Eos: Efficient Private Delegation of zkSNARK provers

Alessandro Chiesa

EPFL, UC Berkeley

Ryan Lehmkuhl

MIT

Pratyush Mishra

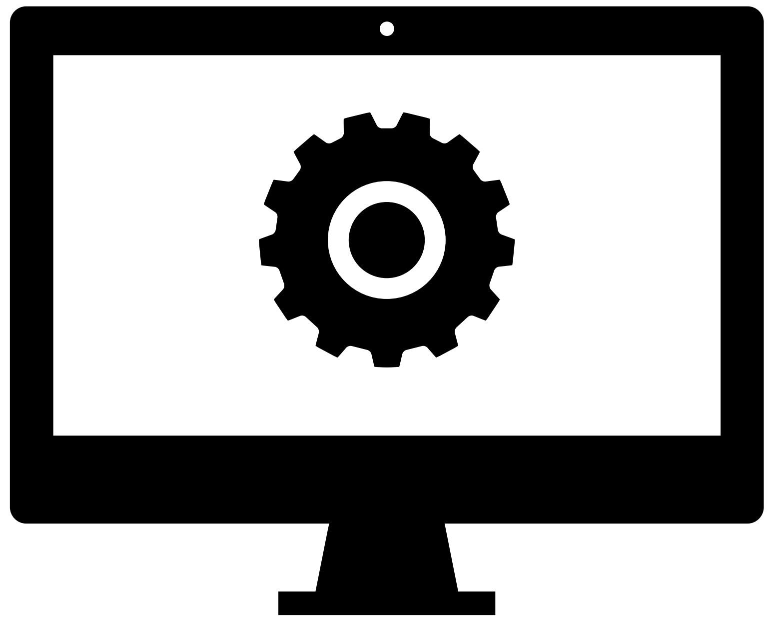
Aleo, UPenn

Yinuo Zhang

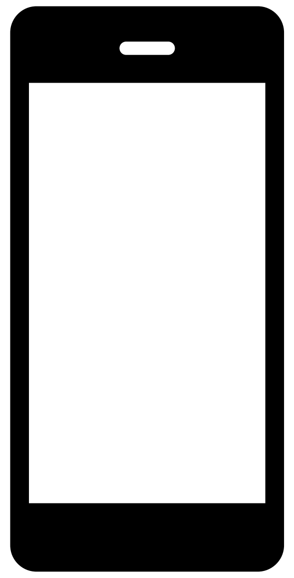
UC Berkeley

zkSNARKs [Mic94, Groth10, GGPR13, Groth16… …, GWC19, CHMMVW20, …]

Goal: Prove that a private value satisfies some property



Prover



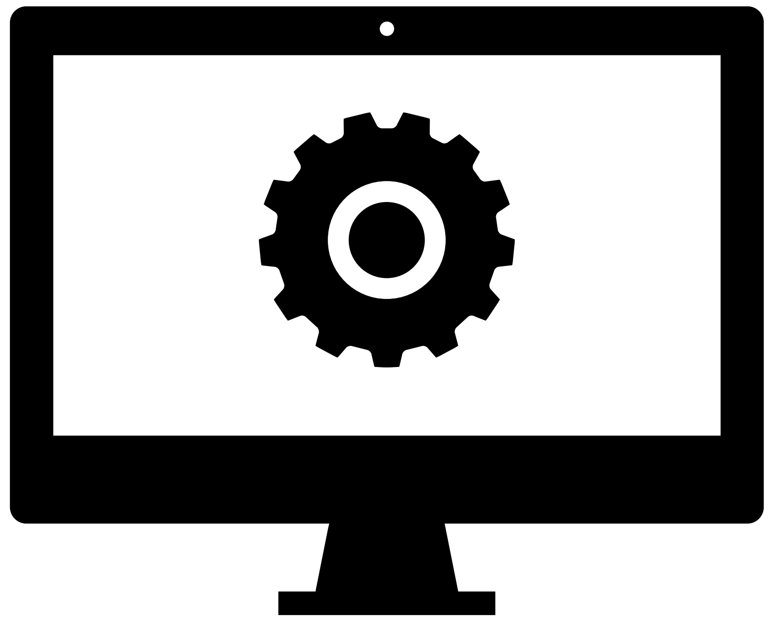
Verifier

zkSNARKs [Mic94, Groth10, GGPR13, Groth16… …, GWC19, CHMMVW20, …]

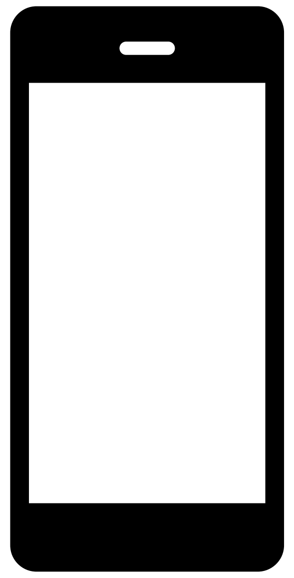
Goal: Prove that a private value satisfies some property

