in isolation**
  - **Agreement:**

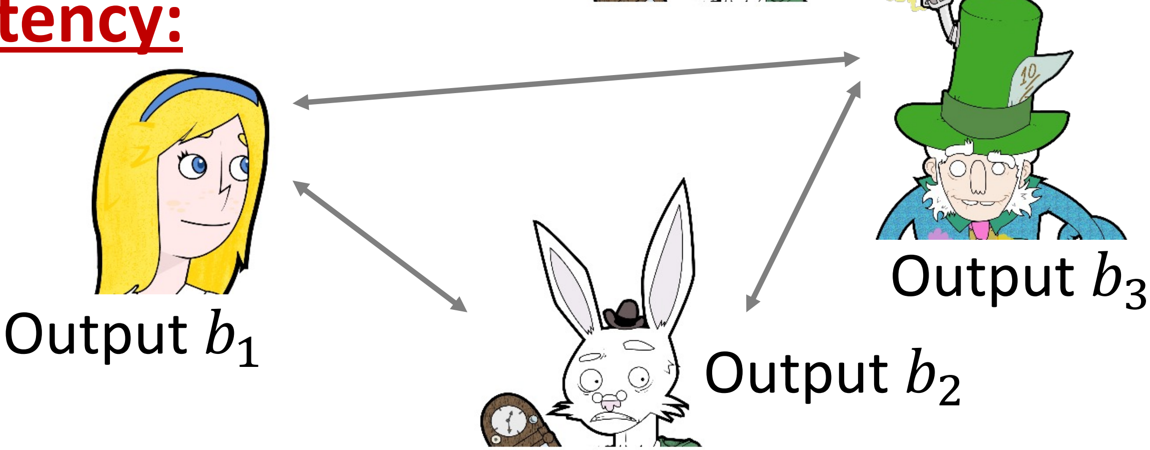


Output 0

Output 0

Output 0

  - **Consistency:**



Output b_1

Output b_3

Output b_2

Facts on Byzantine Agreement (1/3)

- **At least** t rounds are necessary to **deterministically** tolerate t corruptions
- Can tolerate $O(\sqrt{n})$ corruptions in $O(1)$ rounds, via **probabilism**
  - M. Rabin. "Randomized Byzantine generals." 1983

Facts on Byzantine Agreement (2/3)

- And in fact even $n/4$ corruptions in expected $O(1)$ rounds (via **complex** protocol)
  - P. Feldman and S. Micali. "An optimal probabilistic algorithm for synchronous byzantine agreement." 1988
- **Without** assuming a PKI Byzantine agreement is impossible iff $t < n/3$
  - D. Dolev and H.R. Strong. "Authenticated algorithms for Byzantine agreement." 1983

Facts on Byzantine Agreement (3/3)

- **Domain Extension:** Given BA protocol **for bits**, can costruct BA protocol for **arbitrary values** (with overhead of 2 rounds)
  - R. Turpin and B. Coan. "Extending binary Byzantine agreement to multivalued Byzantine agreement." 1984

Broadcast versus Byzantine Agreement

- **Theorem:** If $t < n/2$ broadcast implies Byzantine agreement
- Design protocol for Byzantine agreement
  - All parties send input b_i
  - Each party outputs **majority** of received values
  - **Agreement:** All P_i receive same message via broadcast channel (majority uniquely defined)
  - **Consistency:** If all honest parties start with same input b than all honest parties output b

Let’s Focus on Broadcast!

- Setup: Total of n parties with sender P_s for some $s \in [n]$, out of which $t < n$ malicious
  - **Only sender** has input
  - Honest players decide on output $\tilde{b}_i$
- **Termination:** Protocol terminates after **finite number** of rounds
- **Agreement:** For all honest P_i, P_j, then $\tilde{b}_i = \tilde{b}_j$
- **Consistency:** If P_s is honest, all honest parties P_i output $\tilde{b}_i = b_s$

Dolev-Strong Protocol

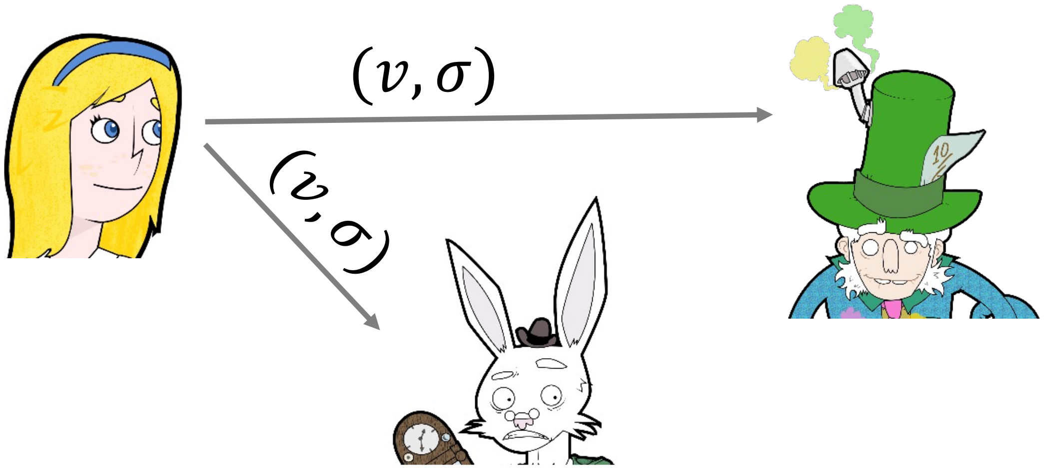
- Goal: Implement broadcast **using PKI**
- Building block: Digital signatures
- Variables maintained by each P_i
  - ACC_i: set of accepted values
  - $SET_{i,0}$: set of signatures received from other parties on message 0
  - $SET_{i,1}$: set of signatures received from other parties on message 1
- Protocol proceeds in 3 stages

Stage 1 (Round $r = 0$)

- Only the sender P_s is active
- All parties initialize

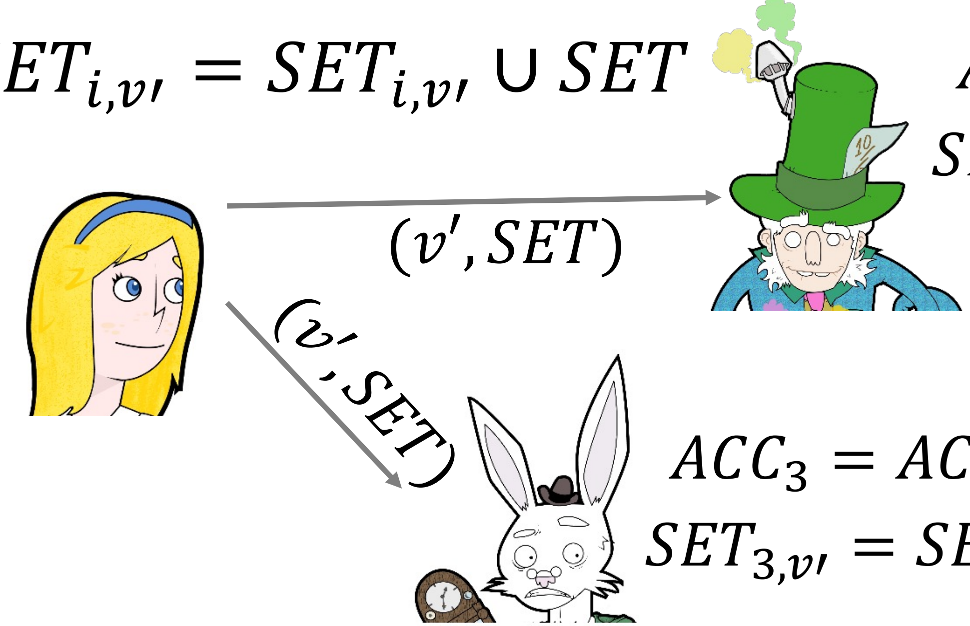
$$ACC_i = SET_{i,0} = SET_{i,1} = \emptyset$$

- P_s sends $(v, \sigma = \mathbf{S}(sk_s, v))$ to everybody
- Finally P_s terminates and outputs v



Stage 2 (Round $r = 1,2,3, \dots$)

- If P_i receives (v', SET) from P_j with $v' \in \{0,1\}$ and where SET contains **valid** signatures on v' from at least r parties (including P_s), then
  - $ACC_i = ACC_i \cup \{v'\}$
  - $SET_{i,v'} = SET_{i,v'} \cup SET$



(v', SET)

(v', SET)

$ACC_2 = ACC_2 \cup \{v'\}$

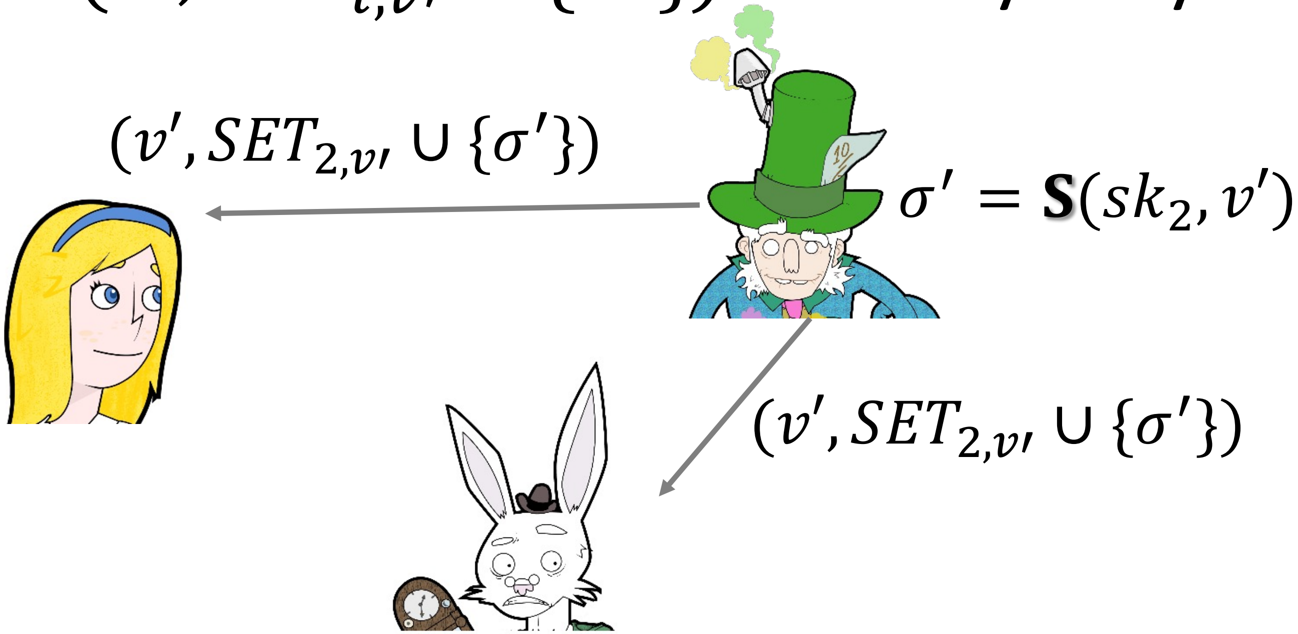
$SET_{2,v'} = SET_{2,v'} \cup SET$

$ACC_3 = ACC_3 \cup \{v'\}$

$SET_{3,v'} = SET_{3,v'} \cup SET$

Stage 2 (Round $r = 1,2,3, \ldots$)

- Each P_i checks if v' was **newly added** to ACC_i during round r
- In that case, it computes $\sigma' = \mathbf{S}(sk_i, v')$ and sends $(v', SET_{i,v'} \cup \{\sigma'\})$ to everybody



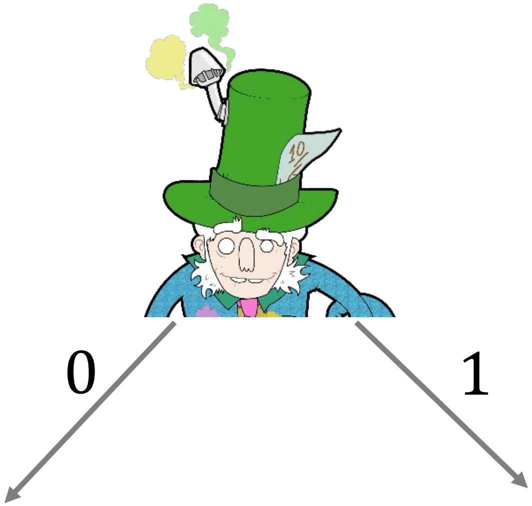
$(v', SET_{2,v'} \cup \{\sigma'\})$

$\sigma' = \mathbf{S}(sk_2, v')$

$(v', SET_{2,v'} \cup \{\sigma'\})$

Stage 3 (Final Round)

