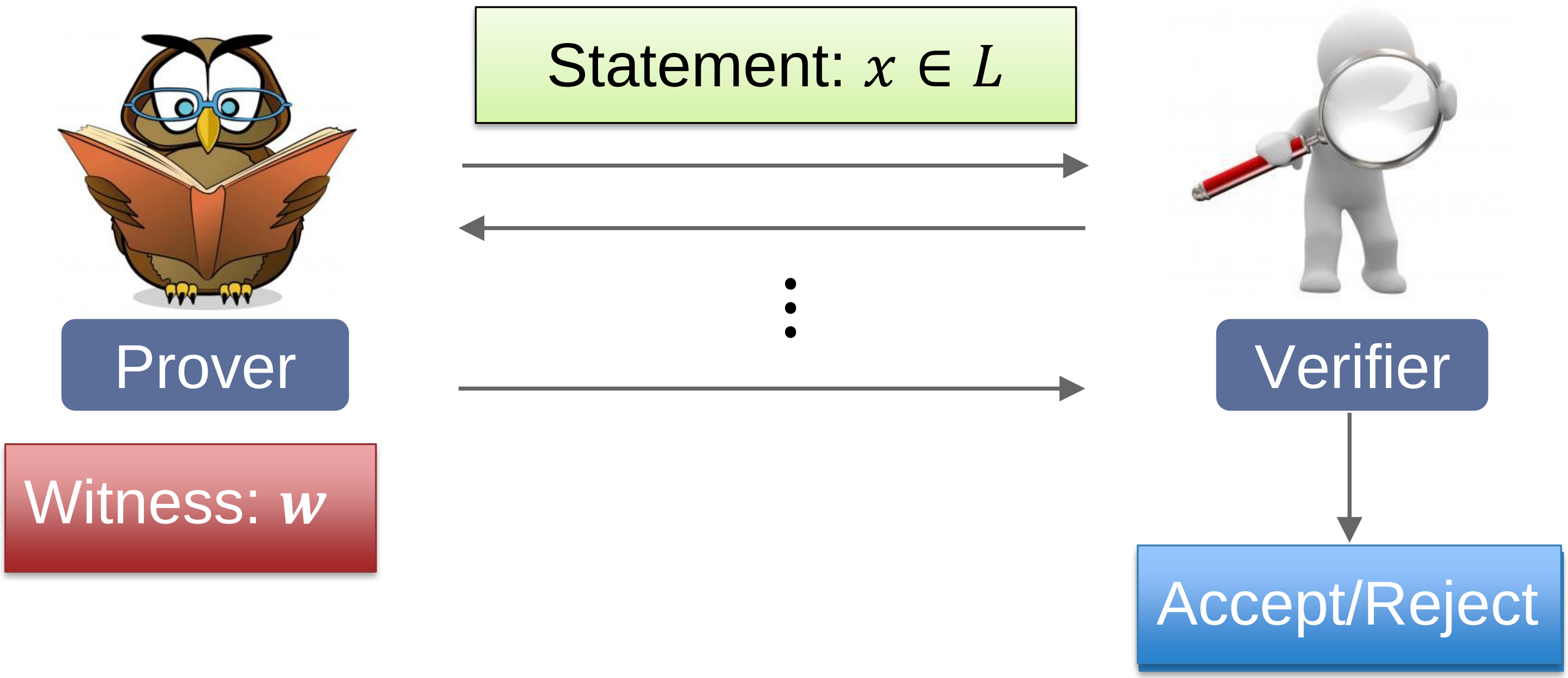
Indistinguishable Proofs of Work or Knowledge

Foteini Baldimtsi, Aggelos Kiayias,
Thomas Zacharias, Bingsheng Zhang

ASIACRYPT 2016
8th December, Hanoi, Vietnam

Motivation

(ZK) Proofs of Knowledge - PoK



1) **Completeness:** the verifier always accepts a valid proof
2) **PoK:** for any convincing verifier, we can extract w
3) **Prover privacy** is preserved via some ZK variant

Schnorr Identification – PoK of DLog

Parameters: g, q

Statement: $\exists sk: pk = g^{sk}$

Prover

Verifier

Witness: sk

pick $t \in Z_q$

$a = gt$

$\xrightarrow{\quad a \quad}$

$\xleftarrow{\quad c \quad}$

pick $c \in Zq$

$r = t + c \cdot sk$

$\xrightarrow{\quad r \quad}$

Check if

$g^r = a \cdot (pk)^c$

Schnorr Identification – PoK of DLog

Parameters: g, q

Statement: $\exists sk : pk = g^{sk}$

Prover

Verifier

Witness: sk

Schnorr identification is a Sigma protocol that achieves **special soundness** and **honest-verifier ZK**

Some motivating thoughts…

- PoK of **DLog** convinces us that the prover actually has the witness.

Some motivating thoughts…

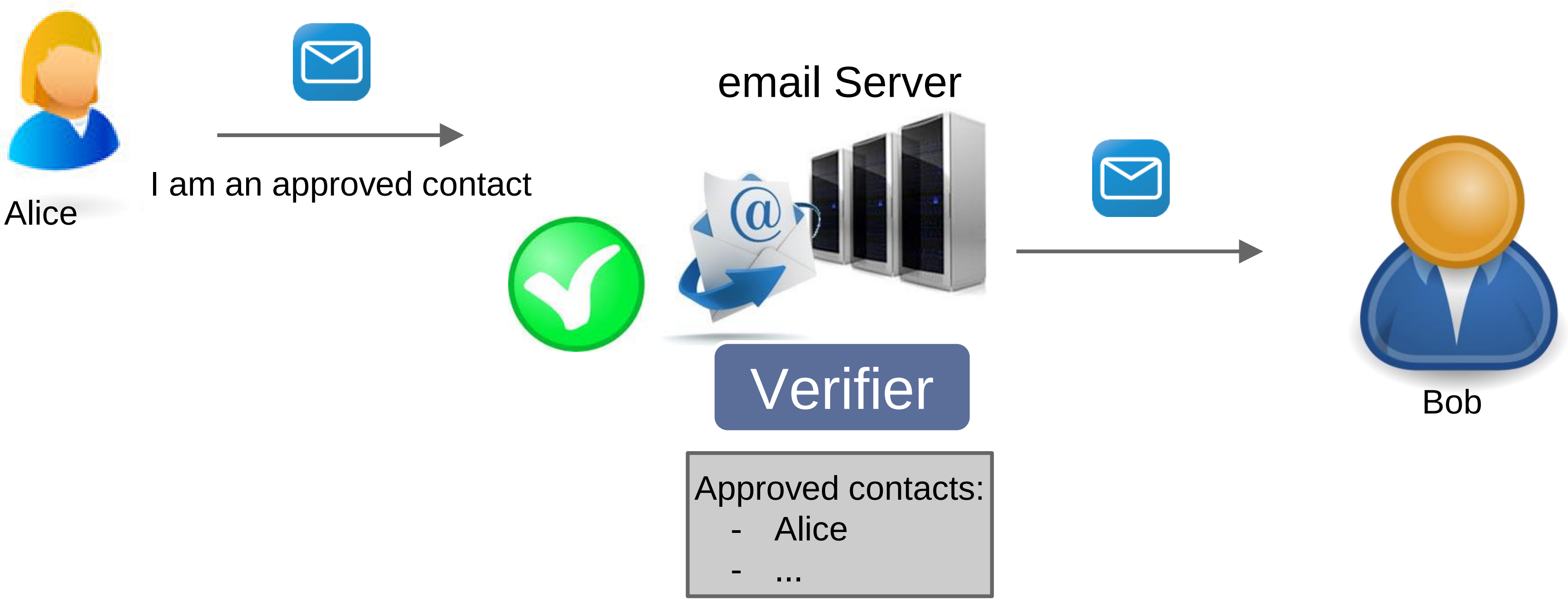
- PoK of **DLog** convinces us that the prover actually has the witness.
- But how did the prover manage to convince us?
  - Did it run efficiently because it had **knowledge** of the witness **OR**
  - Did it **work** for a (superpolynomial) amount of a time to solve the given **DLog** problem?