I know **w** s.t. SHA256(**w**) = **x**

public hash: **x**



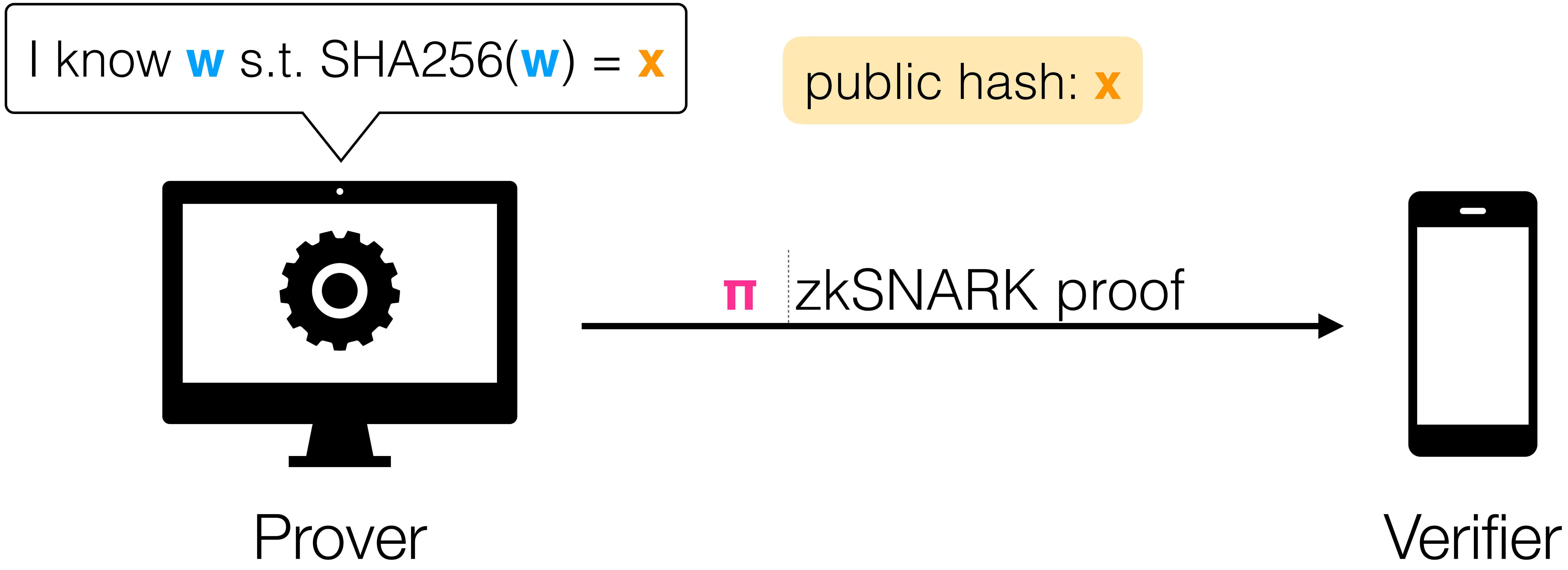
Prover



Verifier

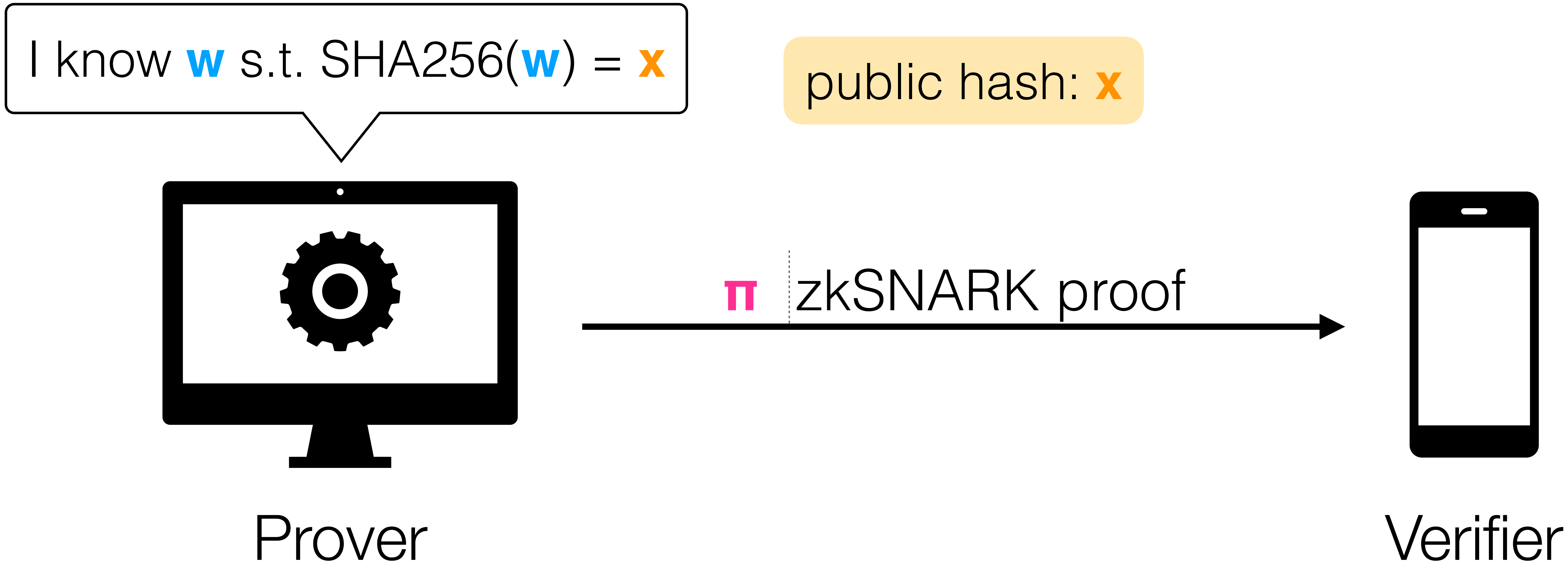
zkSNARKs [Mic94, Groth10, GGPR13, Groth16… …, GWC19, CHMMVW20, …]

Goal: Prove that a private value satisfies some property



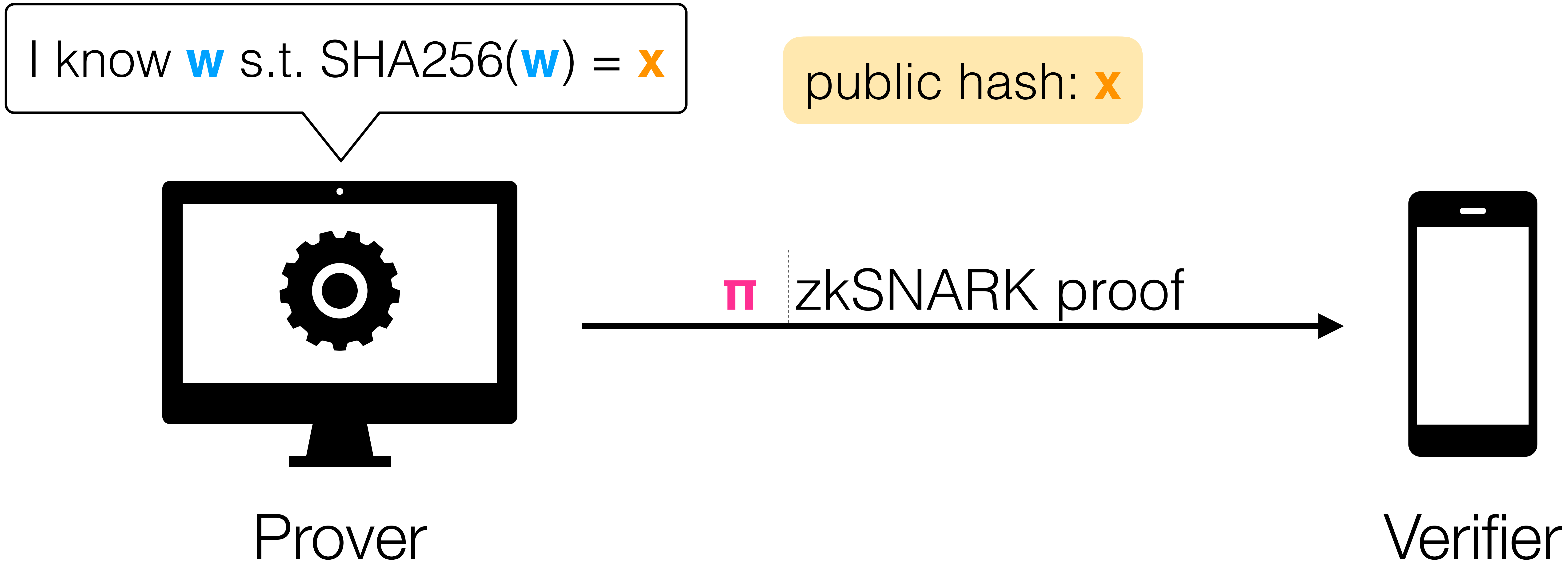
zkSNARKs [Mic94, Groth10, GGPR13, Groth16… …, GWC19, CHMMVW20, …]

Goal: Prove that a private value satisfies some property

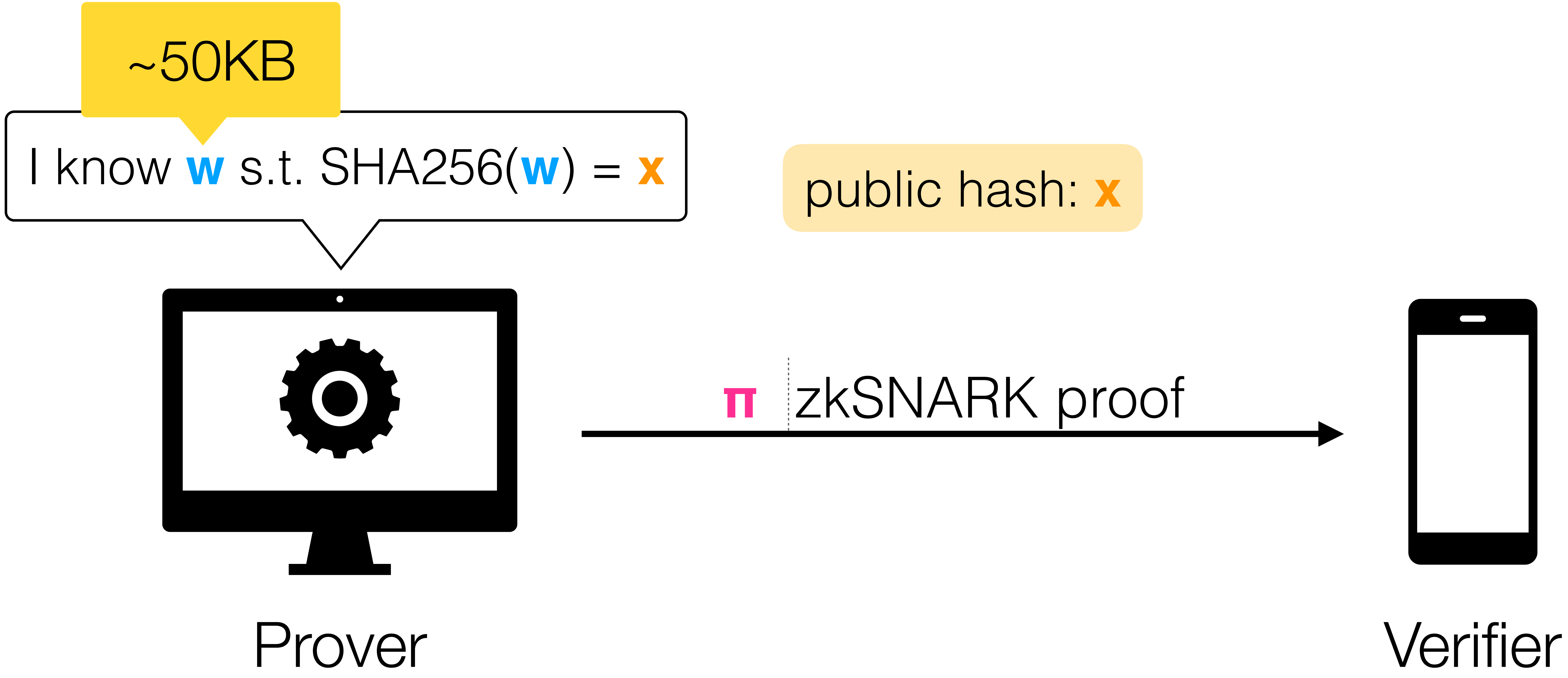


Zero Knowledge: Verifier learns nothing about **w** except that SHA256(**w**) = **x**