- Each P_i proceeds as follows
  - If $ACC_i = 1$ return 1
  - Else, return 0

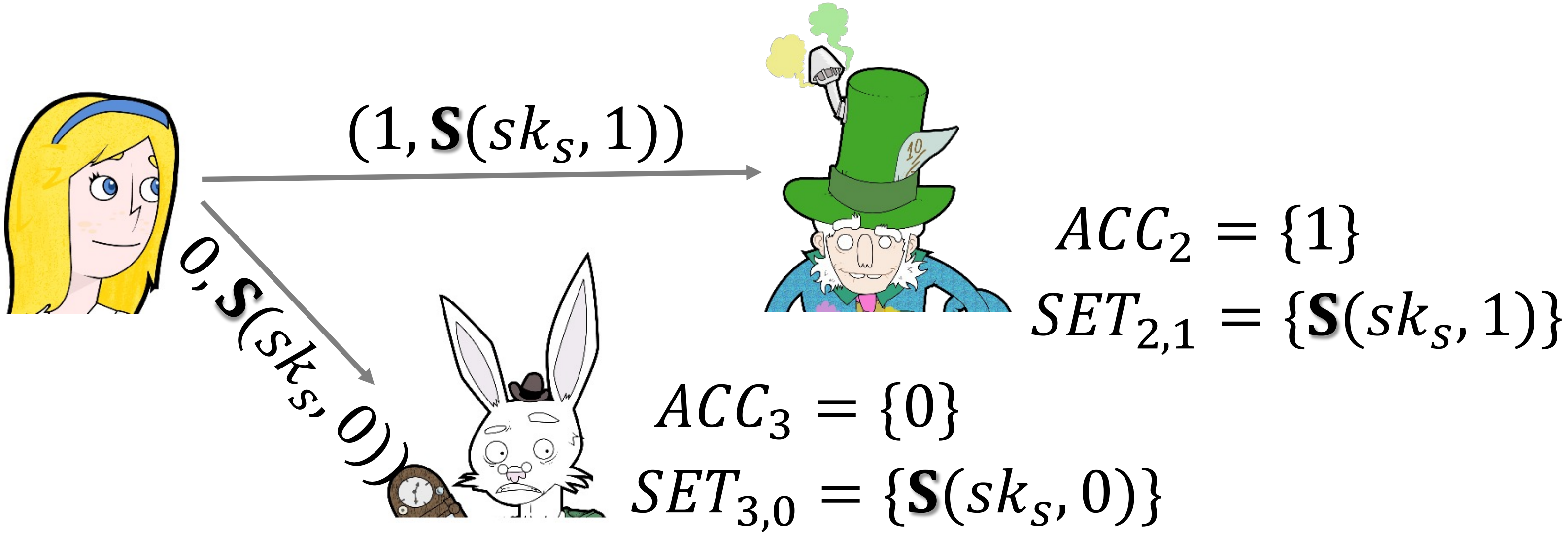


Consistency

- Assume P_s is honest
- Stage 1: P_s sends $v, \sigma = \mathbf{S}(sk_s, v)$
- Stage 2:
  - All honest P_i add v to ACC_i in round $r = 1$ (as σ is accepting) and afterwards **resend signatures**
  - Malicious parties in round $r = 1$ might send $v', \sigma = \mathbf{S}(sk_i, v')$ for $v' \neq v$ (but **never accepted** in future rounds since it does not cointain signature from P_s)
- Stage 3: All parties output v

Agreement (1/3)

- Assume P_s is malicious (honest case is as before)
- Situation after round $r = 1$



$(1, \mathbf{S}(sk_s, 1))$

$0, \mathbf{S}(sk_s, 0))$

$ACC_2 = \{1\}$

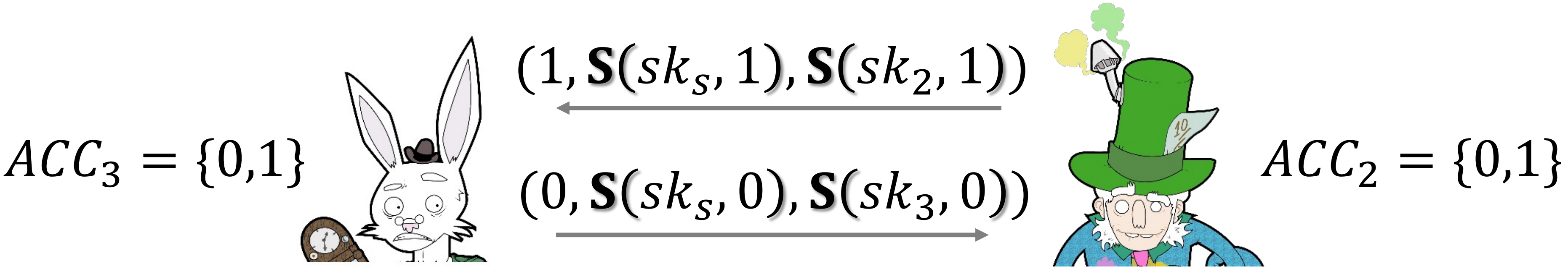
$SET_{2,1} = \{\mathbf{S}(sk_s, 1)\}$

$ACC_3 = \{0\}$

$SET_{3,0} = \{\mathbf{S}(sk_s, 0)\}$

Agreement (2/3)

- Round $r = 2$



$ACC_3 = \{0,1\}$

$(1, \mathbf{S}(sk_s, 1), \mathbf{S}(sk_2, 1))$ ←

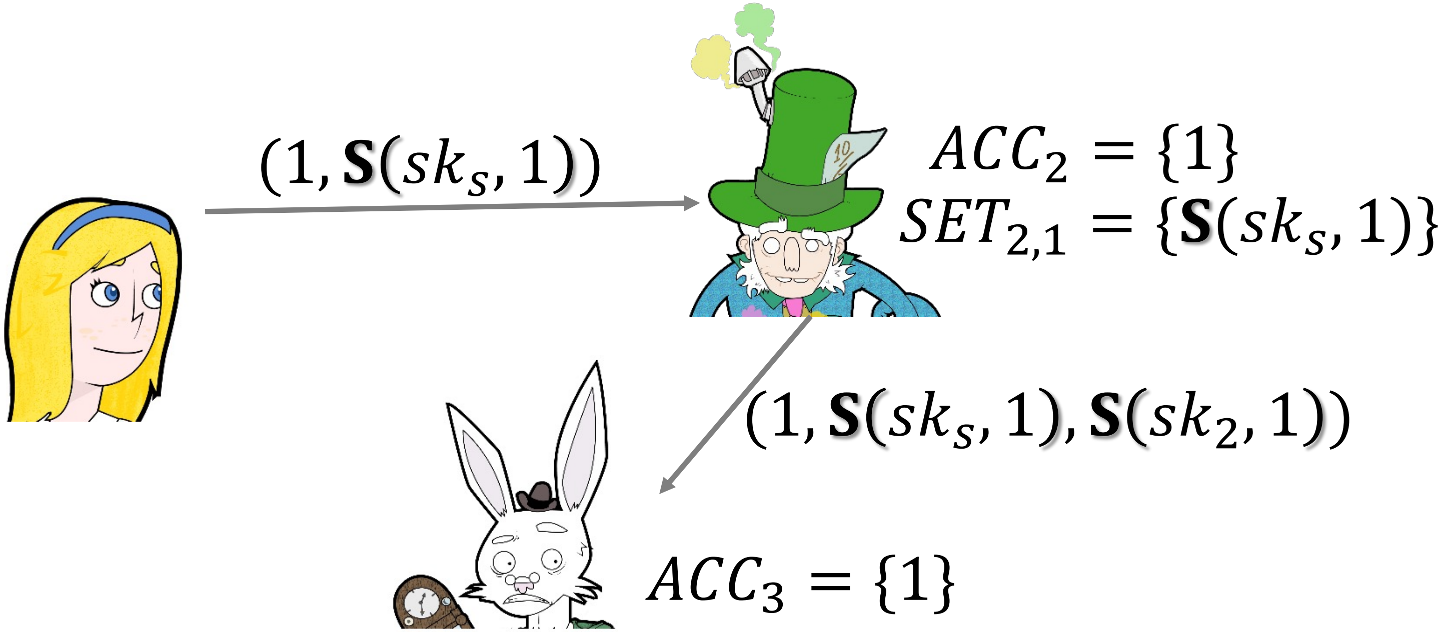
$(0, \mathbf{S}(sk_s, 0), \mathbf{S}(sk_3, 0))$ →

$ACC_2 = \{0,1\}$

- Both honest parties output 0 as $ACC_2, ACC_3 \neq \{1\}$

Agreement (3/3)

- What if P_s sends message **only to one party**?



$(1, \mathbf{S}(sk_s, 1))$

$ACC_2 = \{1\}$
$SET_{2,1} = \{\mathbf{S}(sk_s, 1)\}$

$(1, \mathbf{S}(sk_s, 1), \mathbf{S}(sk_2, 1))$

$ACC_3 = \{1\}$

Byzantine Agreement Made Simple

- New protocol tolerating $n/3$ corruptions in expected 6 **trivial rounds** (using a **PKI**)
  - S. Micali. "Fast and furious Byzantine agreement." 2017
- Assumptions
  - Every player has a public key pk_i
  - A **random string** R independent of the pk_i's

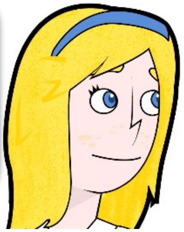
Unique Signatures: $\forall pk_i, m$ **at most one** $\mathbf{SIG}(sk_i, m) = \mathbf{SIG}_i(m)$

Random oracle: $\mathbf{H}(\mathbf{SIG}_i(m))$ **unique, random** string $\forall i, m$

Generic Round

- Instructions for
  - **Reaching agreement** at the end of the round w.p. 1/3 (if not already in agreement)
  - **Remaining in agreement**, if already in agreement
  - Let γ be a counter (initially set to 0)

Received from any "willing" player



$b_j^{r-1}, \mathbf{SIG}_j(R, \gamma)$

- If $\#(0) > 2n/3$, then $b_i^r = 0$
- If $\#(1) > 2n/3$, then $b_i^r = 1$
- Else, $\rho_r = \min_j \mathbf{H}(\mathbf{SIG}_j(R, \gamma))$ and let $b_i^r = \mathbf{lsb}(\rho_r)$

Then, increment the counter γ

Analysis (1/2)

- If agreement on 0 exists, then agreement on 0 **is kept** (similarly for agreement on 1)
- Assume somebody sees more than $2n/3$ 0's
  - The others can't see more than $2n/3$ 1's and thus will follow the **"coin rule"**
  - The bit b_i^r is 0 w.p. $1/2$ and moreover it comes from an honest player w.p. $2/3$
  - Thus, w.p. $1/3$ they also decide on 0, and we get agreement

Analysis (2/2)

- Agreement is reached w.p. $1/3$ in every round
- But players **do not know** when this happens and thus **cannot terminate**
  - Simple but inefficient solution: Repeat for sufficiently large k (say, $k = 300$)
- Run 3 **correlated** executions
  - One with "coin fixed to 0", one with "coin fixed to 1", and one with the "magic coin"
  - The first 2 executions allow players to **understand when agreement is reached**

Adaptations for Algorand (1/2)

- **Gossiping** (instead of multicast)
- Honest majority **of money** (instead of honest majority of users)
- Value R replaced by $Q_r = \mathbf{H}(\mathbf{SIG}_{L_r}(Q_{r-1}, r))$
  - Probabilistic analysis to ensure that the attacker **cannot influence** Q_r

Adaptations for Algorand (2/2)

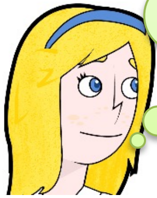
- **Player replaceability**
  - BA still takes more than one round
  - The adversary can still corrupt **the entire set of verifiers** before the second round starts
  - Special property: The protocol works even if each round is executed by **different sets of players**

Another Potential Attack

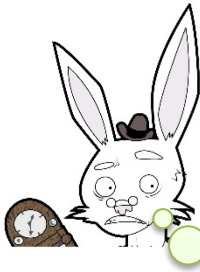
- N. Houy. "It Will Cost You Nothing to Kill a Proof-of-Stake Cryptocurrency." 2014

I am going to destroy this currency by buying > 51% coins and gaining voting majority

If everybody thinks like this the coin price goes to zero and he buys cheaply



Should I sell him my coin?