Reducing Spam

"If I don't know you and you want to send me a message, then you must prove that you spent, say, ten seconds of CPU time, just for me and just for this message" [DN92]

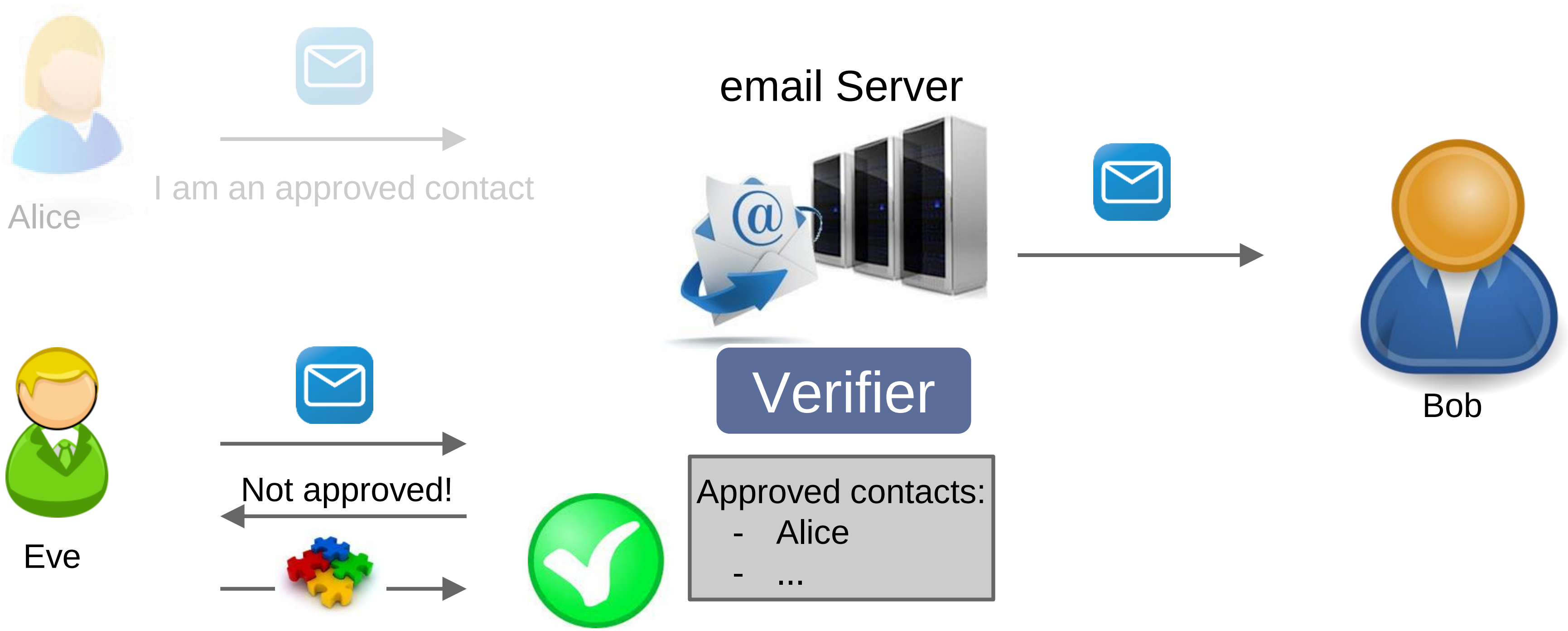
Reducing Spam

"If I don't know you and you want to send me a message, then you must prove that you spent, say, ten seconds of CPU time, just for me and just for this message" [DN92]



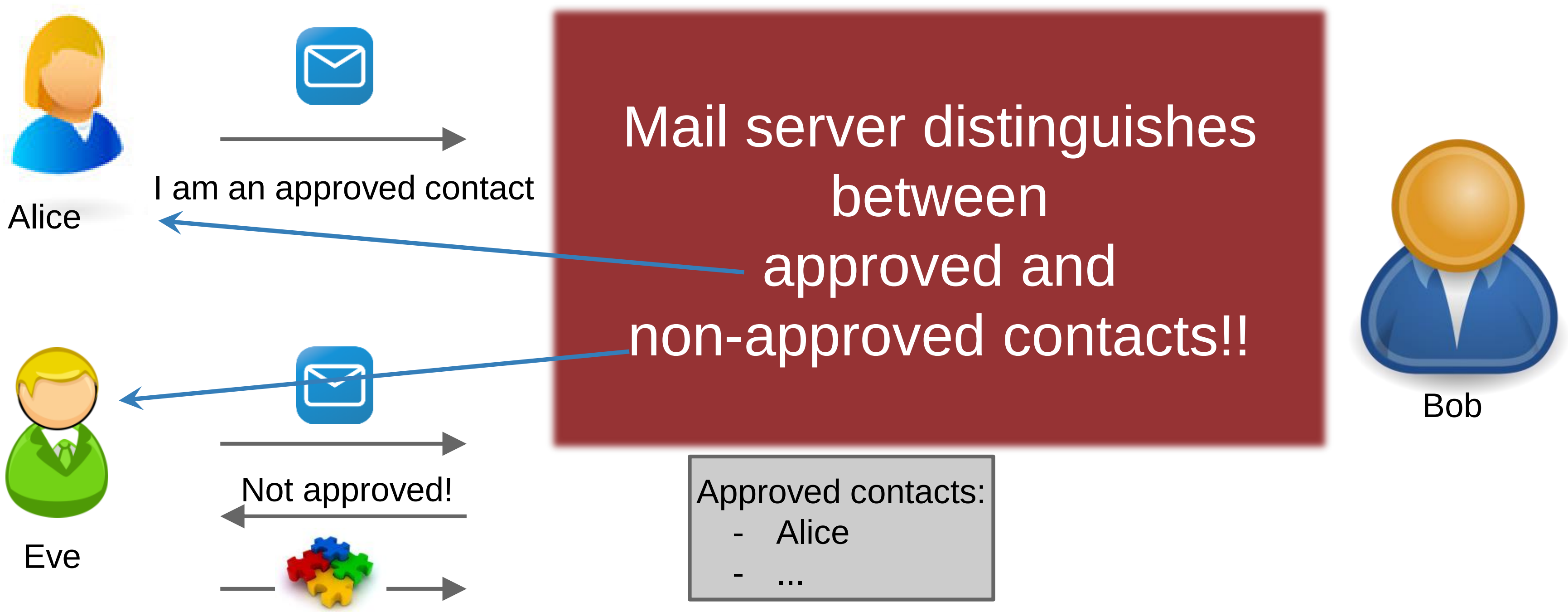
Reducing Spam

"If I don't know you and you want to send me a message, then you must prove that you spent, say, ten seconds of CPU time, just for me and just for this message" [DN92]