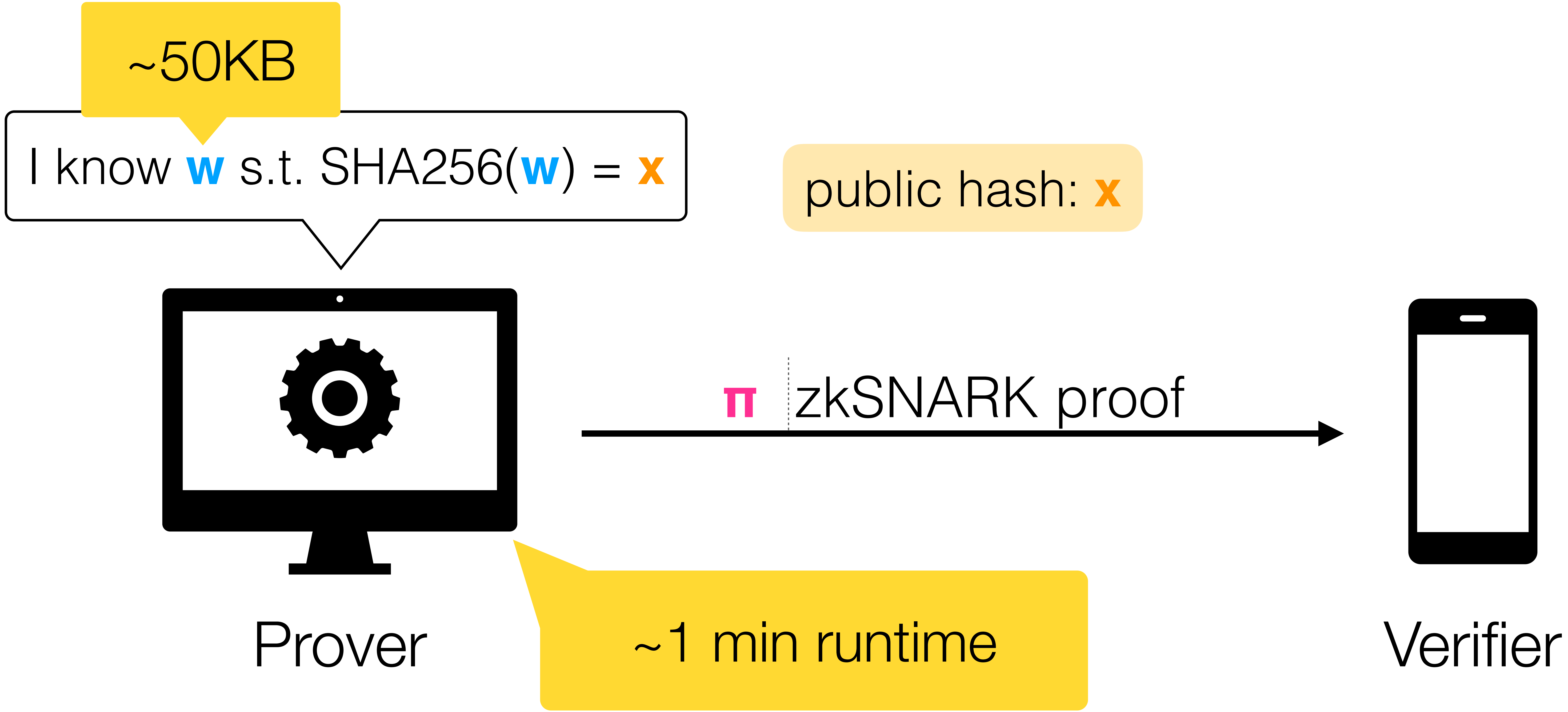
Problem: Proving is *really* slow



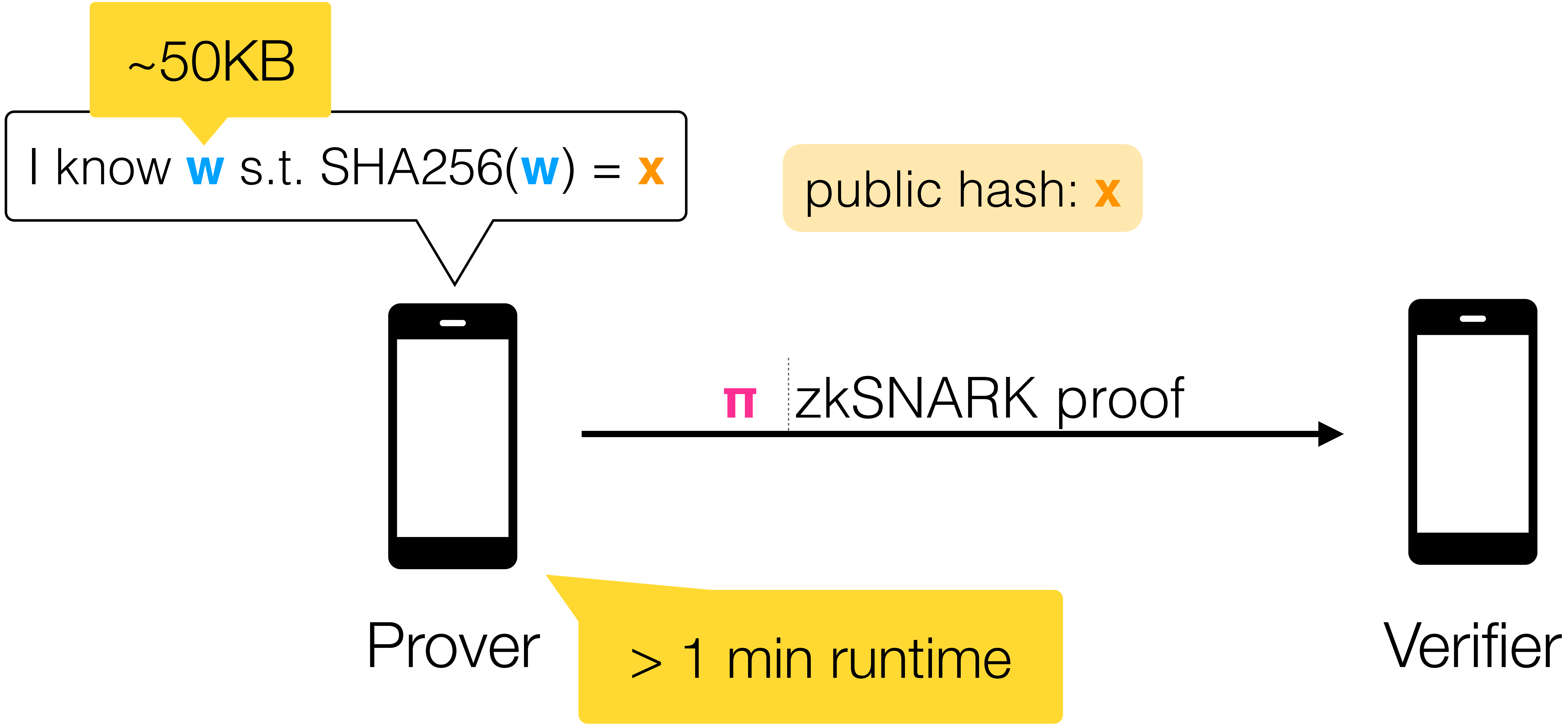
Problem: Proving is *really* slow