If I think he succeeds I should sell at any non-zero price

SpaceMint

SpaceMint

- Based on the following papers:
  - Dziembowski et al., "Proofs of Space", 2015
  - Park et al., "A Cryptocurrency Based on Proofs of Space", 2015
- Main idea: Replace work by **disk space**
- Advantages:
  - No dedicated hardware
  - Less energy waste **("greener")**

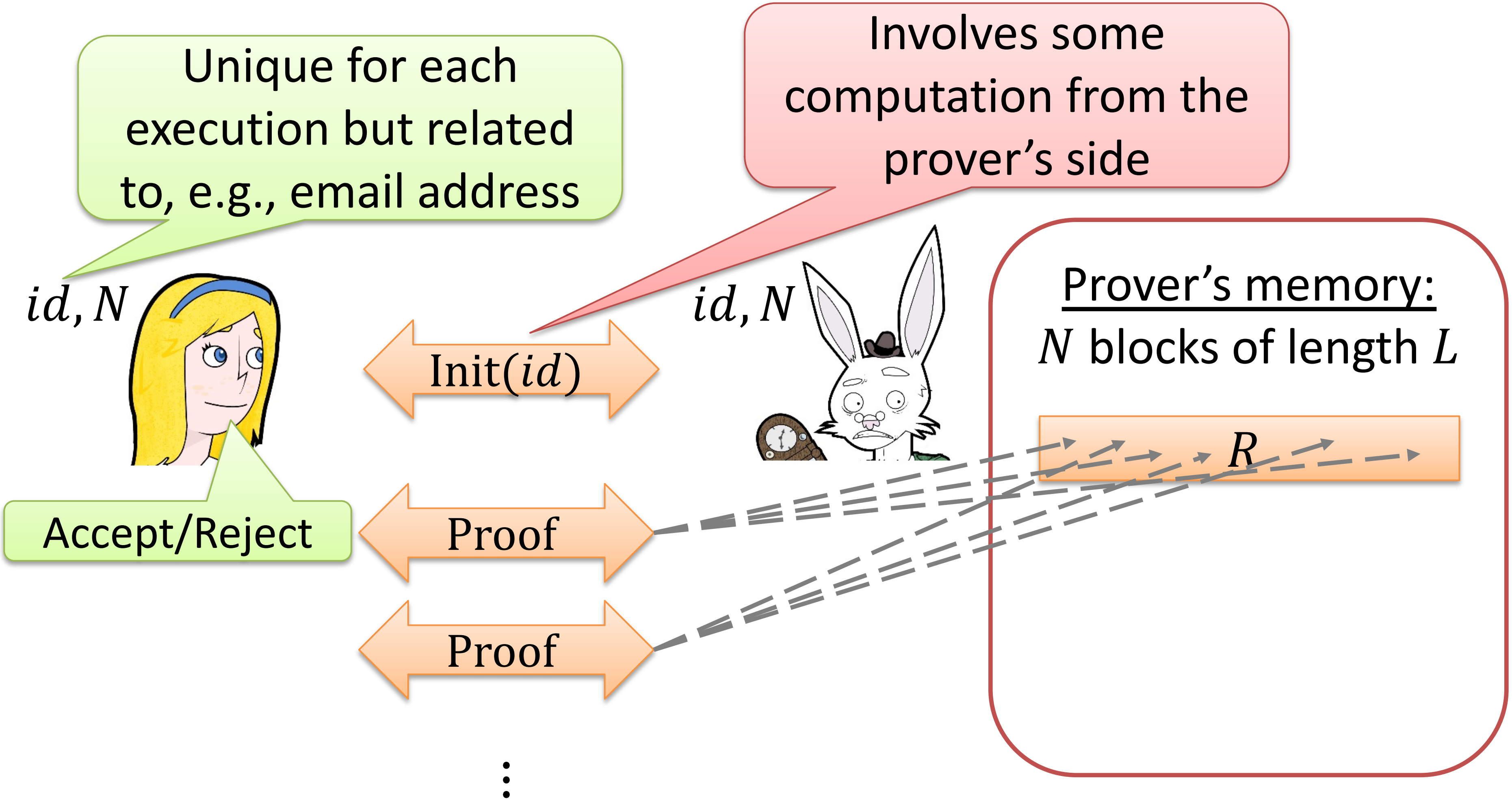
Application beyond Cryptocurrencies

- Goal: Prevent malicious users from opening lots of **fake accounts**
  - E.g. cloud computing services (as gmail)
- Method: Force each account owner to **waste** large part of his **local space**
  - Space remains allocated as long as the user uses the service
  - Periodically the server needs to verify the space is **still allocated**

Advantages

- To prove one wasted n bytes one **does not need to touch all of them**
  - As opposed to CPU cycles in PoW
- More energy efficient
- No hardware acceleration
- Cheaper
  - Users can devote their **unused disk space**

The General Picture



Unique for each execution but related to, e.g., email address

Involves some computation from the prover's side

id, N

id, N

Init(id)

Proof

Proof

⋮

Accept/Reject

Prover's memory:

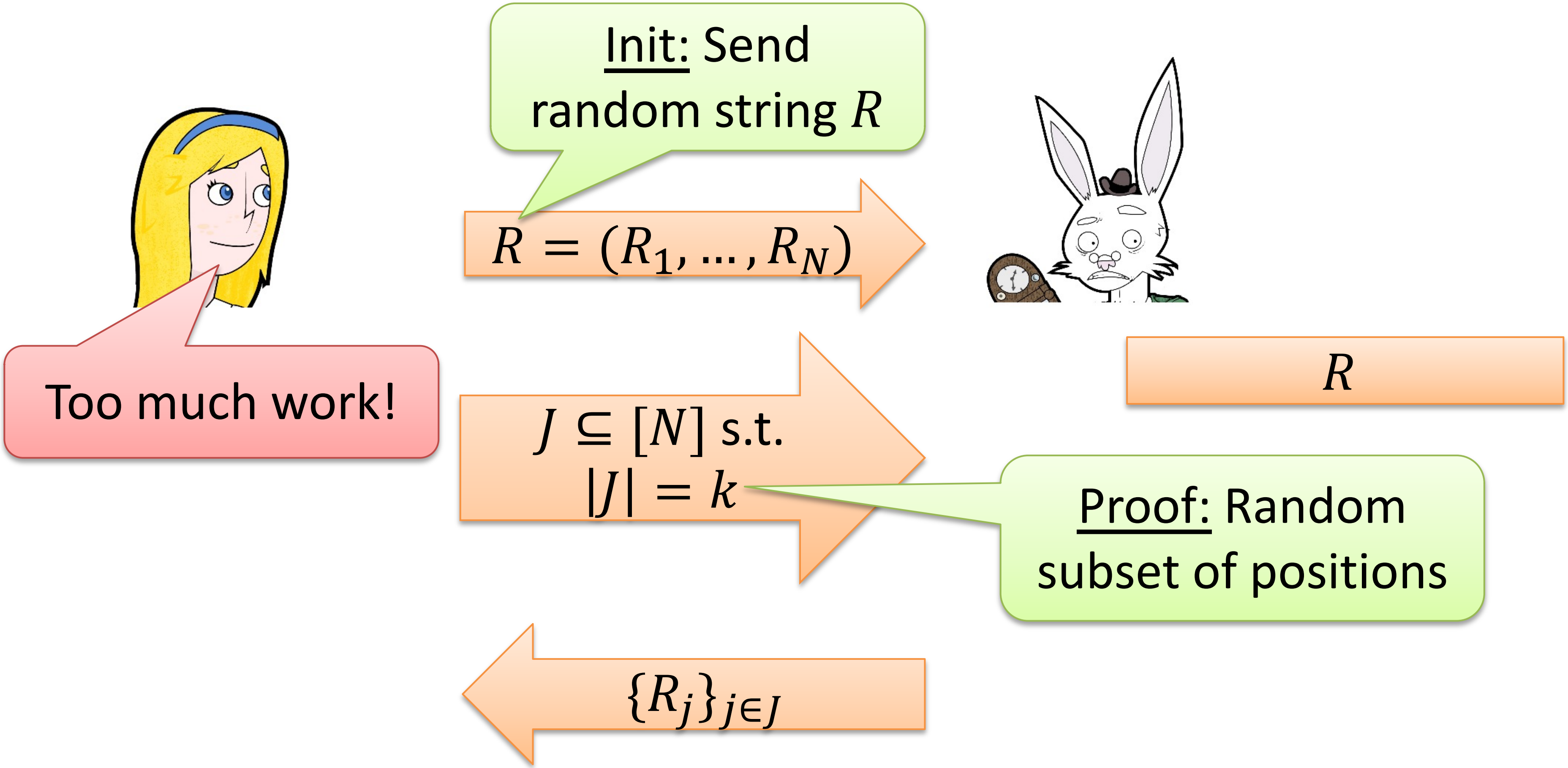
N blocks of length L

R

Security Properties

- **Completeness:**
  - Honest interaction always successful
- **Soundness:**
  - Cheating prover always wastes lots of memory
  - Time measured in terms of # of calls to random oracle $\mathbf{H}$
  - Space measured in terms of # of blocks of length L (output length of $\mathbf{H}$)
- **Efficiency:**
  - To rule out secure but non-efficient solutions

Trivial PoS



Init: Send random string R

$R = (R_1, \dots, R_N)$

R

Too much work!

$J \subseteq [N]$ s.t. $|J| = k$

Proof: Random subset of positions

$\{R_j\}_{j \in J}$

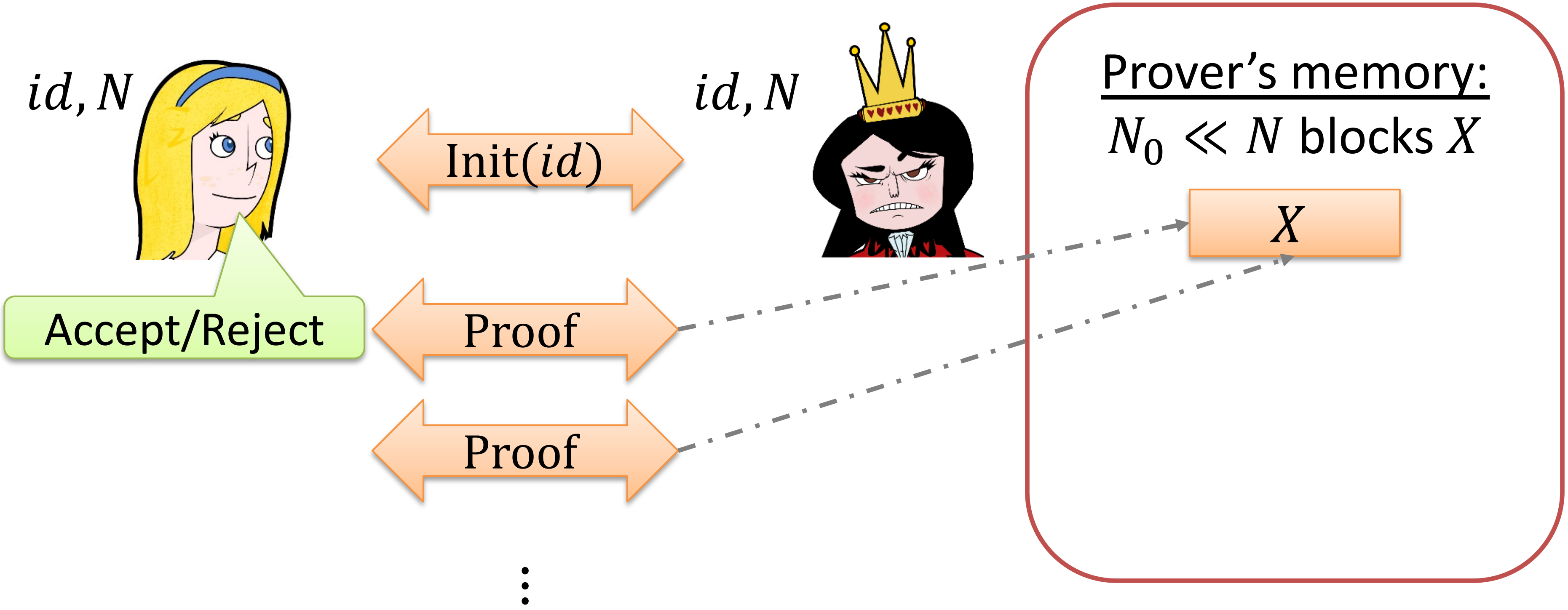
Efficiency

- We require the following bounds for computing times
  - And thus also for communication complexities

	Verifier	**Prover**
Init	$\mathrm{poly}(\log N, k)$	$\mathrm{poly}(N)$
Proof	$\mathrm{poly}(\log N, k)$	$\mathrm{poly}(\log N, k)$

- Example: $\mathrm{poly}(\log N, k) = k \cdot \log N$

Goal of a Cheating Prover



id, N

Accept/Reject

$\text{Init}(id)$

Proof

Proof

$\vdots$

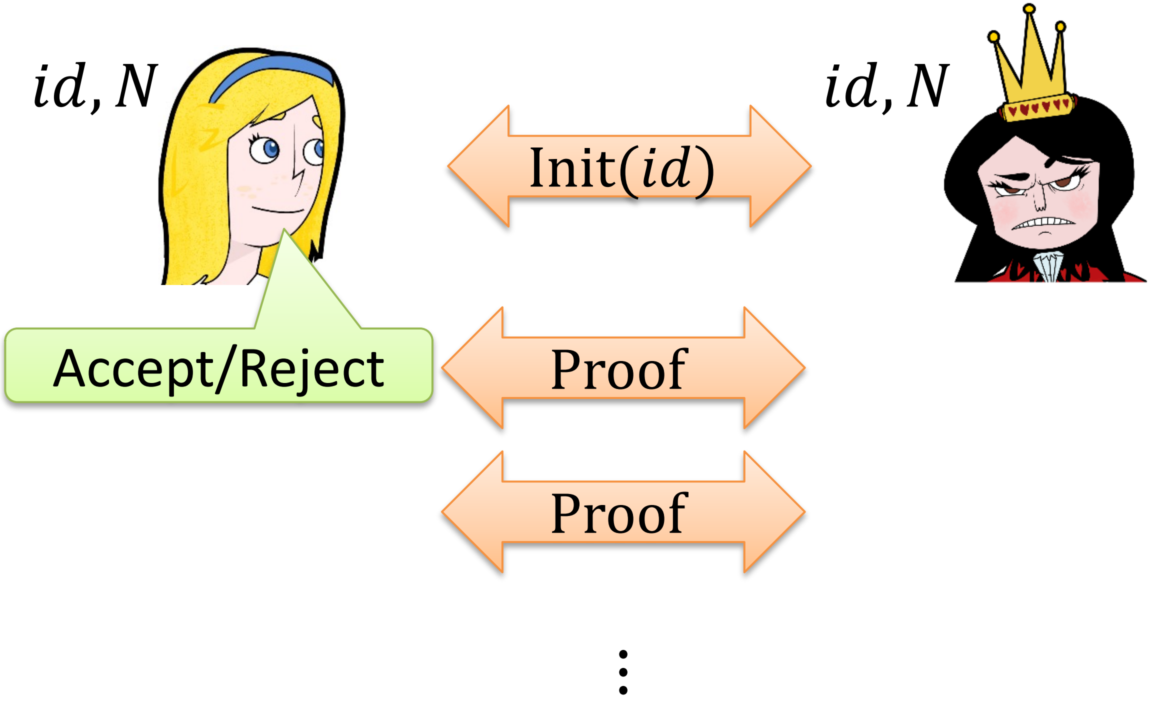
id, N

Prover's memory:

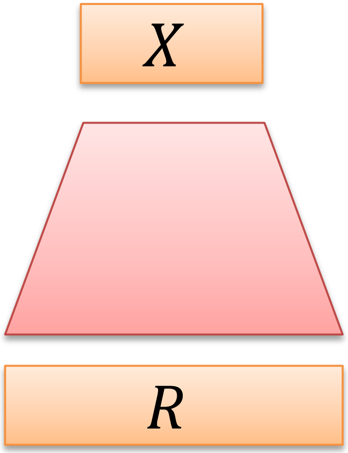
$N_0 \ll N$ blocks X

X

Inefficient Attack



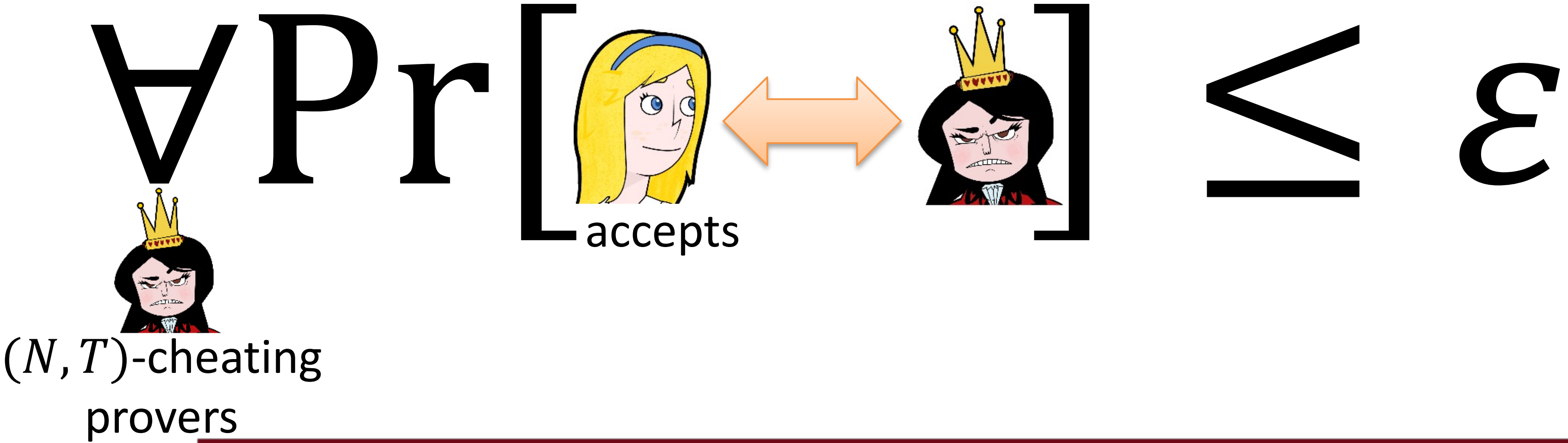
<u>Prover's memory:</u>
Erase R but store all the messages sent by verifier (i.e., $poly(\log N, k)$)



(Afterwards erase R.)

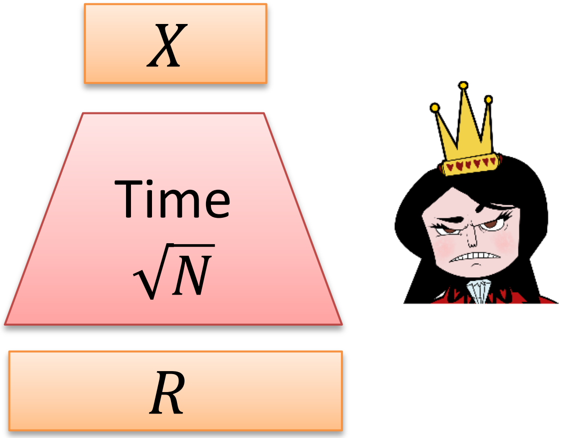
The Definition

- We restrict a cheating prover's **operating time**
  - $\tilde{P}$ is an (N,T)-cheating prover if his storage has size N and his running time during the proof is T
  - **No restriction** on running time during Init phase
- Definition of ε-soundness



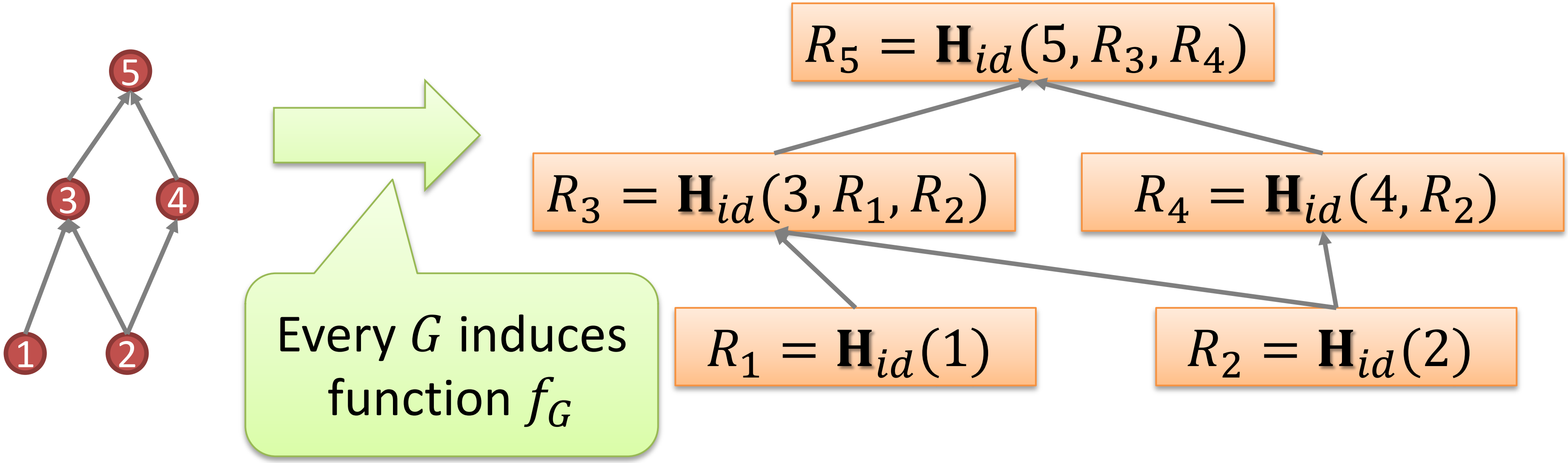
Time-Memory Tradeoffs

- Hardness of constructing PoS is due to so-called **time-memory tradeoffs**
- Example: Instead of storing N blocks, the adversary stores $\sqrt{N}$ blocks
  - Then before each Proof phase can **compute** R in time $\sqrt{N}$



Main Technique

- Let $G = (V, E)$ be a DAG with $|V| = N$
- Let $\mathbf{H}_{id}$ be a hash function depending on id
  - E.g., $\mathbf{H}_{id}(\cdot) = \mathbf{H}'(id || \cdot)$ for auxiliary $\mathbf{H}'$
- Define $R = (R_1, \dots, R_N)$ by **labelling vertices**:



Every G induces function f_G

$R_5 = \mathbf{H}_{id}(5, R_3, R_4)$

$R_3 = \mathbf{H}_{id}(3, R_1, R_2)$

$R_4 = \mathbf{H}_{id}(4, R_2)$

$R_1 = \mathbf{H}_{id}(1)$

$R_2 = \mathbf{H}_{id}(2)$

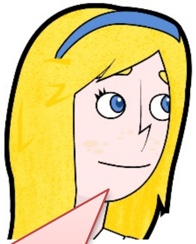
Bad and Good Graphs

- A graph that is bad is one that can be **quickly labelled** by storing a **small number** of labels
- Example of bad graph:

  - Adversary storing labels in position $1, \sqrt{N}, 2\sqrt{N}, \ldots$ can compute all labels in $\sqrt{N}$ steps
- A graph that is not bad is called good

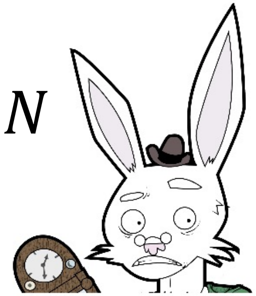
Simple PoS from any Good Graph

id, N



Too much work!