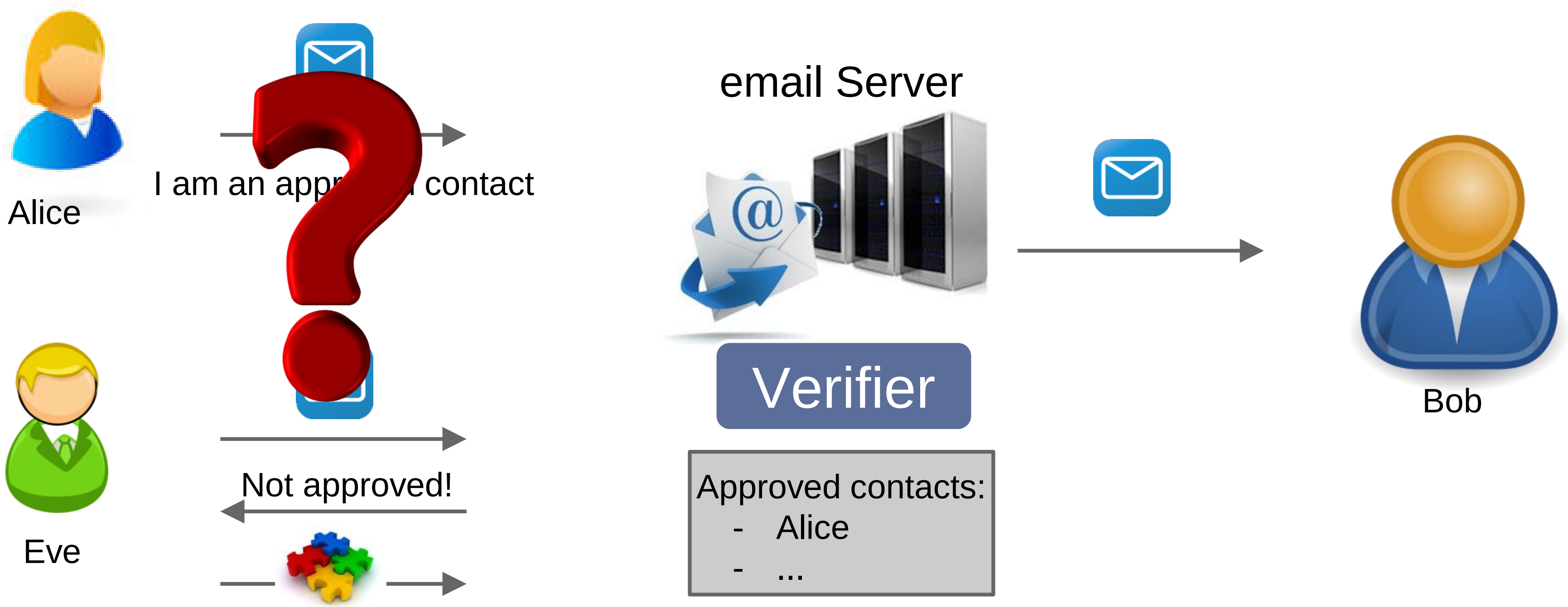
Reducing Spam

"If I don't know you and you want to send me a message, then you must prove that you spent, say, ten seconds of CPU time, just for me and just for this message" [DN92]



Reducing Spam

Where Email approval is done in a privacy-preserving manner!



Reducing spam in a privacy-preserving way

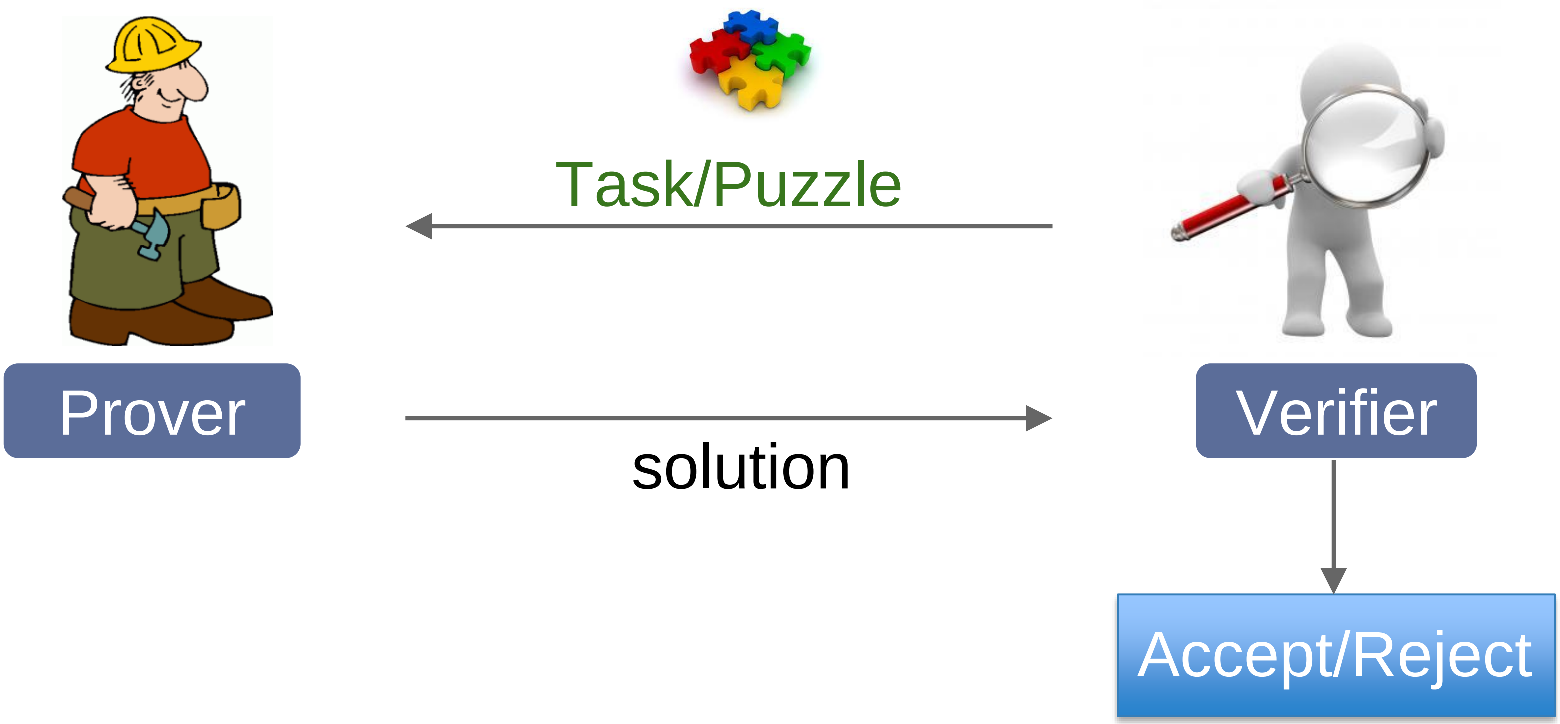
1. For senders to have access, they must **prove** that either
   - **know** some secret that implies their relation with the receiver **OR**
   - has spent a certain amount of **work** in terms of computational resources.