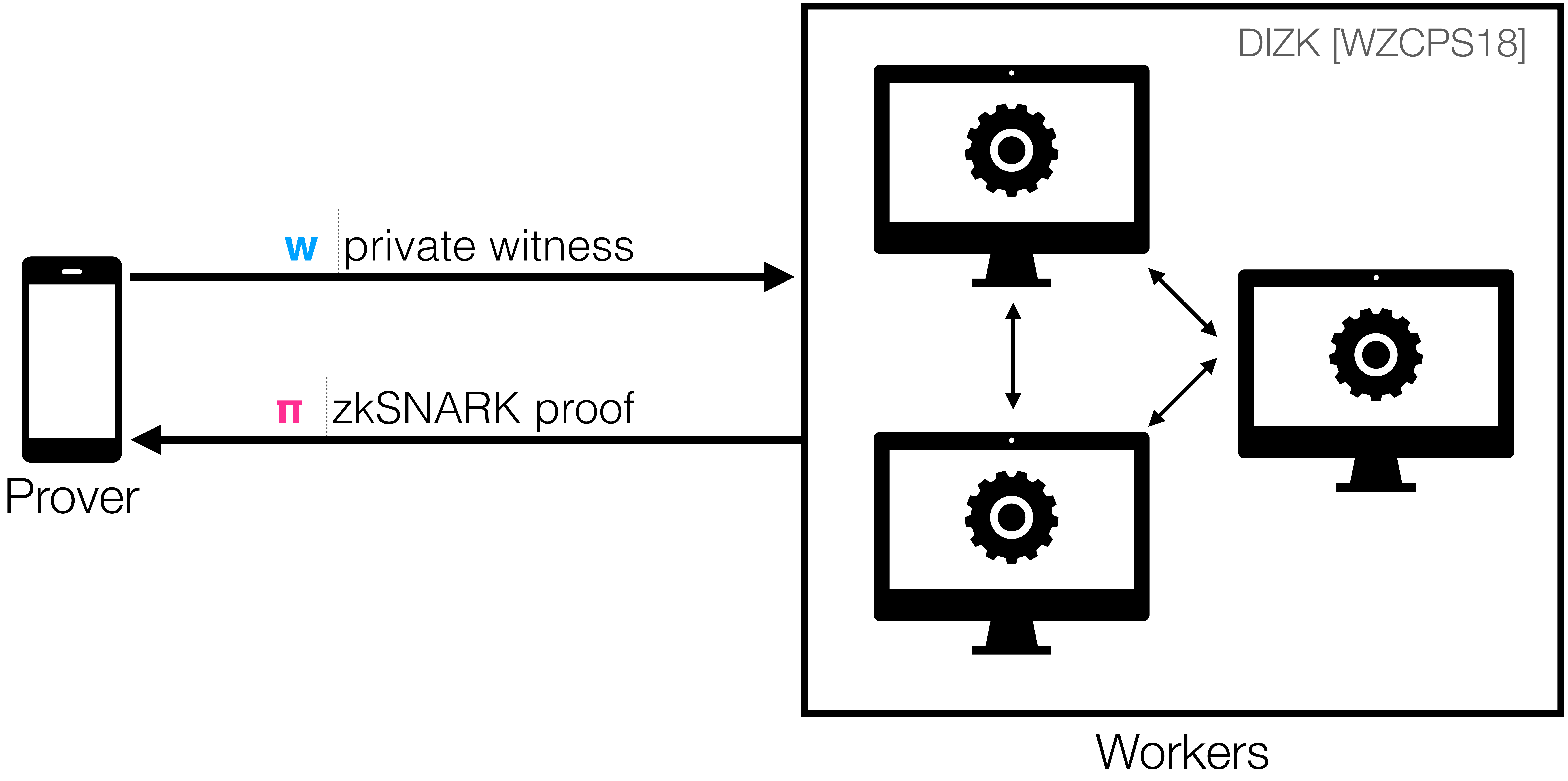
Problem: Proving is *really* slow



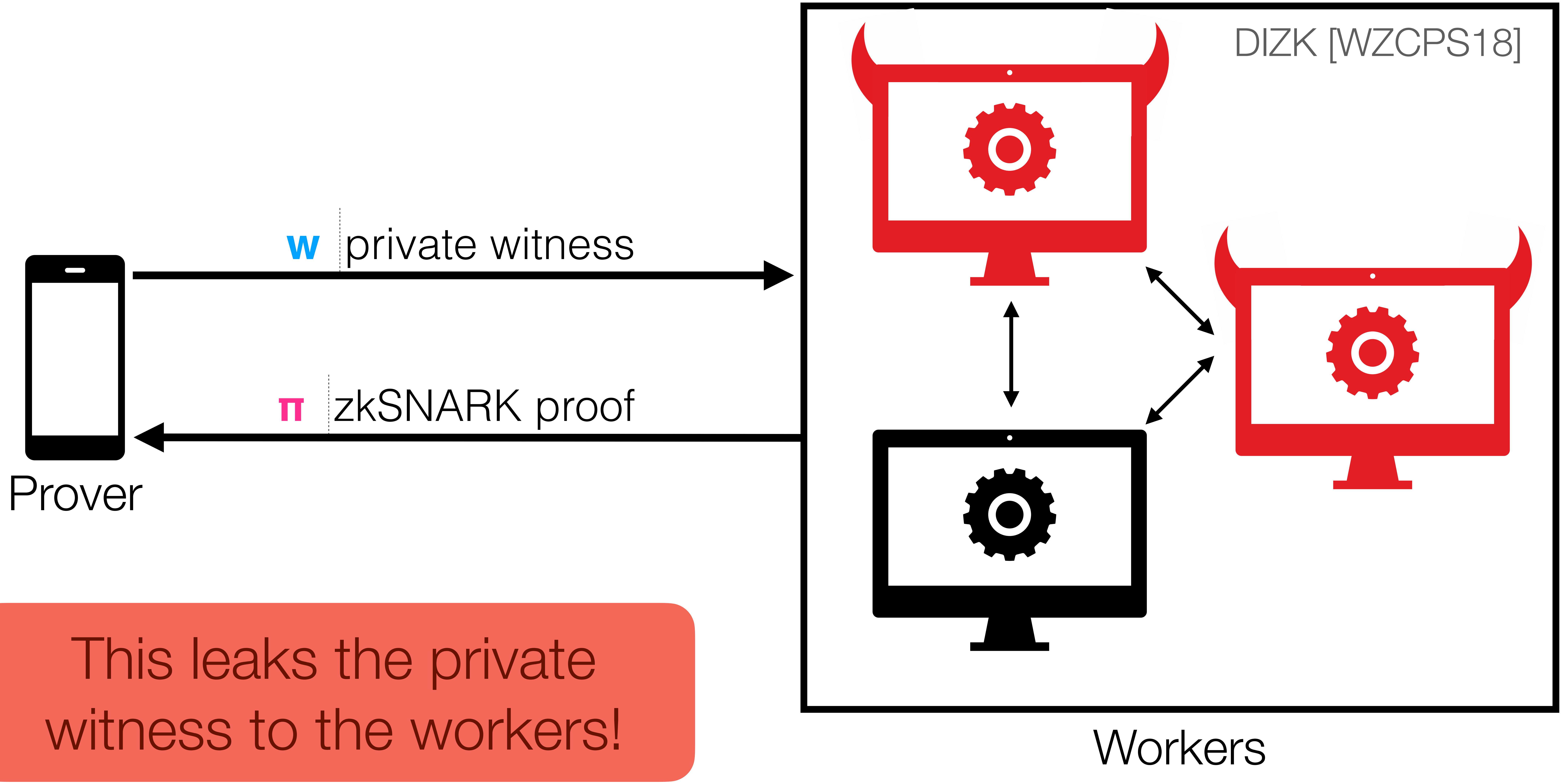
Problem: Proving is *really* slow



Potential Solution: Delegate Proving!