id, N



$R = (R_1, \dots, R_N)$
$= f_G(id)$

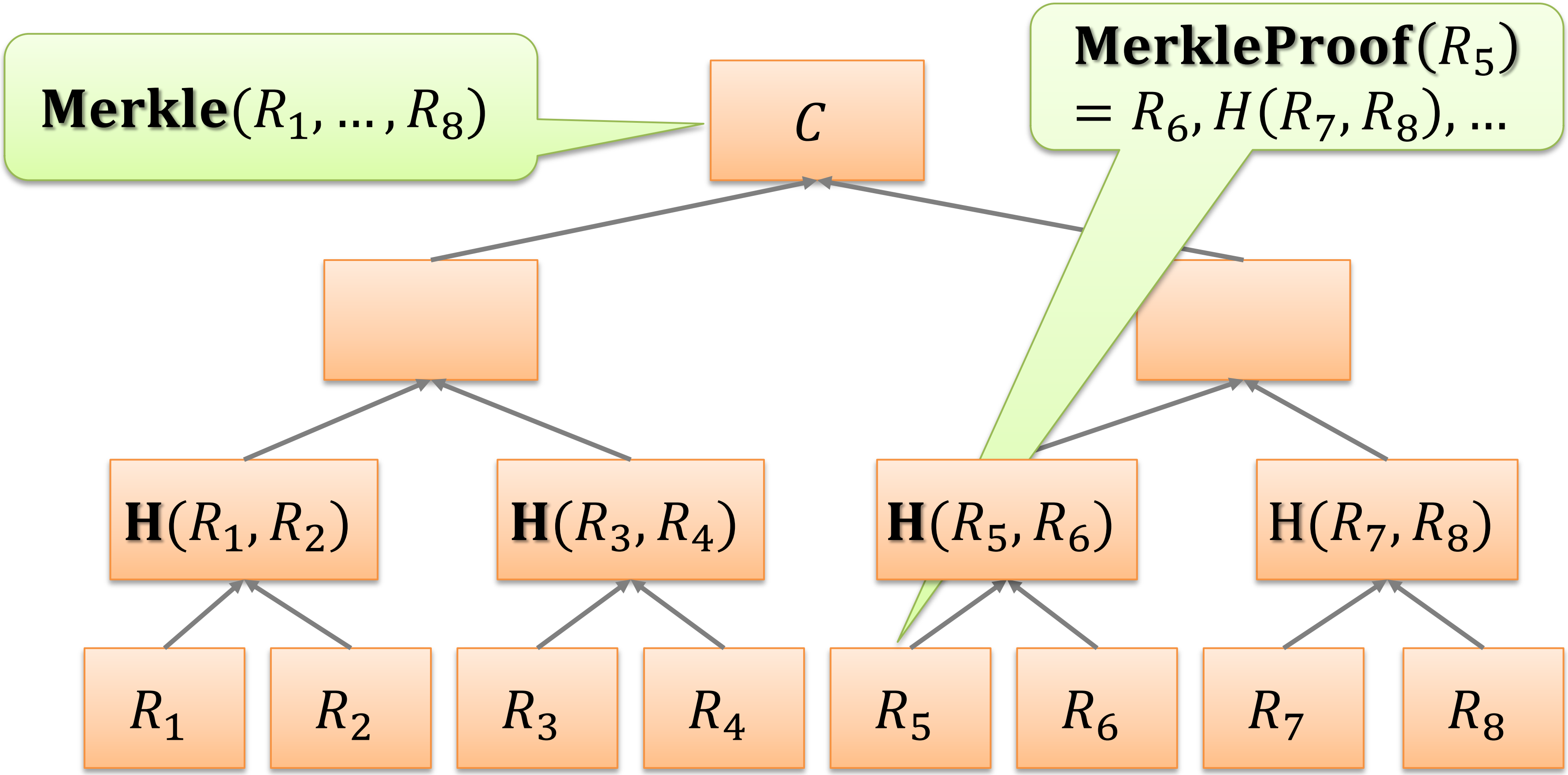
$R = (R_1, \dots, R_N)$

R

$J \subseteq [N]$ s.t.
$|J| = k$

$\{R_j\}_{j \in J}$

Solution: Use Merkle Trees



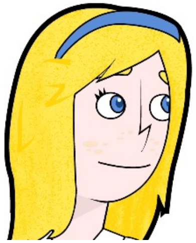
$\mathbf{Merkle}(R_1, \dots, R_8)$

$\mathbf{MerkleProof}(R_5)$
$= R_6, H(R_7, R_8), \dots$

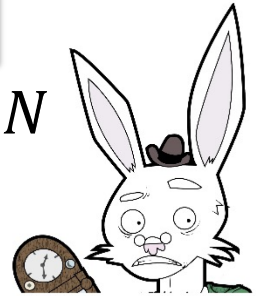
New Init Phase

$\mathbf{Merkle}(R)$

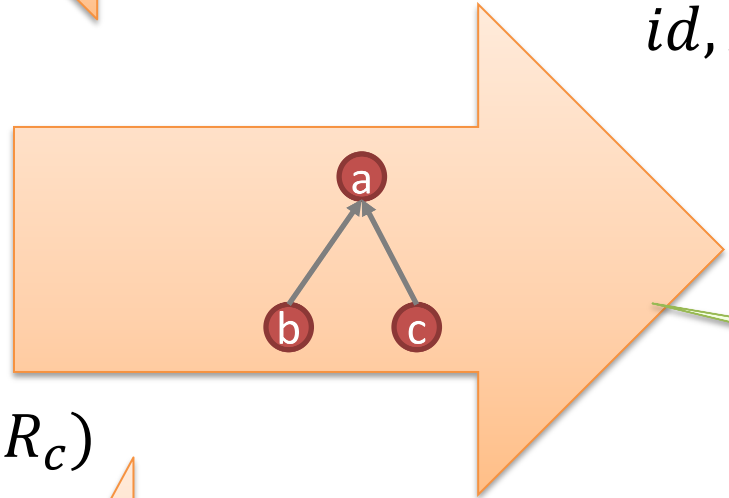
id, N



id, N



$R = (R_1, \ldots, R_N)$
$= f_G(id)$



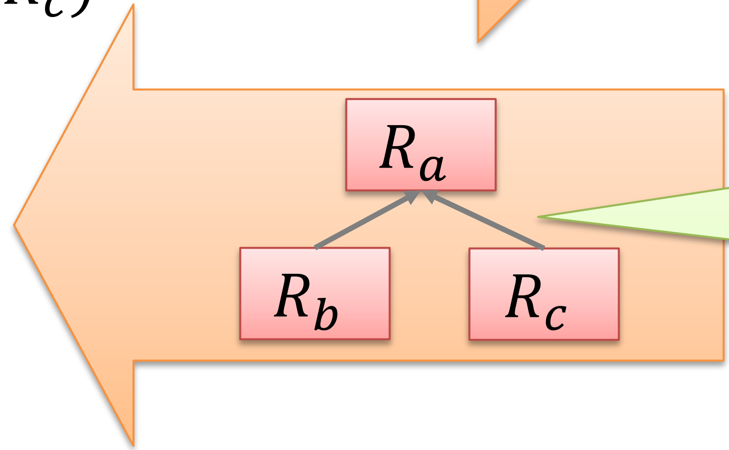
R

Repeat k times

Check:

$R_a = \mathbf{H}_{id}(a, R_b, R_c)$

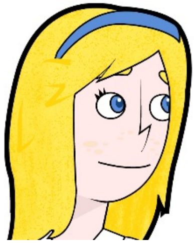
Call such a consistent



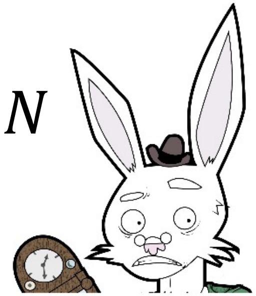
Additionally prove these values are committed in the Merkle tree

New Proof Phase

id, N



id, N



$R = (R_1, \ldots, R_N)$
$= f_G(id)$

$J \subseteq [N]$ s.t.
$|J| = k$

R

$\{R_j\}_{j \in J}$

Additionally prove these values are committed in the Merkle tree

Final Result

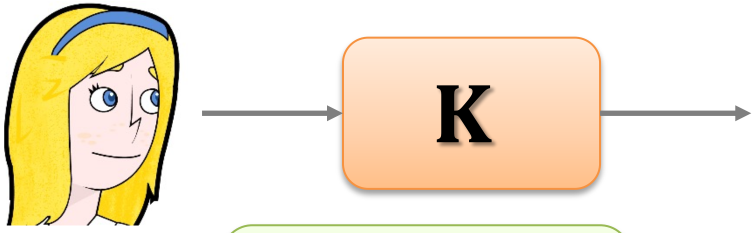
- Assume the adversary committed to graph $\tilde{G}$
  - After Init phase we are sure **large fraction** of nodes in $\tilde{G}$ are consistent
- Some nodes might still be **inconsistent**
  - Adversary not storing x inconsistent nodes with memory N_0 can be simulated with memory $N_0 + x$
- **Theorem:** There exists a $(O(N), O(N))$-PoS
  - Proof constructs good graphs using techniques from **graph pebbling**

Replacing PoW with PoS

- Not immediate how to base a cryptocurrency on a PoS (instead of PoW)
- Some difficulties:
  - PoS runs in **2 stages** (Init + Proof) whereas PoW runs in 1 stage
  - How to make **reward** proportional to the invested resources
  - Where does **the challenge** come from?

Joining SpaceMint

- Every user who wants to join the system declares how much **space** he can devote



(pk, sk)

$R = (R_1, \dots, R_N) = f(pk)$

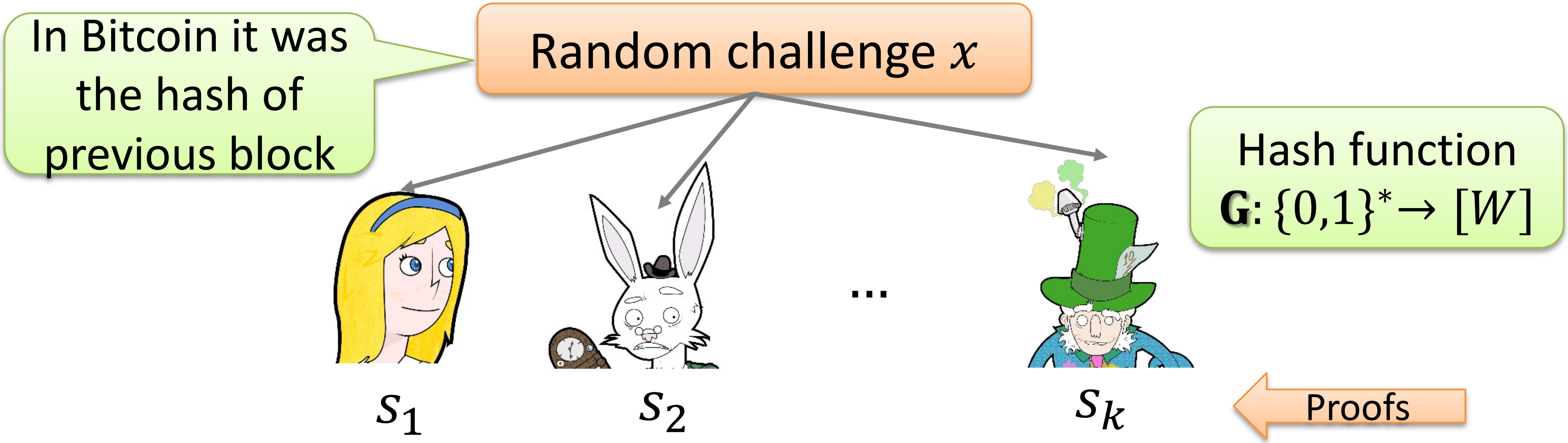
$C = \textbf{Merkle}(R_1, \dots, R_N)$

Can be verified later

- Broadcast special **"commit" transaction** including (pk, C)

Reward in SpaceMint

- Let $N_1, \dots, N_k$ be the memory size of each miner and assume $N_1 = \cdots = N_k$



In Bitcoin it was the hash of previous block

Random challenge x

Hash function $\mathbf{G}: \{0,1\}^* \to [W]$

s_1 s_2 … s_k

Proofs

P_i is the winner if $\mathbf{G}(s_i)$ is **larger** than all other $\mathbf{G}(s_j)$

Reward Calculation (1/2)

- Each player is the winner with probability $1/k$
- This is because for a given commitment C and challenge x the answer s **is unique**
  - As long as one cannot change C (which is why the miners post C on the blockchain)
- Important that miners **can't try different solutions** s
  - Otherwise we would be **back to PoWs**

Reward Calculation (2/2)

- What if the N_i's are **not equal**?
- We need a function D_{N_i} such that the following condition yields a winner w.p. $\frac{N_i}{N_1+\cdots+N_k}$

P_i is the winner if $D_{N_i}(s_i)$ is **larger** than all other $D_{N_i}(s_j)$

- The following function works

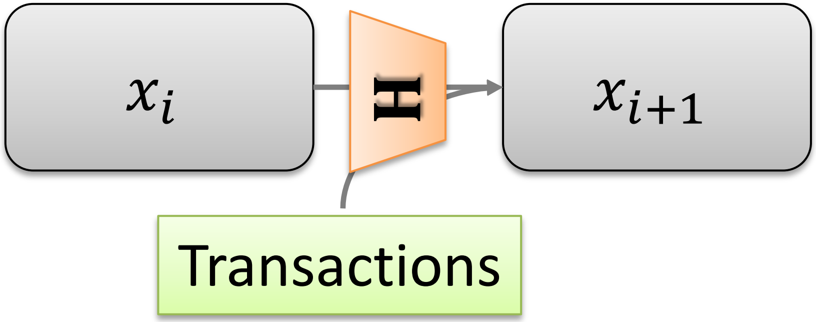
$$D_{N_i}(s) = (\mathbf{G}(s)/W)^{1/N_i}$$

Challenge Generation

- Where does the challenge x come from?
  - In Bitcoin it was the **hash of the last block**
- Use a NIST beacon?
  - Not good for a fully distributed currency
- Ask some other miner?
  - What if he is not online?
- Use previous block (alà Bitcoin)?
  - **Not so easy** as in Bitcoin

Grinding

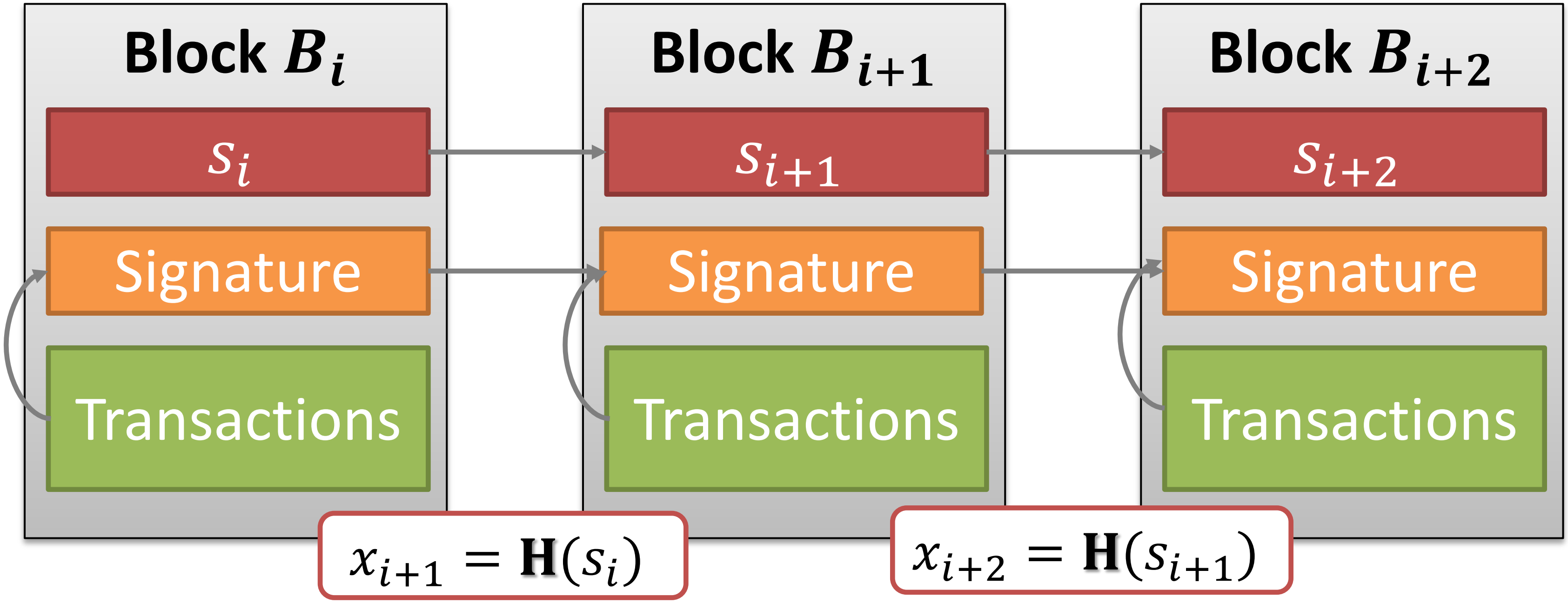
- Problem with using previous block: By **manipulating the transaction list** the miner can produce different x_i's