Reducing spam in a privacy-preserving way

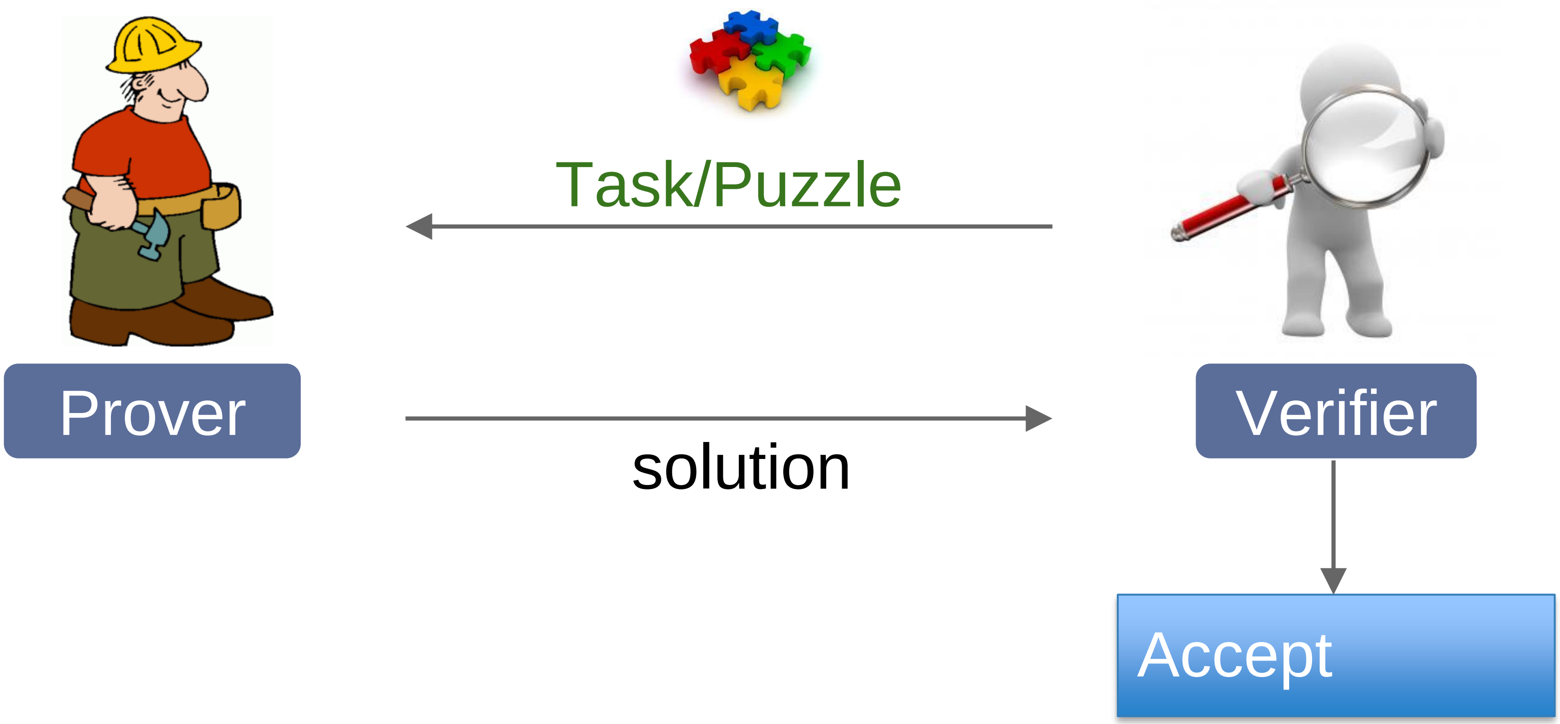
1. For senders to have access, they must **prove** that either
   - **know** some secret that implies their relation with the receiver **OR**
   - has spent a certain amount of **work** in terms of computational resources.

2. The **prover's mode** that provided access to the sender, **remains unknown** to the mail server.

Proofs of Work - PoW



Proofs of Work - PoW



The verifier ascertains that the prover performed some certain amount of work, given the difficulty of the puzzle parameters

Proofs of Work or Knowledge (PoWorKs)

Statement: $x \in L$

PoK:

Prover

PoW:

Prover

Verifier

Prover either knows a witness to the statement or performed work to solve a puzzle

Indistinguishable
Proofs of Work or Knowledge (PoWorKs)

Statement: $x \in L$

PoK:

Prover

PoW:

Prover

Verifier

Prover either knows a witness to the statement or performed work to solve a puzzle

Our contributions

Our contributions

- We define **cryptographic puzzle systems**.

Our contributions

- We define cryptographic puzzle systems.
- We define **PoWorKs** w.r.t. some language in NP and a fixed puzzle system.

Our contributions

- We define cryptographic puzzle systems.
- We define PoWorKs w.r.t. some language in NP and a fixed puzzle system.
- We provide an efficient **3-move PoWorK construction**.

Our contributions

- We define cryptographic puzzle systems.
- We define PoWorKs w.r.t. some language in NP and a fixed puzzle system.
- We provide an efficient 3-move PoWorK construction.
- We provide **two puzzle system instantiations** (one in the RO model and one under complexity assumptions).

Our contributions

- We define cryptographic puzzle systems.
- We define PoWorKs w.r.t. some language in NP and a fixed puzzle system.
- We provide an efficient 3-move PoWorK construction.
- We provide two puzzle system instantiations (one in the RO model and one under complexity assumptions).
- We present **applications of PoWorKs** in
  1. Privacy-preserving **reduce spam**.
  2. Robustness in **cryptocurrencies**.
  3. 3-round **concurrently simulatable** arguments of knowledge.

Cryptographic puzzles

Cryptographic Puzzles

Basic properties:

1) Easy to generate and efficiently sampleable
2) Hard to solve
3) Easy to verify
4) Amortization resistant

Cryptographic Puzzles

Basic properties:

1) **Easy to generate and efficiently sampleable**
2) **Hard to solve**
3) **Easy to verify**
4) **Amortization resistant**
5) **Dense** (can be sampled by just generating random strings)

Cryptographic Puzzles

We do not restrict parallelizability of our puzzles!

Cryptographic Puzzles

Puzzle Space $\boldsymbol{PS}$, Solution Space $\boldsymbol{SS}$, Hardness space $\boldsymbol{HS}$

PuzSys = {Sample, Solve , , Verify}

hardness parameter

- **Sample** $(\boldsymbol{h}) \;->\; \boldsymbol{puz} \in \boldsymbol{PS}$
- **Solve** $(\boldsymbol{h}, \boldsymbol{puz}) \;->\; \boldsymbol{soln} \in \boldsymbol{SP}$
- **Verify**$(\boldsymbol{h}, \boldsymbol{puz}, \boldsymbol{soln}) \;->\; true/false$

Dense Cryptographic Puzzles

Puzzle Space $\boldsymbol{PS}$, Solution Space $\boldsymbol{SS}$, Hardness space $\boldsymbol{HS}$

PuzSys = {Sample, Solve , SampleSol, Verify}

hardness parameter

- **Sample** $(\boldsymbol{h}) \; -> \; \boldsymbol{puz} \in \boldsymbol{PS}$
- **Solve** $(\boldsymbol{h}, \boldsymbol{puz}) \; -> \; \boldsymbol{soln} \in \boldsymbol{SP}$
- **SampleSol**$(\boldsymbol{h}) \; -> \; (\boldsymbol{puz}, \boldsymbol{soln})$
- **Verify**$(\boldsymbol{h}, \boldsymbol{puz}, \boldsymbol{soln}) \; -> \; true/false$

Cryptographic Puzzles Security

PuzSys = {Sample, Solve, SampleSol, Verify}

1) Completeness/Correctness and Efficient Sampleability of **Sample** and **SampleSol**

Cryptographic Puzzles Security

PuzSys = {Sample, Solve , SampleSol, Verify}

1) Completeness and Efficient sampleability of **Sample** and **SampleSol**
2) g-Hardness:

Cryptographic Puzzles Security

PuzSys = {Sample, Solve , SampleSol, Verify}

1) Completeness and Efficient Sampleability of **Sample** and **SampleSol**
2) g-Hardness:

PuzSys is g-**hard**, if for every adversary:

$puz <- $ **Sample** (h) $\xrightarrow{h, puz}$

Verify $(h, puz, soln) -> true$ $\xleftarrow{soln}$

$$Time_{Adversary}(h, puz) < g\,(Time_{Solve}(h, puz))$$

With negligible probability

Cryptographic Puzzles Security

PuzSys = {Sample, Solve , SampleSol, Verify}

1) Completeness and Efficient sampleability of **Sample** and **SampleSol**
2) g-Hardness
3) Statistical indistinguishability of **Sample** and **SampleSol**

Cryptographic Puzzles Security

PuzSys = {Sample, Solve , SampleSol, Verify}

1) Completeness and Efficient sampleability of **Sample** and **SampleSol**
2) g-Hardness
3) Statistical indistinguishability of **Sample** and **SampleSol**
4) (t, k) −amortization resistance

$\boldsymbol{puz_1, \ldots, puzk} < -$ **Sample**$(\boldsymbol{h})$

$\boldsymbol{h, puz_1, \ldots, puzk}$

$\boldsymbol{soln_1, \ldots, solnk}$

for all $1 < i < k$

Verify$(\boldsymbol{h, puzi, solni}) -\!>$

$$\boldsymbol{Time_{Adversary}(h, puz)} < \boldsymbol{t}\left(\sum_{i=1}^{k} \boldsymbol{g}\ (\boldsymbol{TimeSo_{lve}(h, puzi)})\right)$$

With negligible probability

PoWorKs

PoWorK Definition

(P, V) is an **f**-sound PoWorK for $L \in \boldsymbol{NP}$ w.r.t. witness relation R_L and **PuzSys**, if it achieves the following properties:

PoWorK Definition

(P, V) is an **f**-sound PoWorK for $L \in \boldsymbol{NP}$ w.r.t. witness relation R_L and **PuzSys**, if it achieves the following properties:

1) **Completeness:** for all $\boldsymbol{x} \in L, \boldsymbol{w} \in RL(x), \boldsymbol{z} \in \{0,1\}^*, \boldsymbol{h} \in HS$

$$\Pr[< P(\boldsymbol{w}) \leftrightarrow V > (\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{h});\ V \rightarrow \text{“}accept\text{”}] = 1 - \mathrm{negl}(\lambda)\ \&$$

$$\Pr[< P^{\mathsf{Solve}(\boldsymbol{h})} \leftrightarrow V > (\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{h});\ V \rightarrow \text{“accept”}] = 1 - \mathrm{negl}(\lambda)$$

PoWorK Definition

(P, V) is an $\boldsymbol{f}$-sound PoWorK for $L \in \boldsymbol{NP}$ w.r.t. witness relation R_L and **PuzSys**, if it achieves the following properties:

1) **<u>Completeness</u>**
2) $\boldsymbol{f}$-**<u>Soundness:</u>** for all $\boldsymbol{x} \in L, \boldsymbol{y}, \boldsymbol{z} \in \{0,1\}^*, \boldsymbol{h} \in HS$ and prover $\boldsymbol{P'}$:

- $\boldsymbol{puz} \leftarrow$ **Sample**$(\boldsymbol{h})$
- $< \boldsymbol{P'}(\boldsymbol{y}) \leftrightarrow V > (\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{h})$

If V accepts **while** $Time_{\boldsymbol{P'}} \leq \boldsymbol{f}\,(Time_{\textbf{Solve}}(\boldsymbol{h}, \boldsymbol{puz}))$ then

$\exists$ PPT extractor $\boldsymbol{K}$ s.t $\boldsymbol{K}^{\boldsymbol{P'}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{h}) \in R_L(\boldsymbol{x})$

PoWorK Definition

(P,V) is an $\boldsymbol{f}$-sound PoWorK for $L \in \boldsymbol{NP}$ w.r.t. witness relation R_L and **PuzSys**, if it achieves the following properties:

1) **Completeness**
2) $\boldsymbol{f}$**-Soundness**
3) **Stat./Comp. Indistinguishability:** for all $\boldsymbol{x} \in L, \boldsymbol{w} \in R_L(x), \boldsymbol{z} \in \{0,1\}^*, \boldsymbol{h} \in HS$ and verifier $\boldsymbol{V'}$:

$$\{\textbf{view}(\boldsymbol{V'}) \leftarrow < P(\boldsymbol{w}) \leftrightarrow \boldsymbol{V'} > (\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{h})\}$$

$$\{\textbf{view}(\boldsymbol{V'}) \leftarrow < \mathrm{P}^{\textsf{Solve}(\boldsymbol{h})} \leftrightarrow \boldsymbol{V'} > (\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{h})\}$$