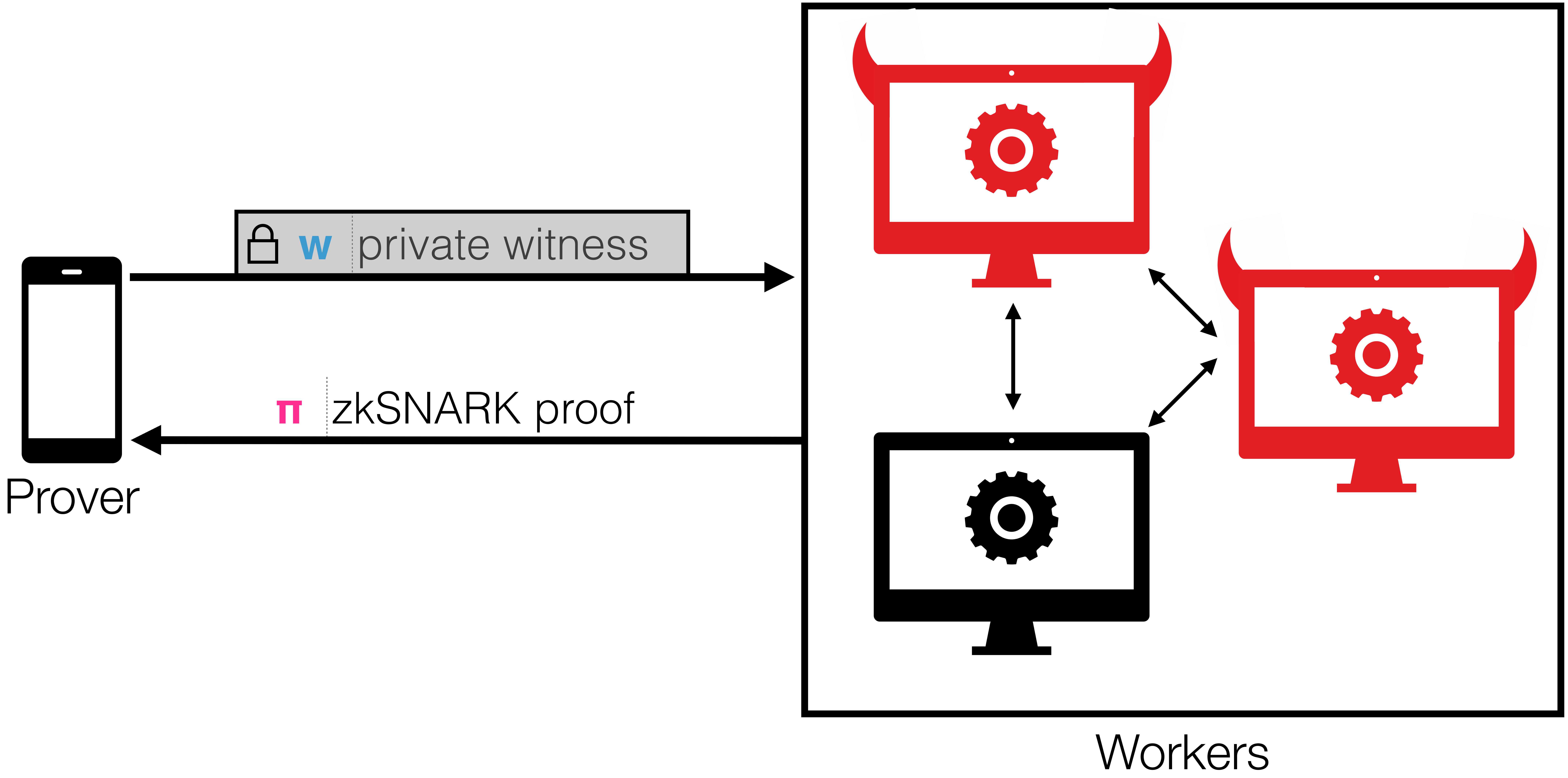


Potential Solution: Delegate Proving!

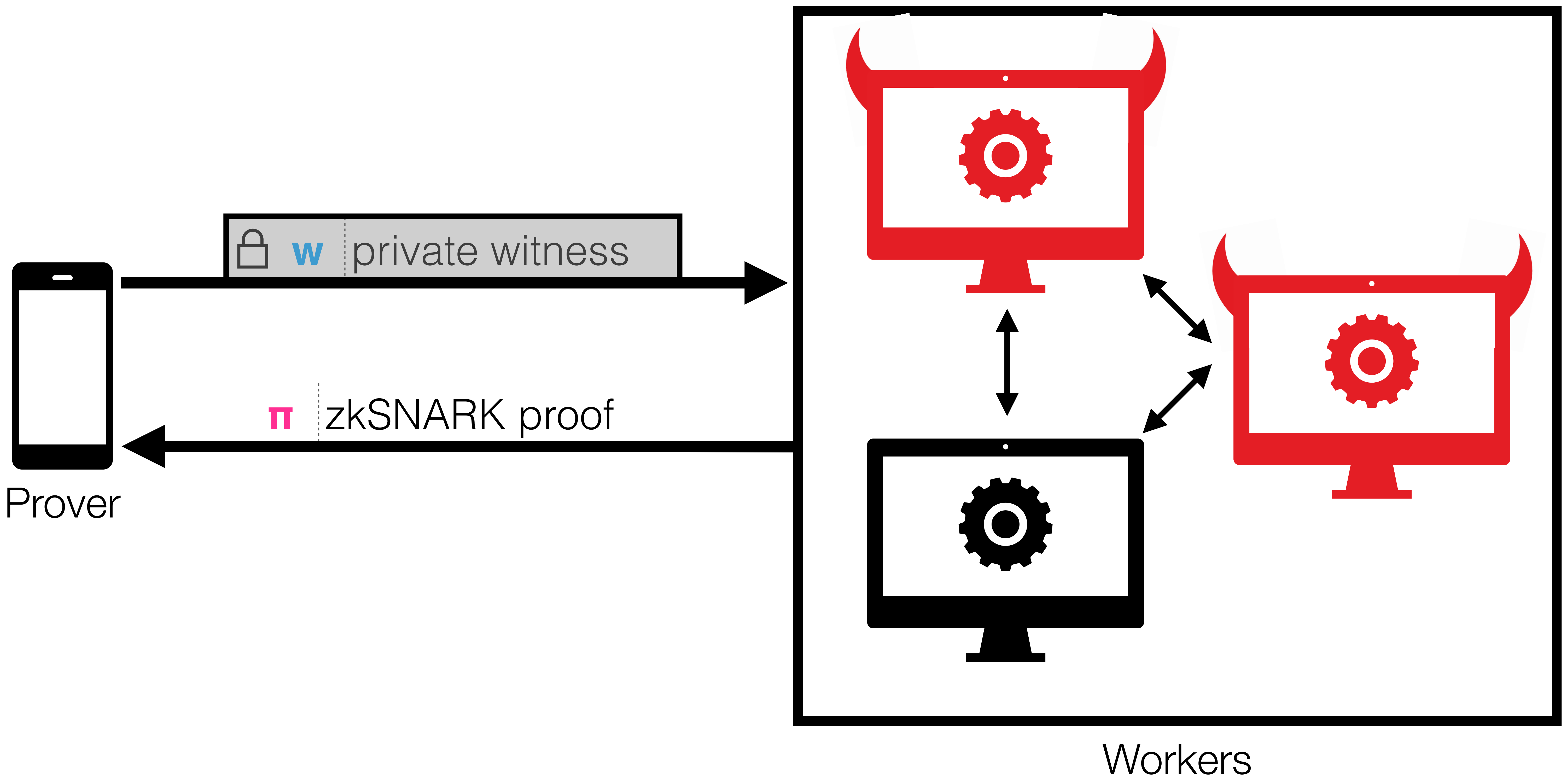


This leaks the private witness to the workers!

Goal: Delegate Proving with Privacy



Goal: Delegate Proving with Privacy



Goal 1: Efficiency — Delegation should be faster than local proving

Goal 2: Privacy — The witness should remain hidden from the workers if at least one worker is honest

Our results

Eos

Private delegation of ‘algebraic’ zkSNARK provers* in the presence of N-1 malicious workers.

*[MBKM19, GWC19, CHMMVW20, CFFQR21, BGH19]

Our results

Eos

Private delegation of 'algebraic' zkSNARK provers* in the presence of N-1 malicious workers.

*[MBKM19, GWC19, CHMMVW20, CFFQR21, BGH19]

Compared to local proving, running Eos on a mobile phone is:

1) **26x** faster
2) Uses **256x** less memory

Contributions

- Efficient circuits for zkSNARK provers
- Prover-assisted MPC
- Lightweight techniques for malicious security
- Systems optimizations