- Similar to the case of PoSs

Transactions Syntax

- The challenge **does not depend on the transactions**



Block B_i

s_i

Signature

Transactions

Block B_{i+1}

s_{i+1}

Signature

Transactions

Block B_{i+2}

s_{i+2}

Signature

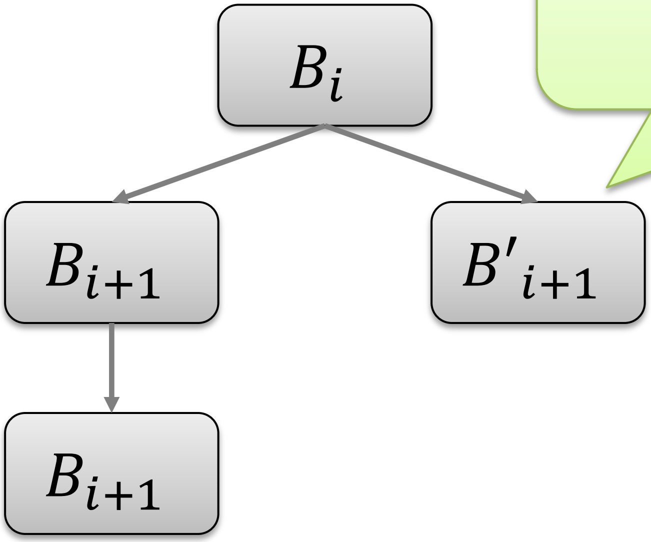
Transactions

$x_{i+1} = \mathbf{H}(s_i)$

$x_{i+2} = \mathbf{H}(s_{i+1})$

Forks

- Suppose there is a **fork**

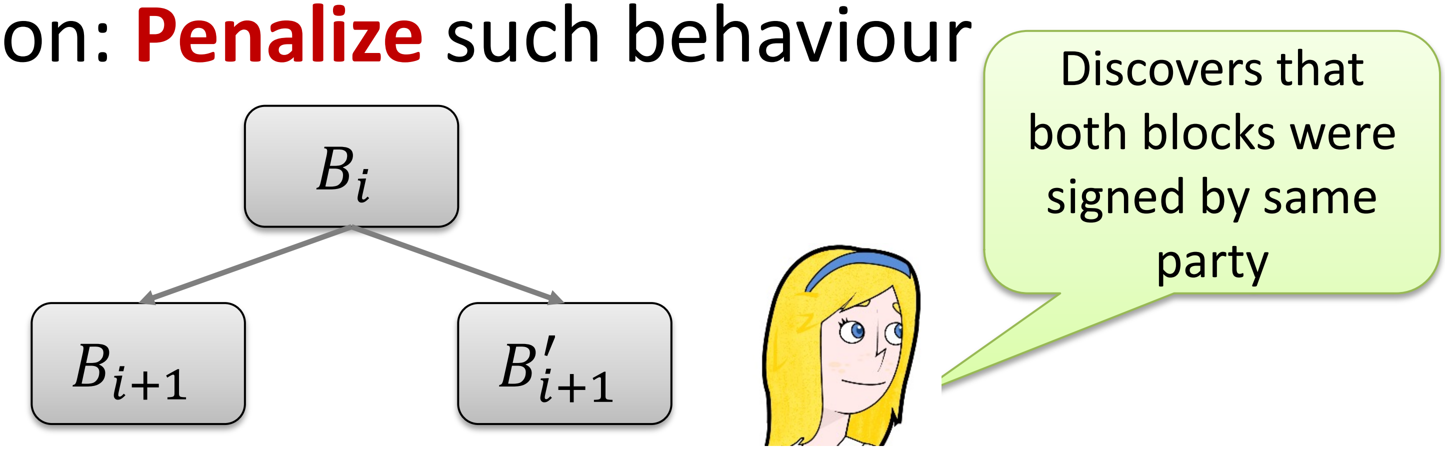


Maybe this block yields a challenge that is better for her

- In Bitcoin it **made no sense**
  - Solution: **Look deeper** in the past (i.e., challenge from block i generated from block $i - 120$)

A Subtle Problem

- In PoW mining costs, while in PoS it is **for free**
- Miners seeing forks could decide to **grow both chains** (so they win in both cases)
- Solution: **Penalize** such behaviour



Discovers that both blocks were signed by same party

  - Post a transaction with a "proof" of this and **get a reward** (the party signing 2 blocks loses her reward)

Permacoin and Primecoin

Permacoin

- Main idea: Parametrize PoW with a **large file** (too large to be stored by individuals)
  - Possibly **useful** data (e.g., the library of congress)
- To solve a PoW need to store parts of the file
  - The more you **store** the more likely it is to win
- Differences with SpaceMint
  - Still a PoW
  - The data is not random
  - Scales less well

A Nice Feature

- The puzzles are **non-outsourceable**
  - A miner in a mining pool could **always** steal the PoW solution
- Thus, it makes **no** sense to create **mining pools**!
- See also:
  - A. Miller, A. E. Kosba, J. Katz, E. Shi. "Nonoutsourceable scratch-off puzzles to discourage Bitcoin mining coalitions." 2014

Finding Chains of Primes

- Cunningham chain of the **first kind**:
  - p_0
  - $p_1 = 2p_0 + 1$
  - $p_2 = 2p_1 + 1$
  - $p_3 = 2p_2 + 1$
  - …
- Example: 2, 5, 11, 23, 47,…

- Cunningham chain of the **second kind**:
  - p_0
  - $p_1 = 2p_0 - 1$
  - $p_2 = 2p_1 - 1$
  - $p_3 = 2p_2 - 1$
  - …
- Example: 151, 301, 601, 1201,…

Bi-twin chain: $p_0, q_0, p_1, q_1, p_2, q_2$,… such that

- $p_0, p_1, p_2, ..$ are a Cunningham chain of the first kind
- $q_0, q_1, q_2, ..$ are a Cunningham chain of the second kind
- (p_i, q_i) are a prime twin pair (i.e., $q_i = p_i + 2$)

Conjecture: For **any** k there are **infinitely many** chains as above of length k

Primecoin

- Main idea: For solving PoW need to find **longest possible** chain of primes
- Verification of a PoW should be fast
  - Limit the size of primes
  - Allow **pseudoprimes**

Fermat Test:

$$2^{n-1} \equiv 1 \bmod n$$

- Quality measure
  - Accept chains $p_1, \ldots, p_{k-1}, p_k$ where all p_i's but p_k are primes
  - Quality is $k + r$ where r measures how close p_k is to be a prime (in terms of Fermat's test)

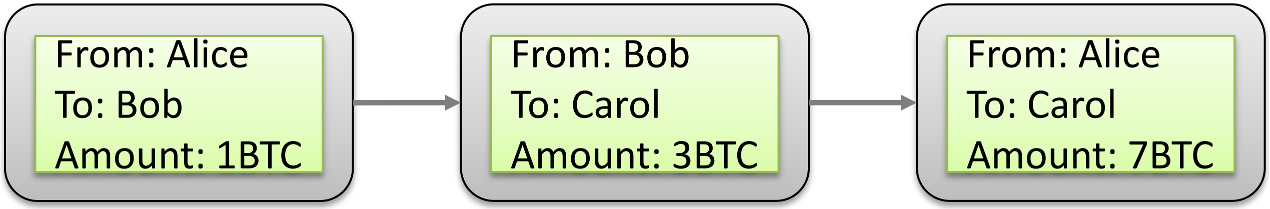
Linking the Blocks

- How to link the current solution to the hash of the previous block B_i?
- Require that $p_1 + 1$ is a **multiple** of $\mathbf{H}(B_i)$
- For more details see:
  - S. King. "Primecoin: Cryptocurrency with prime number proof of work." 2013

ZCash

Bitcoin's Privacy Problem

- Bitcoin prevents doublespending via keeping a consistent public ledger **storing all transactions**



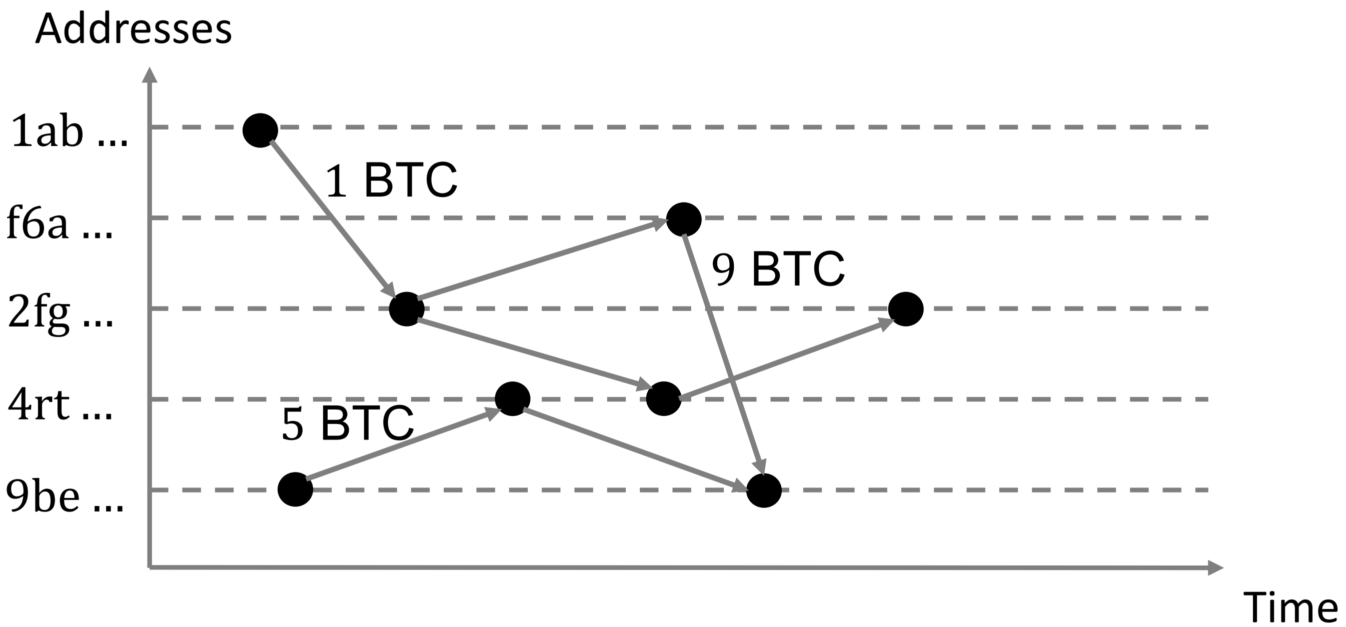
From: Alice
To: Bob
Amount: 1BTC

From: Bob
To: Carol
Amount: 3BTC

From: Alice
To: Carol
Amount: 7BTC

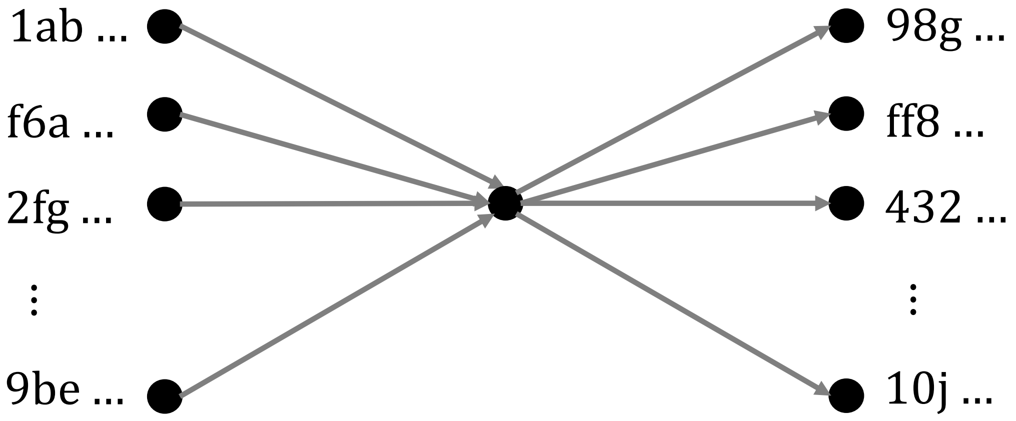
- The cost: **Privacy**!
  - **Consumer purchases** (timing, amounts, merchant) seen by friends, neighbors, and co-workers
  - **Account balance** revealed in every transaction
  - **Merchant's cash flow** exposed to competitors

Those Are Just Addresses!