PoWorK construction

Trivial 4-round PoWorK construction

Prover

Parameters:
L, x, λ, h

Verifier

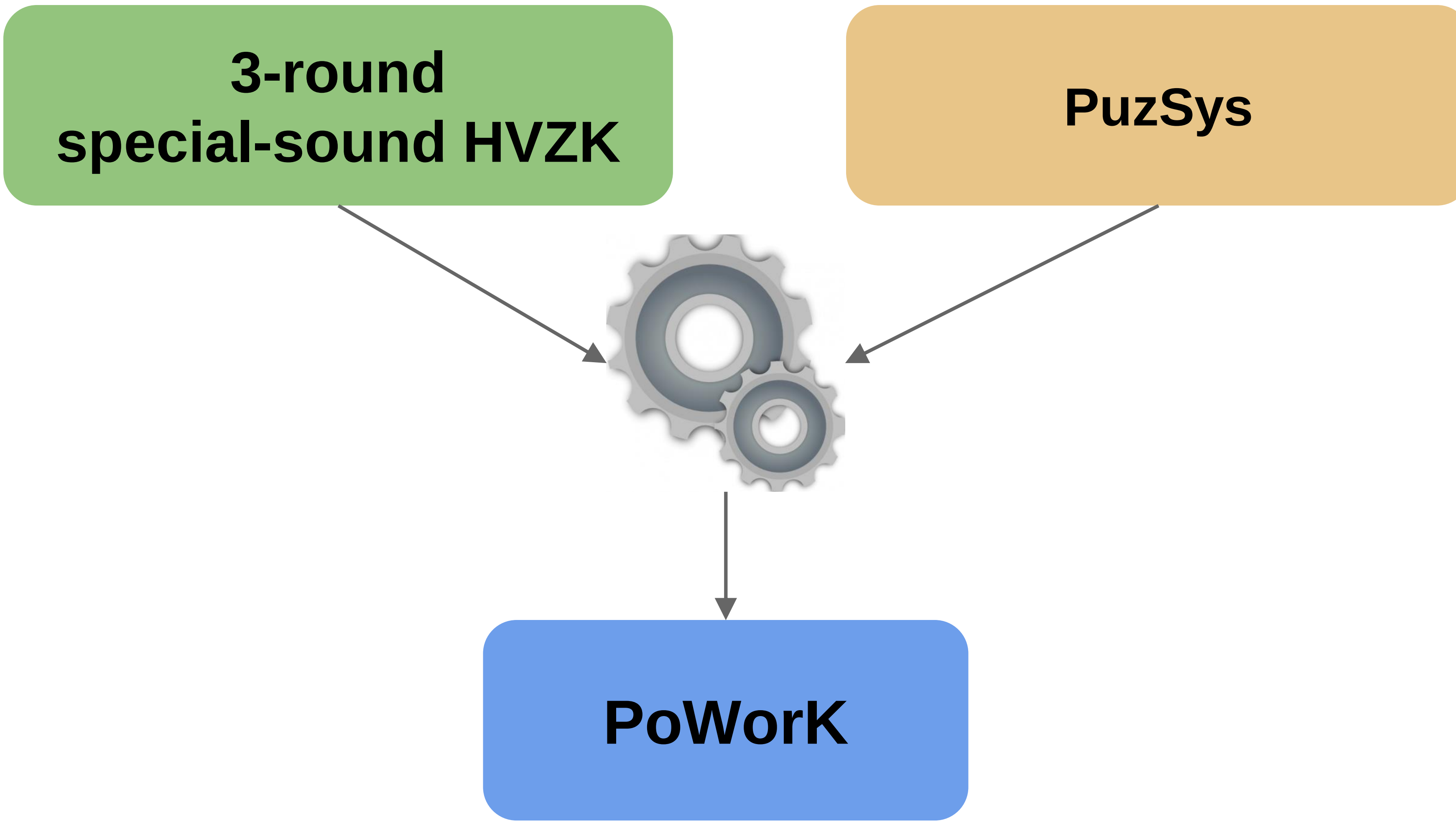
puz

pick puzzle puz

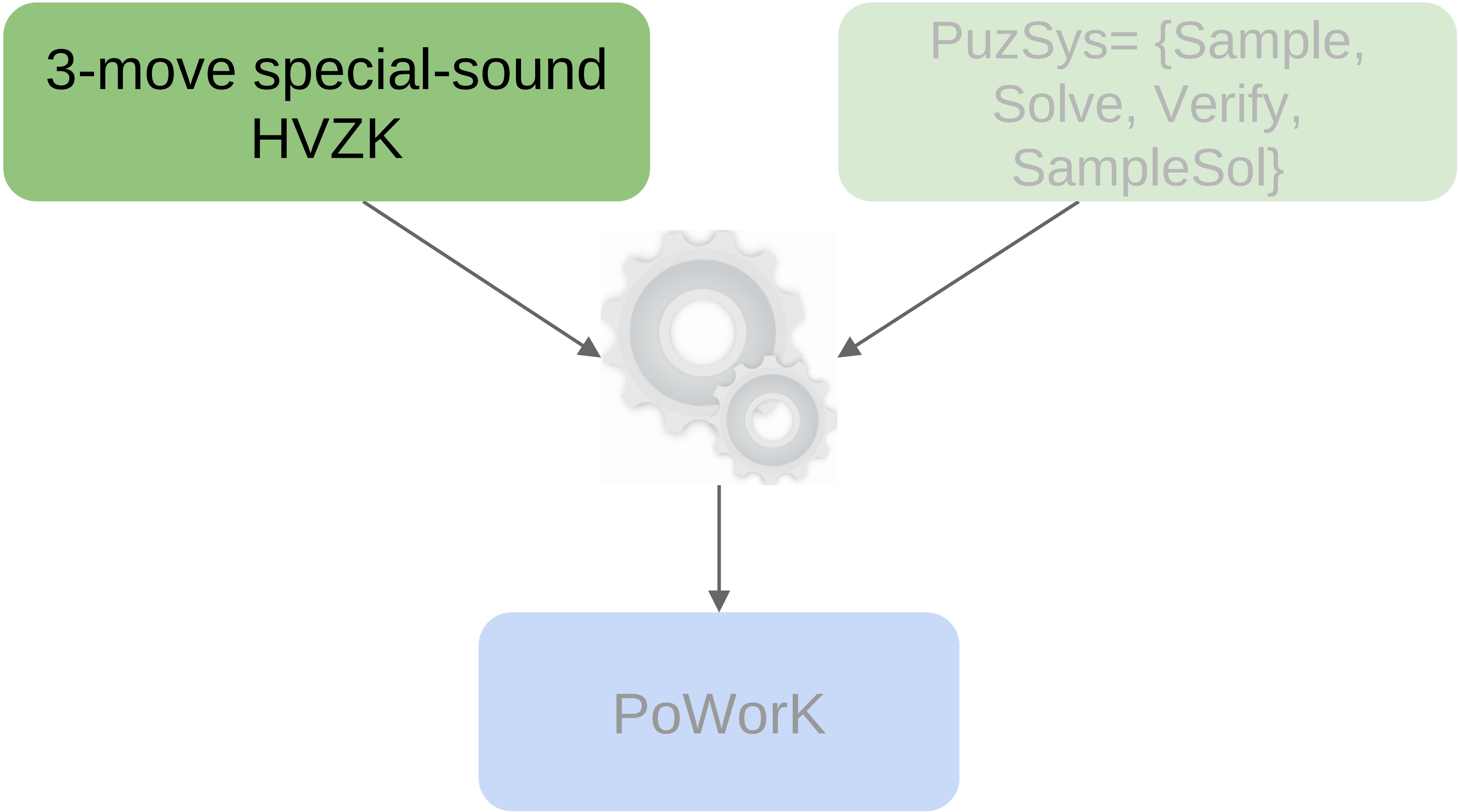
compute commitment com s.t.
com = ***Commit*** (x) **+**
ZK: know w that $x \in L$
OR
com = ***Commit*** (sol)**+**
ZK : solved puz to ***sol***

com+ZK proof

3- round PoWorK Compiler



PoWorK Compiler



3-move special-sound HVZK

Π = (P$_1$,P$_2$,Ver)

$L, RL, \boldsymbol{x}$

Prover (w) | Verifier

$(\boldsymbol{a}, \boldsymbol{u}) \leftarrow$ **P$_1$**$(\boldsymbol{w}, \boldsymbol{x})$ $\xrightarrow{\boldsymbol{a}}$

$\boldsymbol{c} \leftarrow$ ChallengeSpace

$\xleftarrow{\boldsymbol{c}}$

$\boldsymbol{r} \leftarrow$ **P$_2$**$(\boldsymbol{c}, \boldsymbol{u})$ $\xrightarrow{\boldsymbol{r}}$

0/1 $\leftarrow$ **Ver**$(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{c}, \boldsymbol{r})$

Goal: prove that $(\boldsymbol{x}, \boldsymbol{w}) \in RL$

3-move special-sound HVZK

Π = (P_1,P_2,Ver)

$L, RL, \boldsymbol{x}$

Prover (w) | Verifier

$(\boldsymbol{a}', \boldsymbol{u}') \leftarrow \mathbf{P_1}(\boldsymbol{w}, \boldsymbol{x})$ $\xrightarrow{\boldsymbol{a}'}$

$\boldsymbol{c}' \leftarrow$ ChallengeSpace

$\xleftarrow{\boldsymbol{c}'}$

$\boldsymbol{r}' \leftarrow \mathbf{P_2}(\boldsymbol{c}', \boldsymbol{u}')$ $\xrightarrow{\boldsymbol{r}'}$

$0/1 \leftarrow \mathbf{Ver}(\boldsymbol{x}, \boldsymbol{a}', \boldsymbol{c}', \boldsymbol{r}')$

- **Completeness**
- **Special Soundness:** poly-time extractor K that on input (x,a,c,r) & (x,a,c',r') outputs w s.t. (x,w) ∈ R_L
- **HVZK:** poly-time simulator Sim that on input (x) outputs an accepting (x,a,c,r) with same distribution as P on input (x,w) and honest V

PoWorK Compiler - PoK mode

$L, RL, \boldsymbol{x}, \boldsymbol{h}$

Prover ($\boldsymbol{w}$)

Verifier

PoWorK Compiler - PoK mode

$L, RL, \boldsymbol{x}, \boldsymbol{h}$

Prover ($\boldsymbol{w}$)

Verifier

$(\boldsymbol{a}', \boldsymbol{u}) \leftarrow \mathbf{P_1}(\boldsymbol{w}, \boldsymbol{x})$

$\xrightarrow{\boldsymbol{a}'}$

PoWorK Compiler - PoK mode

$L, RL, \boldsymbol{x}, \boldsymbol{h}$

Prover ($\boldsymbol{w}$)

Verifier

$(\boldsymbol{a}', \boldsymbol{u}) \leftarrow \mathbf{P_1}(\boldsymbol{w}, \boldsymbol{x})$

$\xrightarrow{\boldsymbol{a}'}$

$\boldsymbol{c} \leftarrow$ ChallengeSpace

$\xleftarrow{\boldsymbol{c}}$

PoWorK Compiler - PoK mode

$L, RL, \boldsymbol{x}, \boldsymbol{h}$

Prover ($\boldsymbol{w}$) | Verifier

$(\boldsymbol{a'}, \boldsymbol{u}) \leftarrow \mathsf{P_1}(\boldsymbol{w}, \boldsymbol{x})$

$\xrightarrow{\boldsymbol{a'}}$

$\boldsymbol{c} \leftarrow$ ChallengeSpace

$\xleftarrow{\boldsymbol{c}}$

$(\boldsymbol{puz}, \boldsymbol{soln}) \leftarrow \mathsf{SampleSol}(\boldsymbol{h})$

Set $\boldsymbol{c'} = \boldsymbol{c} \oplus \boldsymbol{puz}$

$\boldsymbol{r'} \leftarrow \mathsf{P_2}(\boldsymbol{c'}, \boldsymbol{u})$

$\xrightarrow{\boldsymbol{c'}, \boldsymbol{r'}, \boldsymbol{puz}, \boldsymbol{soln}}$

PoWorK Compiler - PoK mode

$L, RL, \boldsymbol{x}, \boldsymbol{h}$

Prover ($\boldsymbol{w}$)

Verifier

$(\boldsymbol{a'}, \boldsymbol{u}) \leftarrow \mathsf{P_1}(\boldsymbol{w}, \boldsymbol{x})$