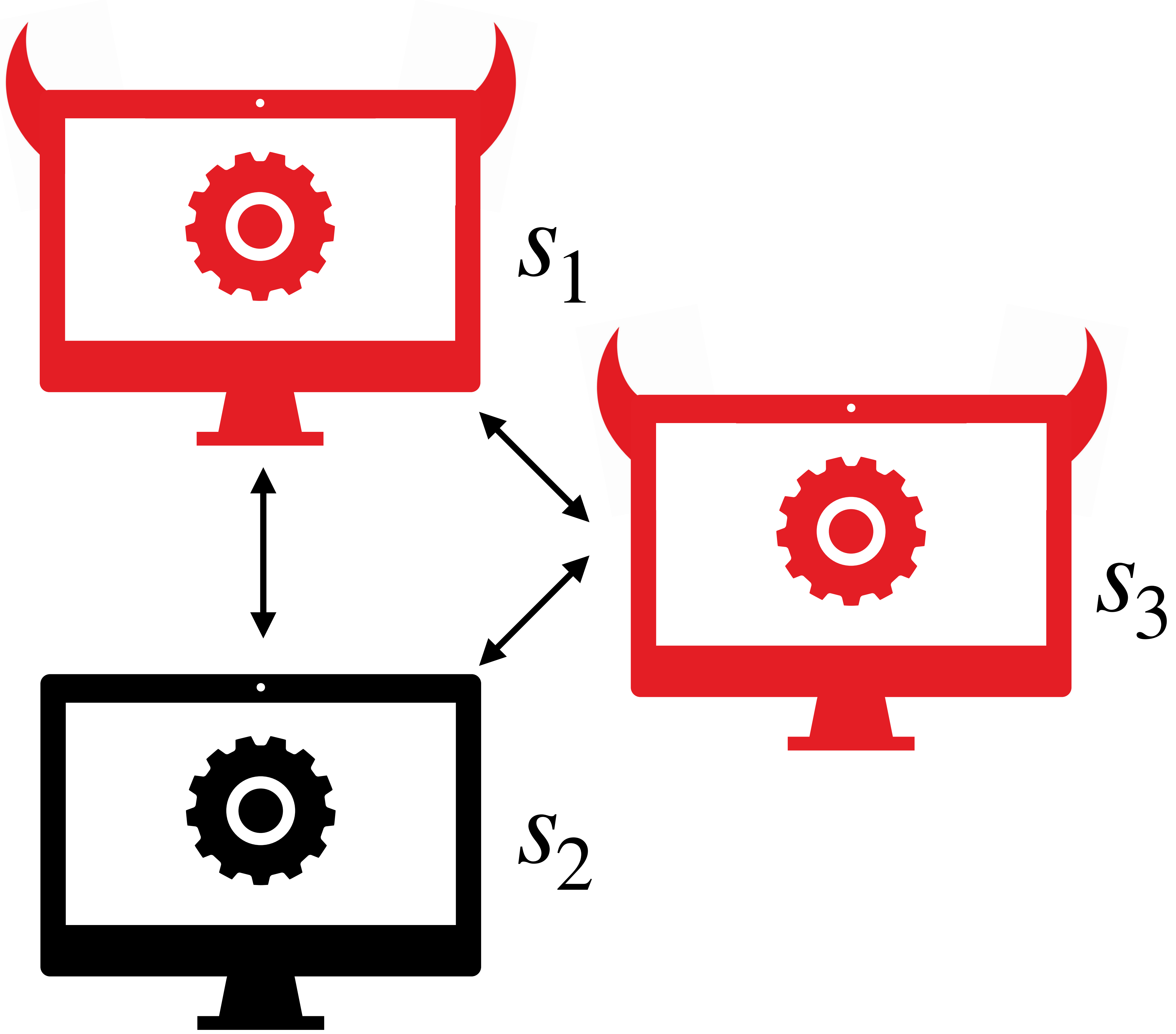
Contributions

- **Efficient circuits for zkSNARK provers**
- **Prover-assisted MPC**
- **Lightweight techniques for malicious security**
- Systems optimizations

Starting point: MPC

Allows multiple parties to compute a function F over their private inputs

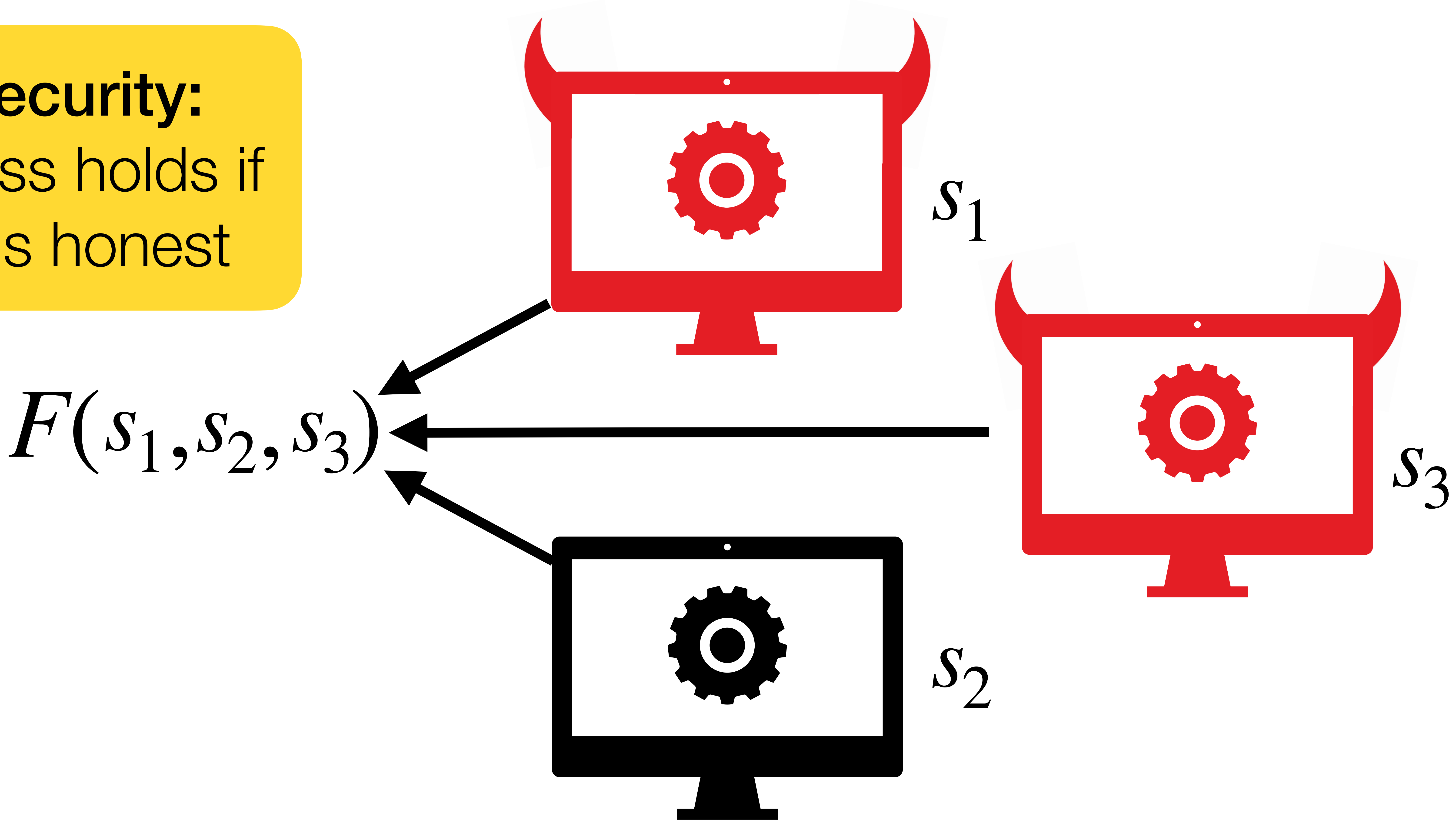
N-1 Malicious Security:
Privacy + correctness holds if at least one party is honest



Starting point: MPC

Allows multiple parties to compute a function F over their private inputs

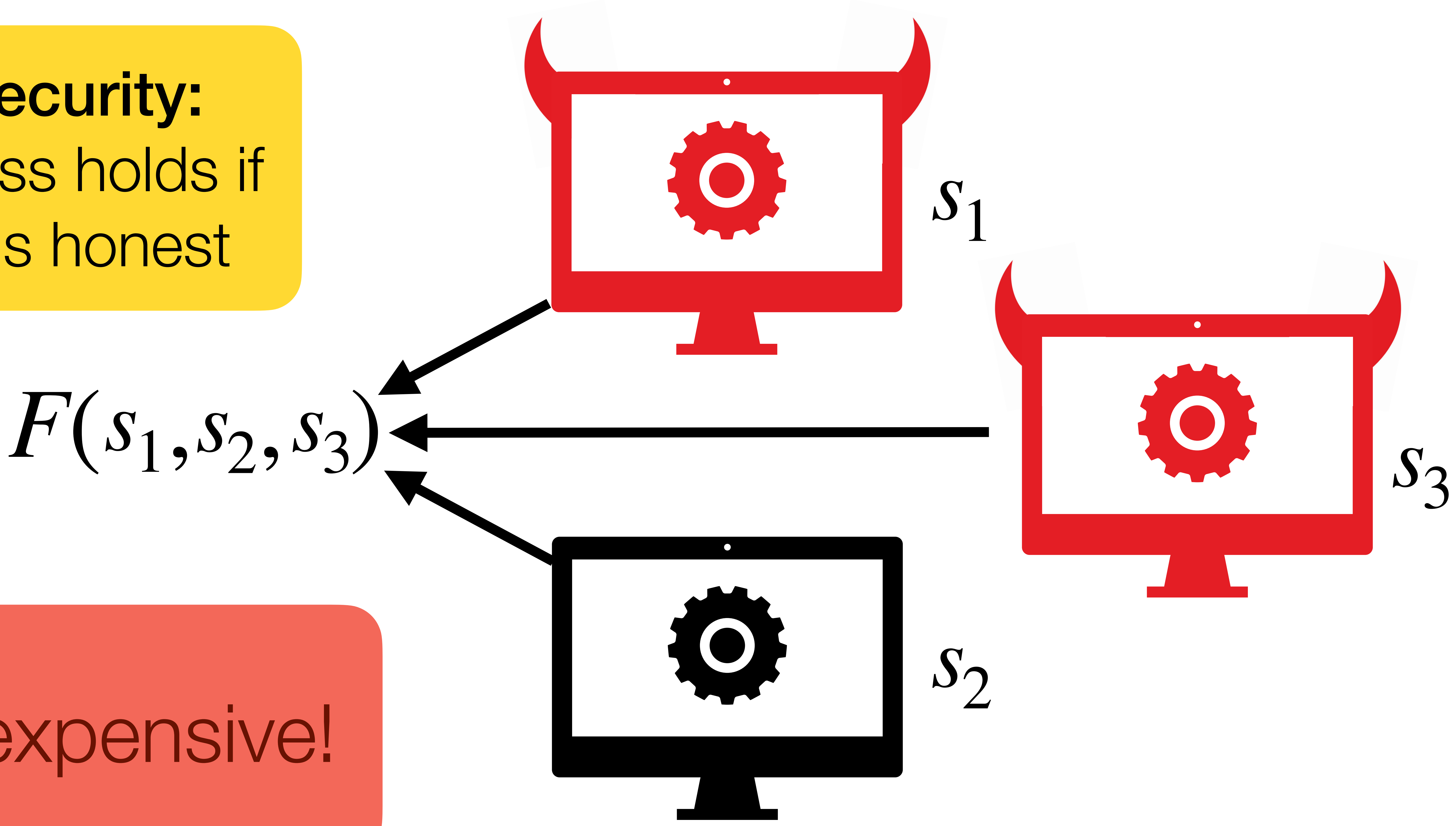
N-1 Malicious Security:
Privacy + correctness holds if at least one party is honest