- Transaction graph + side info
  - Addresses becomes **names of people**
- **Not** just **theoretical**
  - FBI Silk Road Investigations, …

Possible Mitigations

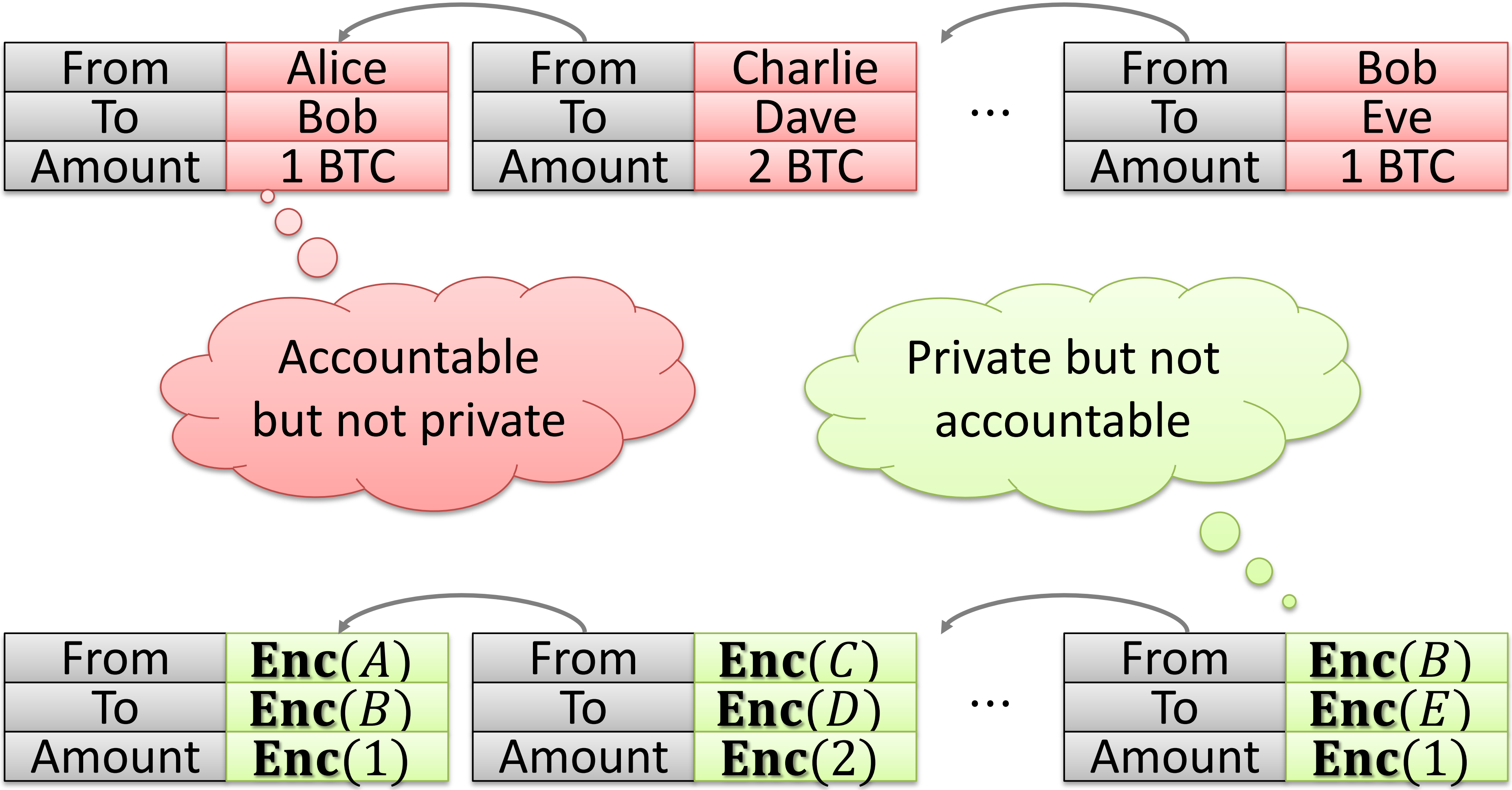


- Use **new address** for each payment
- Launder money with others
- Harder to analyze, **but tracks remain**
  - Blockchain is public **forever**!

Money Fungibility

- "A dollar is a dollar, regardless of its history"
  - Recognized as a crucial property of money more than 350 years ago
- Bitcoin **not fungible**, as coins' pedigree is **public**
  - Ill-defined value (different people value the same coin differently, new coins more valuable than old coins,…)
  - Price discrimination (salary rise yields rent hike)
  - Censorship (miners filter transactions)

Privacy vs Accountability



From	Alice
To	Bob
Amount	1 BTC

From	Charlie
To	Dave
Amount	2 BTC

…

From	Bob
To	Eve
Amount	1 BTC

Accountable but not private

Private but not accountable

From	$\mathbf{Enc}(A)$
To	$\mathbf{Enc}(B)$
Amount	$\mathbf{Enc}(1)$

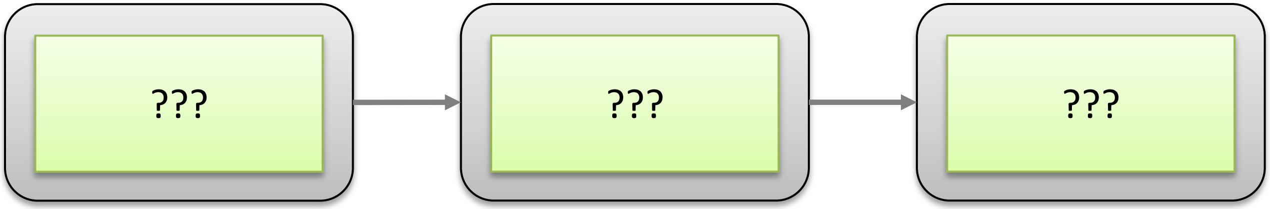
From	$\mathbf{Enc}(C)$
To	$\mathbf{Enc}(D)$
Amount	$\mathbf{Enc}(2)$

…

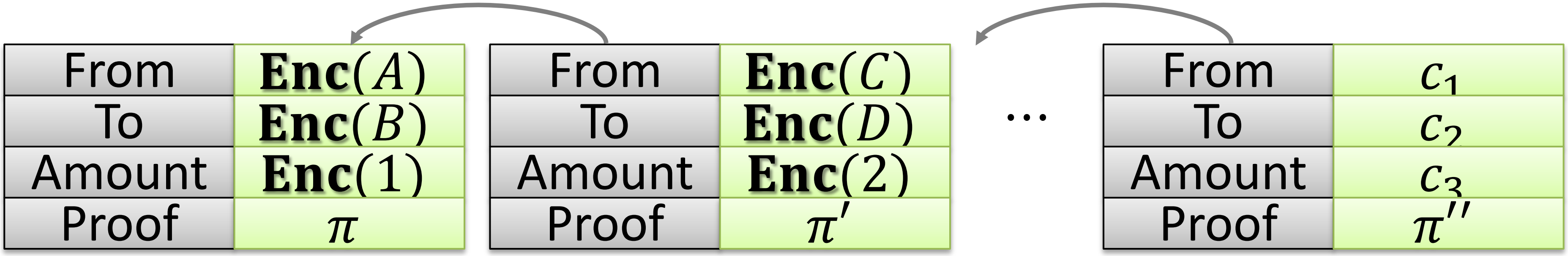
From	$\mathbf{Enc}(B)$
To	$\mathbf{Enc}(E)$
Amount	$\mathbf{Enc}(1)$

ZCash: Divisible Anonymous Payments

- A **privacy-preserving** cryptocurrency
  - – Can sit on top of Bitcoin or similar systems
- Main feature: Transactions reveal neither the **origin, destination, or amount**



Basic Intuition for ZCash

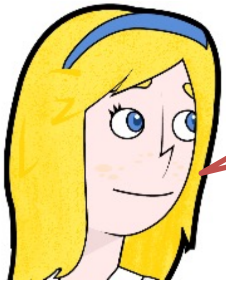


From	$\mathbf{Enc}(A)$
To	$\mathbf{Enc}(B)$
Amount	$\mathbf{Enc}(1)$
Proof	π

From	$\mathbf{Enc}(C)$
To	$\mathbf{Enc}(D)$
Amount	$\mathbf{Enc}(2)$
Proof	π'

…

From	c_1
To	c_2
Amount	c_3
Proof	π''

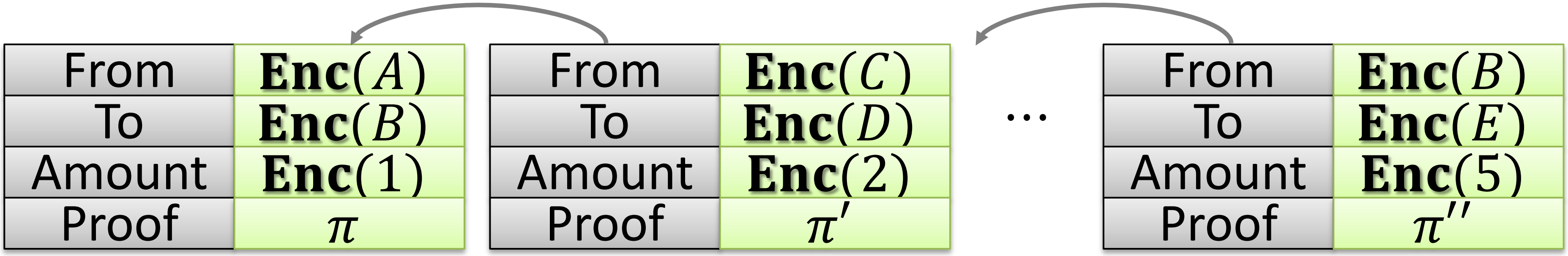


I am publishing ciphertexts c_1, c_2, c_3 which contain a **sender address, a receiver address, and a transfer amount**. Moreover the amount transferred **has not been double spent**. Here is a cryptographic **proof** $\pi''$of this fact!

Q1: What kind of proof?

Q2: What is the statement being proven?

In SNARKs We Trust



From	$\mathbf{Enc}(A)$
To	$\mathbf{Enc}(B)$
Amount	$\mathbf{Enc}(1)$
Proof	π

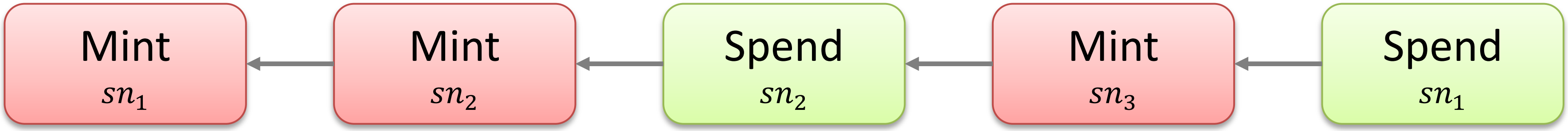
From	$\mathbf{Enc}(C)$
To	$\mathbf{Enc}(D)$
Amount	$\mathbf{Enc}(2)$
Proof	π'

$\cdots$

From	$\mathbf{Enc}(B)$
To	$\mathbf{Enc}(E)$
Amount	$\mathbf{Enc}(5)$
Proof	π''

- What type of proof?
  - **Argument** (true statements have proofs, false statements have not)
  - **Non-interactive** (need to write it down)
  - **Zero-knowledge** (reveals nothing beyond validity)
  - **Of knowledge** (technical)
  - **Succinct** (short proofs, cheap to verify)

Attempt #1: Plain Serial Numbers



Transaction types:

Mint sn — **Consume** 1 BTC to create a value-1 coin w/ serial number sn

Spend sn — **Consume** the coin w/ serial number sn

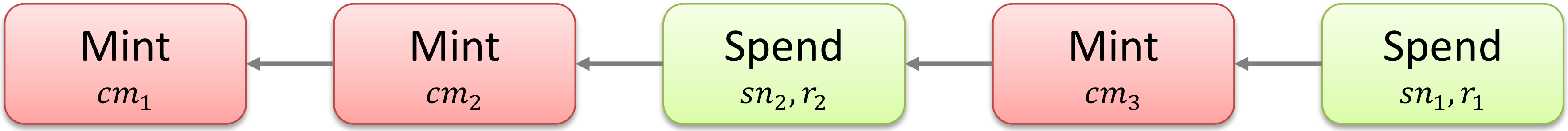
Coin

sn

Attempt #1: Plain Serial Numbers

- Good
  - **Cannot** double spend
- Bad
  - Anyone can **spend my coins**
  - Spend **linkable** to its mint
  - **Fixed** denomination
  - **Does not hide** the sender and the receiver

Attempt #2: Committed Serial Numbers



Transaction types:

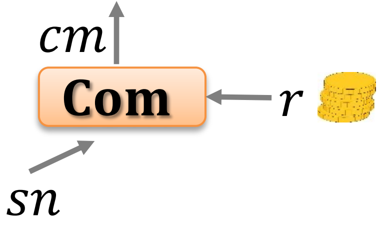
Mint cm

Spend sn, r

Consume 1 BTC to create a value-1 coin w/ comm. cm

Consume the coin w/ serial number sn

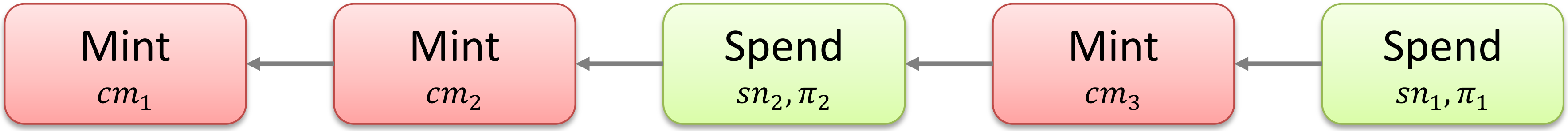
Coin



Attempt #2: Committed Serial Numbers

- Good
  - **Cannot** double spend
  - Others **cannot** spend my coins
- Bad
  - Spend **linkable** to its mint
  - **Fixed** denomination
  - **Does not hide** the sender and the receiver

Attempt #3: ZK-PoK of Commitment



Transaction types:

Mint cm — **Consume** 1 BTC to create a value-1 coin w/ comm. cm

Spend sn, π — **Consume** the coin w/ serial number sn

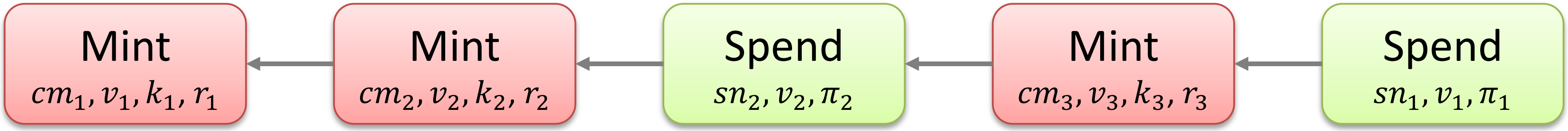
Here is a proof π that I know secret r:

- **(exists)** $cm \in$ "list of previous commitments"
- **(well-formed)** $cm = \mathbf{Com}(sn; r)$

Attempt #3: ZK-PoK of Commitment

- Good
  - **Cannot** double spend
  - Others **cannot** spend my coins
  - Spend and mint **unlinkable**
- Bad
  - **Fixed** denomination
  - Hides **only** the sender

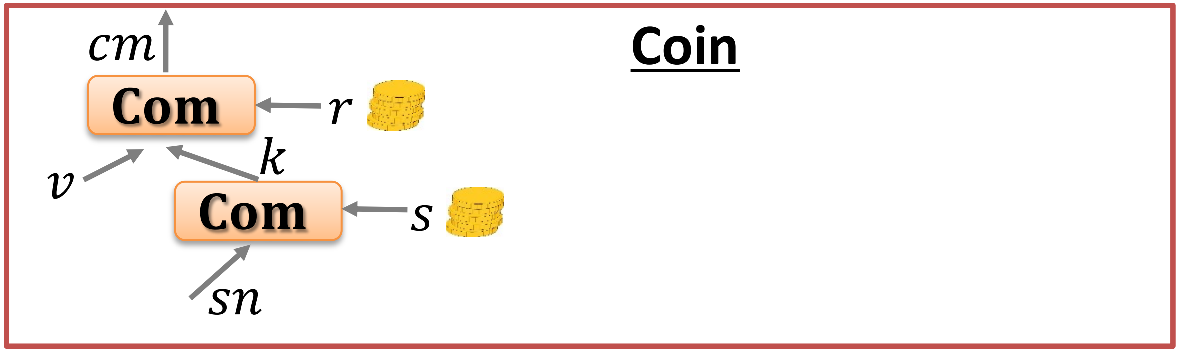
Attempt #4: Variable Denomination



Transaction types:

Mint cm, v, k, r	**Consume** v BTC to create a value-v coin w/ comm. cm
Spend sn, v, π	**Consume** the value-v coin w/ serial number sn Here is a proof π that I know secret (cm, k, r, s): • **(exists)** $cm \in$ "list of previous commitments" • **(well-formed)** $cm = \mathbf{Com}(v, k; r)$; $k = \mathbf{Com}(sn; s)$

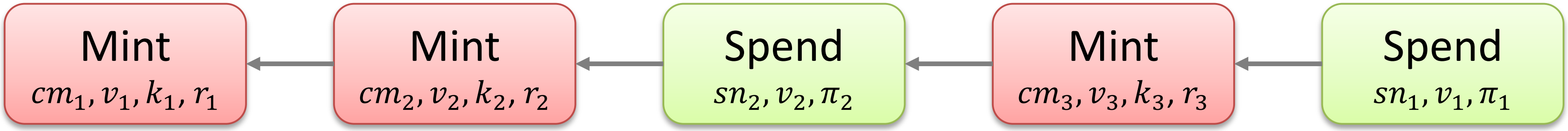
Coin



Attempt #4: Variable Denomination

- Good
  - **Cannot** double spend
  - Others **cannot** spend my coins
  - Spend and mint **unlinkable**
  - **Variable** denomination
- Bad
  - Hides **only** the sender

Attempt #5: Payment Addresses



Mint cm_1, v_1, k_1, r_1 ← Mint cm_2, v_2, k_2, r_2 ← Spend sn_2, v_2, π_2 ← Mint cm_3, v_3, k_3, r_3 ← Spend sn_1, v_1, π_1

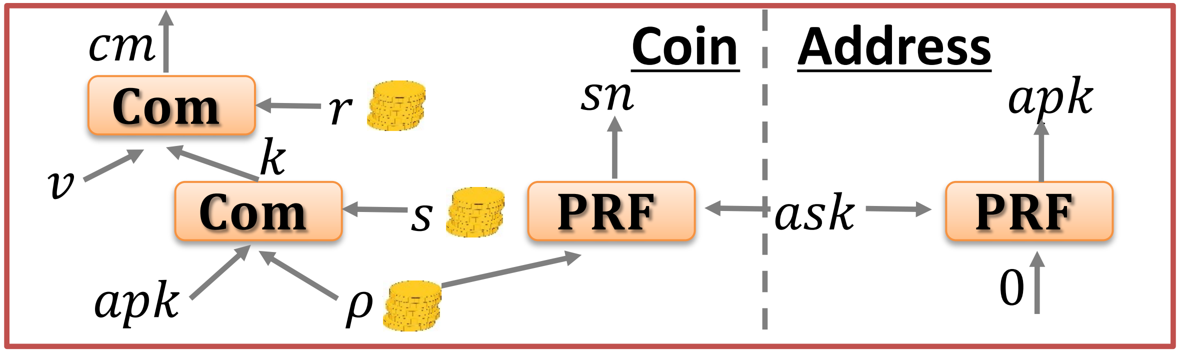
Transaction types:

Mint cm, v, k, r — **Consume** v BTC to create a value-v coin w/ comm. cm

Spend sn, v, π — **Consume** the value-v coin w/ serial number sn

Here is a proof π that I know secret $(cm, k, r, s, \rho, apk, ask)$:

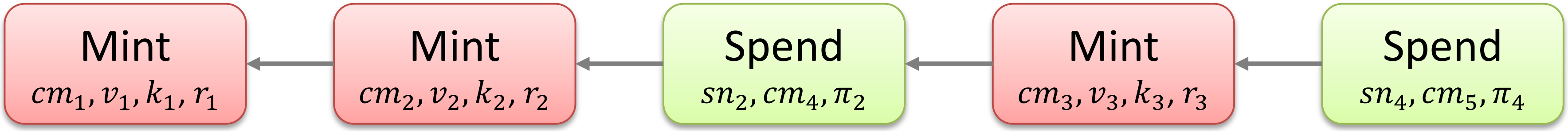
- **(exists)** $cm \in$ "list of previous commitments"
- **(well-formed)** $cm = \mathbf{Com}(v, k; r)$; $k = \mathbf{Com}(apk, \rho; s)$
- **(mine)** $sn = \mathbf{PRF}(ask, \rho)$; $apk = \mathbf{PRF}(ask, 0)$



Attempt #5: Payment Address

- Good
  - **Cannot** double spend
  - Others **cannot** spend my coins
  - Spend and mint **unlinkable**
  - **Variable** denomination
- Bad
  - Still hides **only** the sender

Attempt #6: Direct Payments



Mint cm_1, v_1, k_1, r_1

Mint cm_2, v_2, k_2, r_2

Spend sn_2, cm_4, π_2

Mint cm_3, v_3, k_3, r_3

Spend sn_4, cm_5, π_4

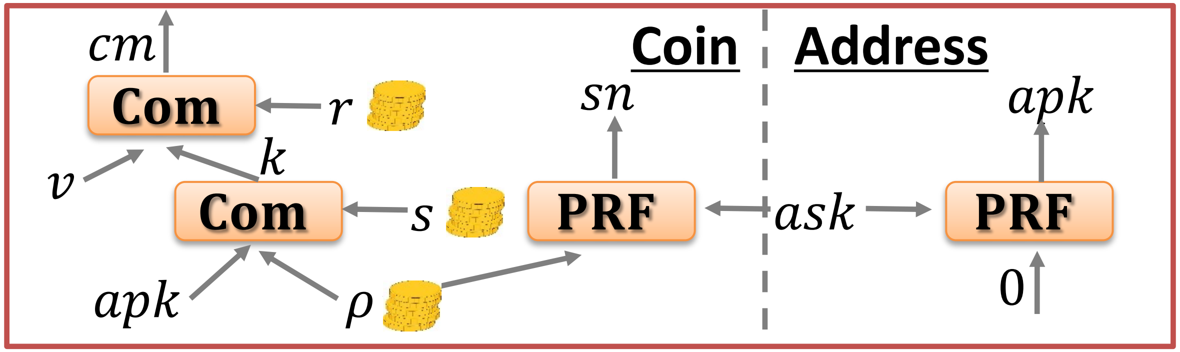
Transaction types:

Mint cm, v, k, r — **Consume** v BTC to create a value-v coin w/ comm. cm

Spend sn^A, cm^B, π — **Consume** coin w/ serial number sn^A; create coin w/ comm. cm^B

Here is a proof π that I know secret $(cm^A, v^A, k^A, r^A, s^A, \rho^A, apk^A, ask^A)$:

- **(exists)** $cm^A \in$ "list of previous commitments"
- **(well-formed)** $cm^A = \mathbf{Com}(v^A, k^A; r^A)$; $k^A = \mathbf{Com}(apk^A, \rho^A; s^A)$
- **(mine)** $sn = \mathbf{PRF}(ask^A, \rho^A)$; $apk^A = \mathbf{PRF}(ask^A, 0)$
- **(well-formed)** $cm^B = \mathbf{Com}(v^B, k^B; r^B)$; $k^B = \mathbf{Com}(apk^B, \rho^B; s^B)$
- **(same value)** $v^A = v^B$



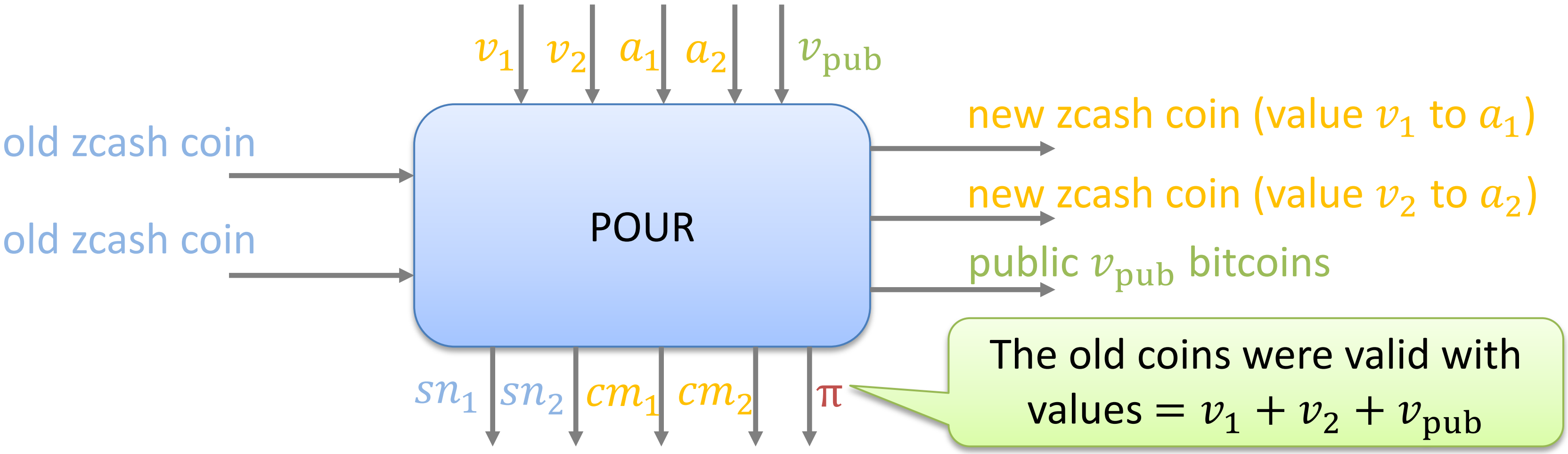
$(cm^B, v^B, k^B, r^B, s^B,$
$\rho^B, apk^B)$ sent **out of band**
or via blockchain

Attempt #6: Direct Payments

- Good
  - **Cannot** double spend
  - Others **cannot** spend my coins
  - Spend and mint **unlinkable**
  - **Variable** denomination
  - Hides **sender, receiver, and amount**

Additional Features

- POUR transactions
  - **Single type** of transaction for sending payments, making change, exchanging into bitcoins,…



The old coins were valid with values $= v_1 + v_2 + v_{\text{pub}}$

Decentralized Anonymous Payments

- A **standalone** cryptographic primitive
- Security
  - **Ledger indistinguishability:** Nothing revealed besides public information, even by chosen-transaction adversary
  - **Balance:** Can’t own more money than received or minted
  - **Transactions non-malleability:** Cannot manipulate transactions en-route to the ledger

ZCash Performances

- Efficiency
  - Size of proofs **288 bytes** (at **128 bits of security**)
  - Proof verification/creation is **< 6 ms/1min**
  - System parameters size **869 MB** (once and for all)
- Parameter generation **must be trusted**
- Crypto **assumptions**
  - Elliptic curves with pairings
  - Knowledge of exponent assumptions
  - SHA256, encryption, and signatures

Other Applications to Bitcoin

- **Lightweight** clients
  - Proof of transaction validity (verification only w.r.t. blockchain head)
  - Compressing the blockchain (e.g., only keeping unspent transactions)
- **Turing-complete** scripts/contracts with cheap verification
- … and much more (see Bitcoin forum)