$\xrightarrow{\boldsymbol{a'}}$

$\boldsymbol{c} \leftarrow$ ChallengeSpace

$\xleftarrow{\boldsymbol{c}}$

$(\boldsymbol{puz}, \boldsymbol{soln}) \leftarrow$ SampleSol$(\boldsymbol{h})$

Set $\boldsymbol{c'} = \boldsymbol{c} \oplus \boldsymbol{puz}$

$\boldsymbol{r'} \leftarrow \mathsf{P_2}(\boldsymbol{c'}, \boldsymbol{u})$

$\xrightarrow{\boldsymbol{c'}, \boldsymbol{r'}, \boldsymbol{puz}, \boldsymbol{soln}}$

Verification

- $\boldsymbol{c} = \boldsymbol{c'} \oplus \boldsymbol{puz}$
- Ver$(\boldsymbol{x}, \boldsymbol{a'}, \boldsymbol{c'}, \boldsymbol{r'})$
- Verify$(\boldsymbol{h}, \boldsymbol{puz}, \boldsymbol{soln})$

PoWorK Compiler - PoW mode

$L, RL, \boldsymbol{x}, \boldsymbol{h}$

Prover

Verifier

PoWorK Compiler - PoW mode

$L, RL, \boldsymbol{x}, \boldsymbol{h}$

Prover

Verifier

$(\boldsymbol{a'}, \boldsymbol{c'}, \boldsymbol{r'}) \leftarrow \textbf{Sim}(\boldsymbol{x})$

$\xrightarrow{\boldsymbol{a'}}$

PoWorK Compiler - PoW mode

$L, RL, \boldsymbol{x}, \boldsymbol{h}$

Prover

Verifier

$(\boldsymbol{a'}, \boldsymbol{c'}, \boldsymbol{r'}) \leftarrow$ **Sim**$(\boldsymbol{x})$

$\xrightarrow{\boldsymbol{a'}}$

$\boldsymbol{c} \leftarrow$ ChallengeSpace

$\xleftarrow{\boldsymbol{c}}$

PoWorK Compiler - PoW mode

$L, RL, \boldsymbol{x}, \boldsymbol{h}$

Prover

Verifier

$(\boldsymbol{a'}, \boldsymbol{c'}, \boldsymbol{r'}) \leftarrow$ **Sim**$(\boldsymbol{x})$

$\xrightarrow{\boldsymbol{a'}}$

$\boldsymbol{c} \leftarrow$ ChallengeSpace

$\xleftarrow{\boldsymbol{c}}$

Set $\boldsymbol{puz} = \boldsymbol{c} \oplus \boldsymbol{c'}$

$\boldsymbol{soln} \leftarrow$ **Solve**$(\boldsymbol{h}, \boldsymbol{puz})$

$\xrightarrow{\boldsymbol{c'}, \boldsymbol{r'}, \boldsymbol{puz}, \boldsymbol{soln}}$

PoWorK Compiler - PoW mode

$L, RL, \boldsymbol{x}, \boldsymbol{h}$

Prover

Verifier

$(\boldsymbol{a'}, \boldsymbol{c'}, \boldsymbol{r'}) \leftarrow$ **Sim**$(\boldsymbol{x})$

$\xrightarrow{\boldsymbol{a'}}$

$\boldsymbol{c} \leftarrow$ ChallengeSpace

$\xleftarrow{\boldsymbol{c}}$

Set $\boldsymbol{puz} = \boldsymbol{c} \oplus \boldsymbol{c'}$

$\boldsymbol{soln} \leftarrow$ **Solve**$(\boldsymbol{h}, \boldsymbol{puz})$

$\xrightarrow{\boldsymbol{c'}, \boldsymbol{r'}, \boldsymbol{puz}, \boldsymbol{soln}}$

<u>Verification</u>

- $\boldsymbol{c} = \boldsymbol{c'} \oplus \boldsymbol{puz}$
- **Ver**$(\boldsymbol{x}, \boldsymbol{a'}, \boldsymbol{c'}, \boldsymbol{r'})$
- **Verify**$(\boldsymbol{h}, \boldsymbol{puz}, \boldsymbol{soln})$

Security of PoWorK compiler



<u>Assumptions</u>

- Challenge and puzzle sampling distributions are statistically close
- Both distributions are (statistically) invariant to any group operation $\oplus$
- **Solve** asymptotically dominates the protocol run

Theorem:

- L language in $\boldsymbol{NP}$ with a witness relation R_L
- **Π** =(**P**$_{\mathbf{1}}$, **P**$_{\mathbf{2}}$, **Ver**) special-sound 3-move statistical HVZK for R_L
- **PuzSys** = (**Sample**, **Solve**,**SampleSol** , **Verify**) with $\boldsymbol{g}$-hardness

(P, V) is a $(\Theta(\boldsymbol{g}))$-sound PoWorK with statistical indistinguishability.

Dense Puzzle Instantiations

Dense Puzzle Instantiations

PuzSys = (Sample,SampleSol, Solve, Verify)

(1) Based on random oracles

(2) Based on complexity assumptions

Random Oracle instantiation

Assume a hash function $H: \{0,1\}^{\lambda} \to \{0,1\}^{\lambda}$

- **Sample** $(\boldsymbol{h})$: return $\boldsymbol{puz} \in \{0,1\}^{\lambda}$

- **SampleSol** $(\boldsymbol{h})$: pick $\boldsymbol{x} \in \{0,1\}^{\lambda}$ and set $\boldsymbol{puz} = \boldsymbol{LSB_h}(H(\boldsymbol{x}))$ and $\boldsymbol{soln} = \boldsymbol{x}$

- **Solve** $(\boldsymbol{puz})$: randomly pick $\boldsymbol{x}' \in \{0,1\}^{\lambda}$ and try whether
$$\boldsymbol{LSB_h}(H(\boldsymbol{x}')) = \boldsymbol{puz}$$
If yes, then output $\boldsymbol{soln} = \boldsymbol{x}'$

- **Verify** $(\boldsymbol{h}, \boldsymbol{puz}, \boldsymbol{soln})$: check whether
$$\boldsymbol{LSB_h}(H(\boldsymbol{soln})) = \boldsymbol{puz}$$

Random Oracle instantiation

Theorem:

For every $h \in [\log^2\lambda, \lambda/4], c > 2$, $k = O(\sqrt[8]{2^\lambda})$, if H is a RO, then the RO instantiation is a dense puzzle system with $\sqrt[c]{(\cdot)}$- soundness and (id, k)-amortization resistance.

DLog instantiation

- We construct target collision resistant (**TCR**) **strong extractors from** regular universal oneway hash functions (**UOWHFs**).

DLog instantiation

- We construct target collision resistant (**TCR**) **strong extractors** **from** regular universal oneway hash functions (**UOWHFs**).
- We prove that given a target TCR strong extractor $\mathbf{Ext}$, and a one-way function $\boldsymbol{f}$, we get that

$$\Psi(\boldsymbol{x}, seed) = (\mathbf{Ext}(\boldsymbol{f}(\boldsymbol{x}), seed), seed)$$

is a **dense one-way function** (i.e. its output is close to uniform)

DLog instantiation

- We construct target collision resistant (**TCR**) **strong extractors from** regular universal oneway hash functions (**UOWHFs**).
- We prove that given a target TCR strong extractor $\mathbf{Ext}$, and a one-way function $\boldsymbol{f}$, we get that

$$\Psi(\boldsymbol{x}, seed) = (\mathbf{Ext}(\boldsymbol{f}(x), seed), seed)$$

is a **dense one-way function**

- Given randomness $\boldsymbol{r}$ and hardness parameter $\boldsymbol{h}$ we set the puzzle

$$\boldsymbol{puz} = (\mathbf{Ext}(\mathbf{DLog}^{-1}(\boldsymbol{x} + \boldsymbol{r}), seed)), seed, \boldsymbol{r})$$

with solution

$$\boldsymbol{soln} = \boldsymbol{x} \in \{0,1\}^{\boldsymbol{h}}$$

DLog instantiation

Theorem:

For every $h \in [2\log^4\lambda, \log^5\lambda], c > 2$, $k = O(2^{\log^3\lambda})$, if the TCR property of **Ext** is $O(\sqrt{2^h})$ –secure and **DLog** is $O(\sqrt[c]{2^h})$ – hard, then the DLog instantiation is a dense puzzle system with $\sqrt[c]{(\cdot)}$- soundness and (id, k)-amortization resistance.