Starting point: MPC

Allows multiple parties to compute a function F over their private inputs

N-1 Malicious Security:
Privacy + correctness holds if at least one party is honest



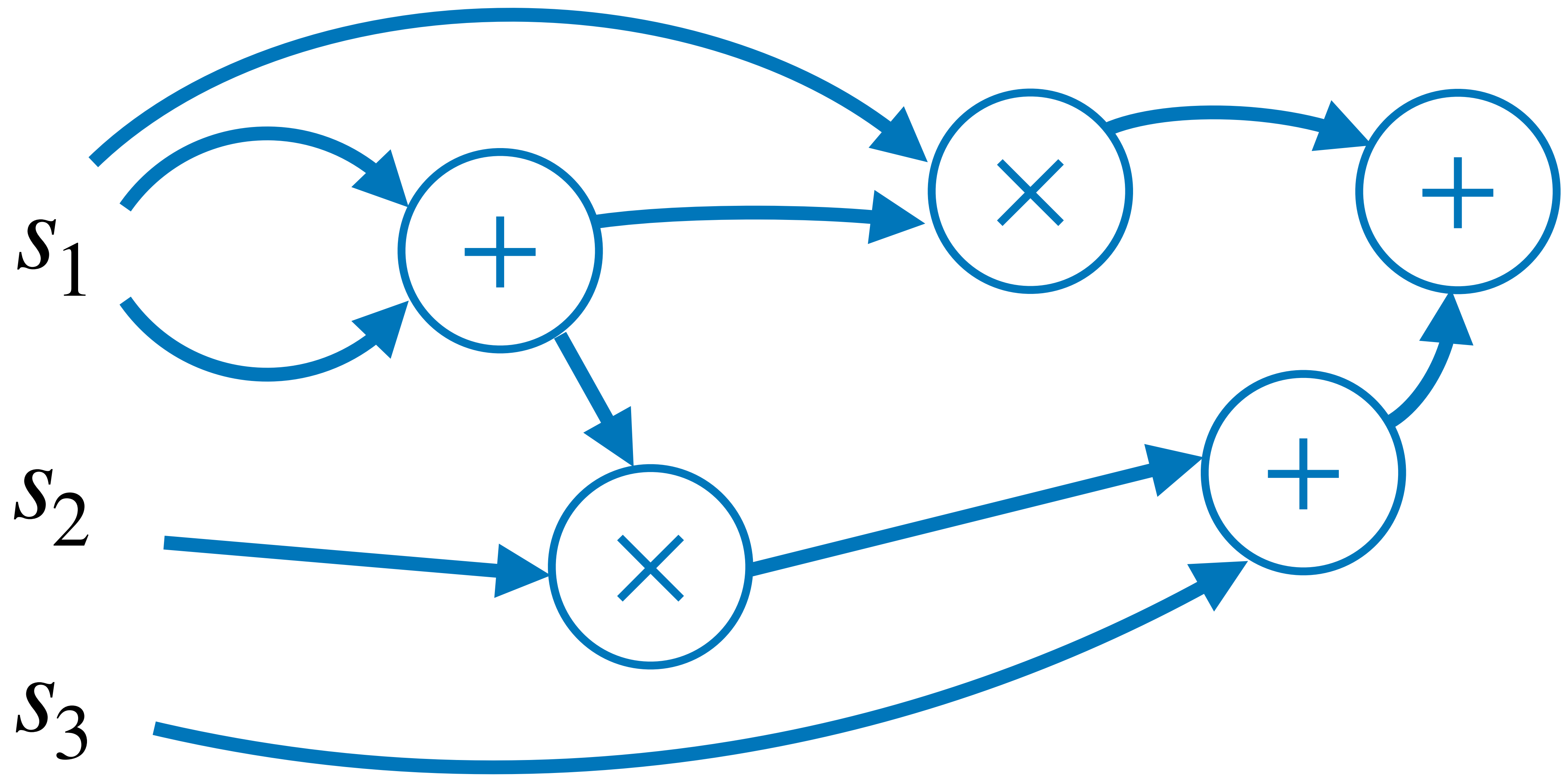
Generic MPC is expensive!

Efficient Circuits for zkSNARK Provers

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit

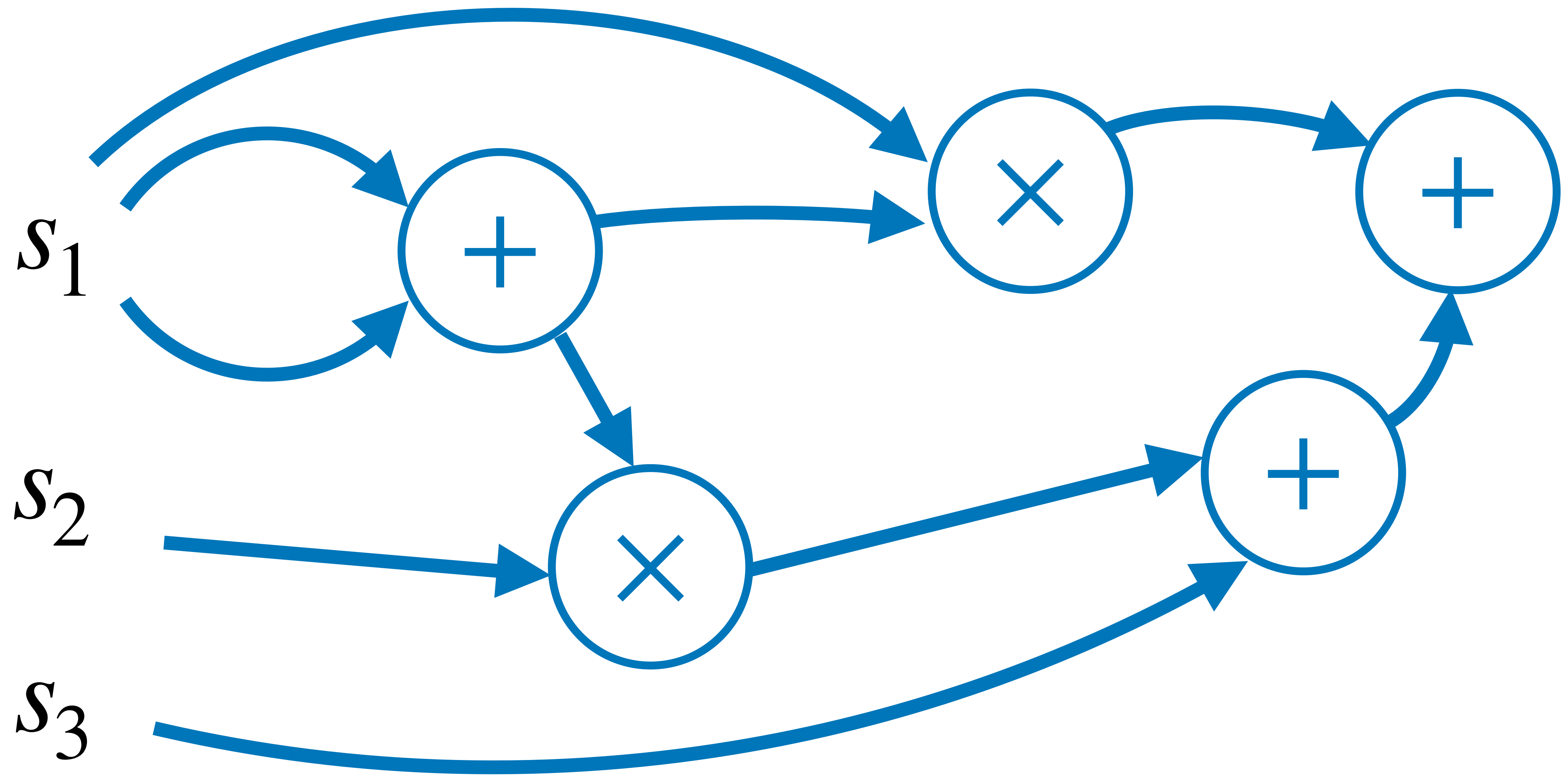
2) All parties execute circuit gate-by-gate