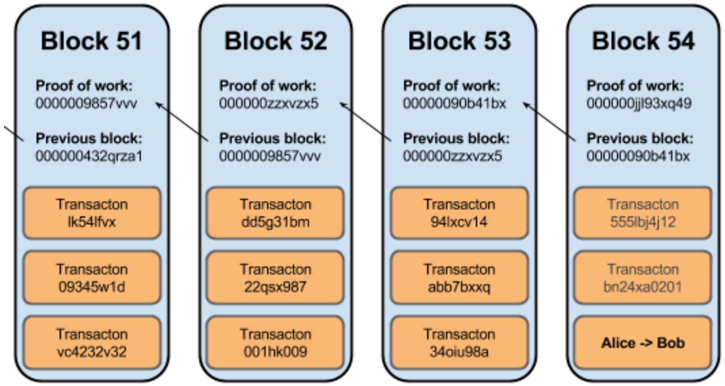
PoWorK applications

Privacy-Preserving Reducing Spam

“If I don’t know you and you want to send me a message, then you must prove that you spent, say, ten seconds of CPU time, just for me and just for this message” [DN92]

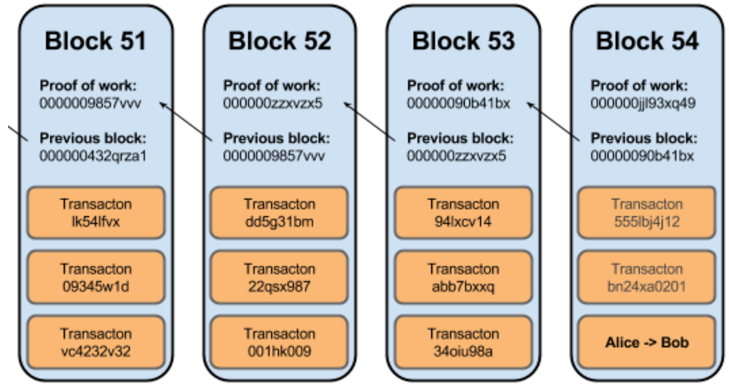
Mail server cannot distinguish between approved contacts or not

Cryptocurrencies with enhanced liveness



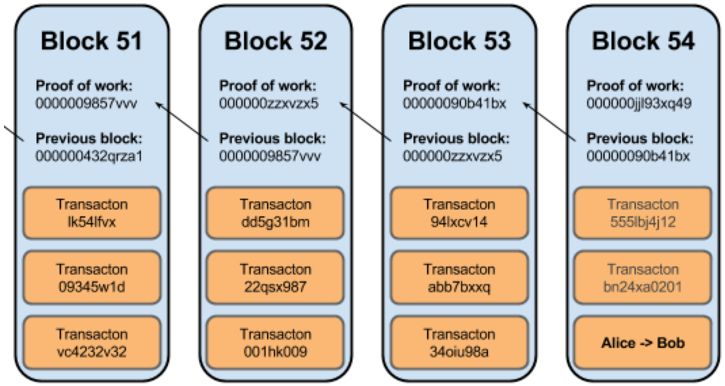
Most blockchains are maintained via proofs of work

Cryptocurrencies with enhanced liveness



But...recent suggestions exist that are based in signatures/ proofs of knowledge

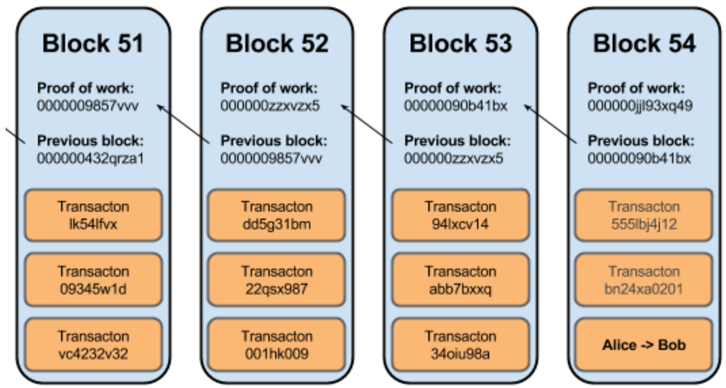
Cryptocurrencies with enhanced liveness



Hybrid PoW - PoK Cryptocurrencies

OR

Cryptocurrencies with enhanced liveness

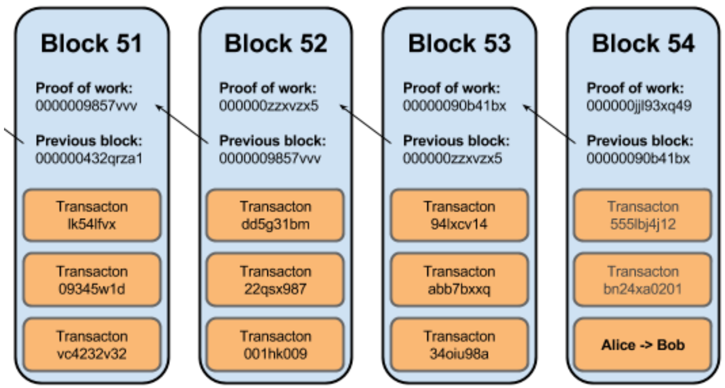


Hybrid PoW - PoK Cryptocurrencies

The ledger remains live even if many miners go offline

OR

Cryptocurrencies with enhanced liveness



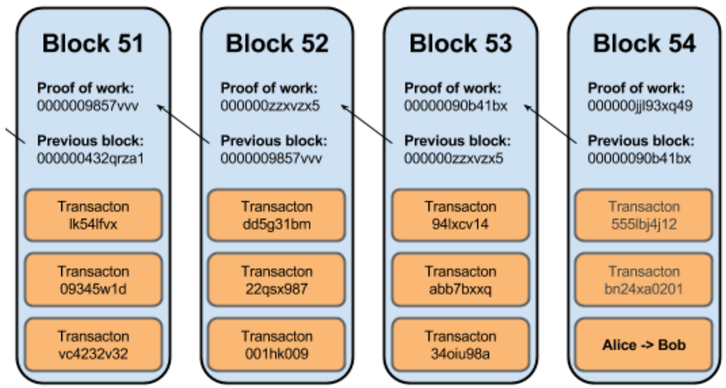
Hybrid PoW - PoK Cryptocurrencies

A trusted party could issue blocks in case of such emergency

?

OR

Cryptocurrencies with enhanced liveness



Hybrid PoW - PoK Cryptocurrencies

the trusted party's involvement will be unnoticed and hence will have no impact to the economy that the cryptocurrency supports

?

OR

3-round concurrently simulatable arguments of knowledge

- We show that under reasonable assumptions our 3-move PoWorK construction is **straight-line simulatable** in $O(\lambda^{\mathrm{poly}(\log\lambda)})$ time.
- $\lambda^{\mathrm{poly}(\log\lambda)}$ is **closed under polynomial**.
- By the results of Pass, our PoWorK construction is a **3-round concurrently simulatable argument of knowledge**.

Conclusions
and
Future Work

Conclusions

- We **define PoWorKs**, a meaningful novel class of interactive proof systems.
- We **define and instantiate** cryptographic **puzzle systems**.
- We **provide** an efficient 3-round **PoWorK construction**.
- We motivate the applicability of **PoWorK**s via real-world and theoretic **applications**.

Future directions

- Alternative **PoWorK constructions**.
- **Relation** of PoWorKs **with other complexity classes**.
- **Applications** of PoWorKs **in real-world** scenarios.
- **Puzzle system instantiations**.

 ***Thank you***!!! 

Indistinguishable Proofs of Work or Knowledge

Foteini Baldimtsi, Aggelos Kiayias,
Thomas Zacharias, Bingsheng Zhang

ASIACRYPT 2016
8th December, Hanoi, Vietnam