Efficient Circuits for zkSNARK Provers

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit
2) All parties execute circuit gate-by-gate

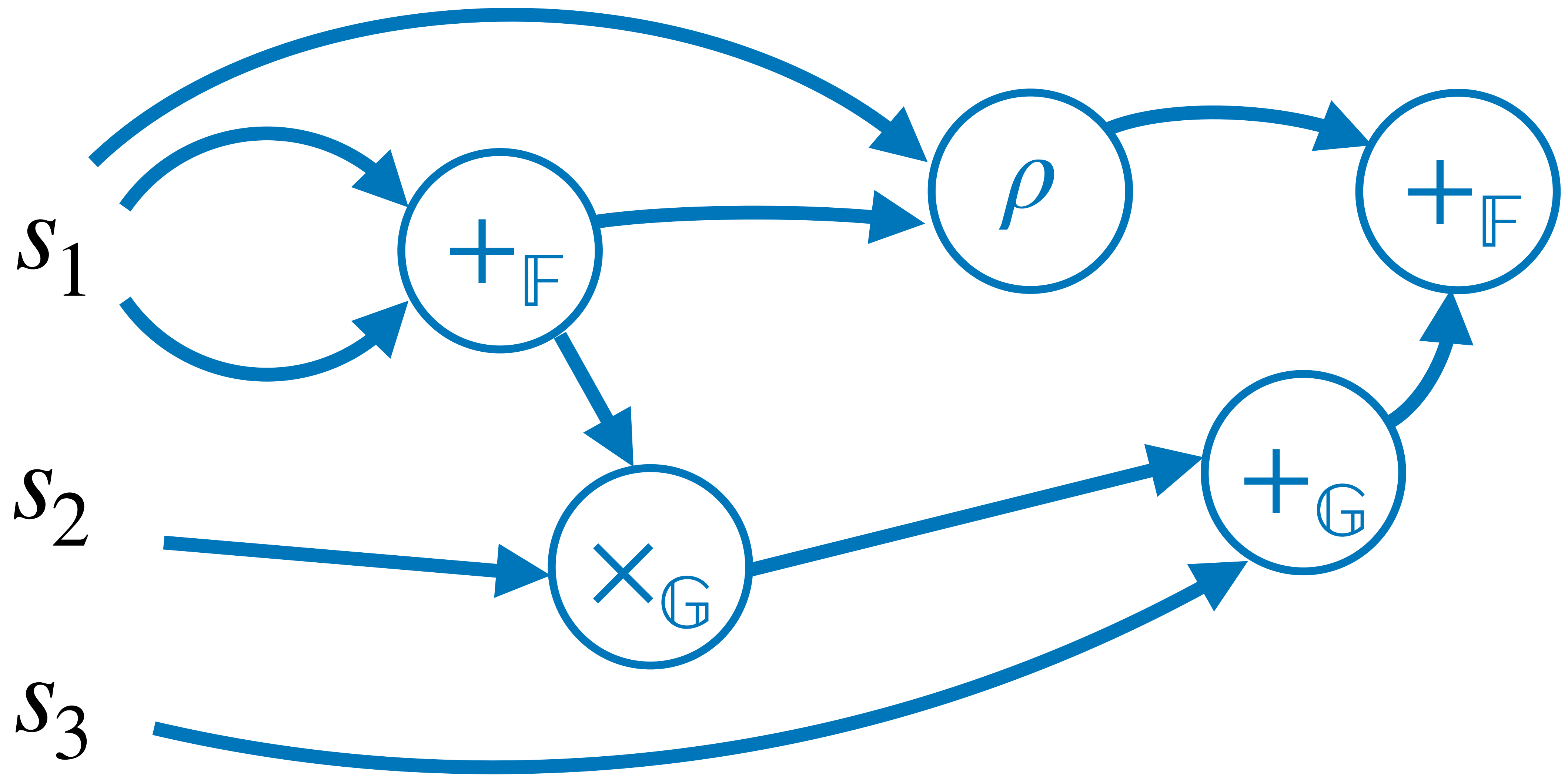


Circuit for zkSNARK prover is large! Need to support polynomial arithmetic, group operations, and random oracle calls

Efficient Circuits for zkSNARK Provers

MPC Protocol [SPDZ]

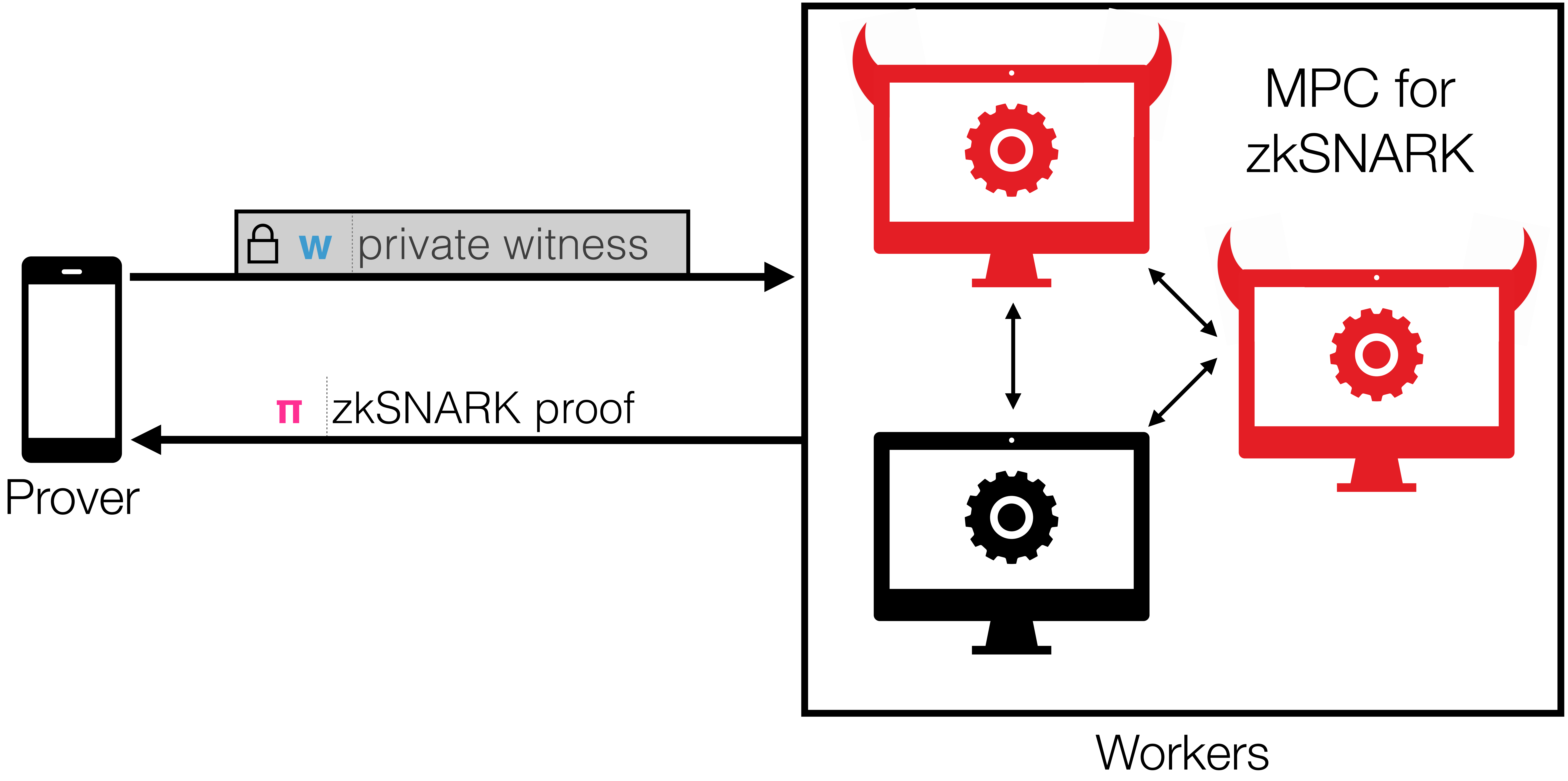
1) Express function as an arithmetic circuit
2) All parties execute circuit gate-by-gate



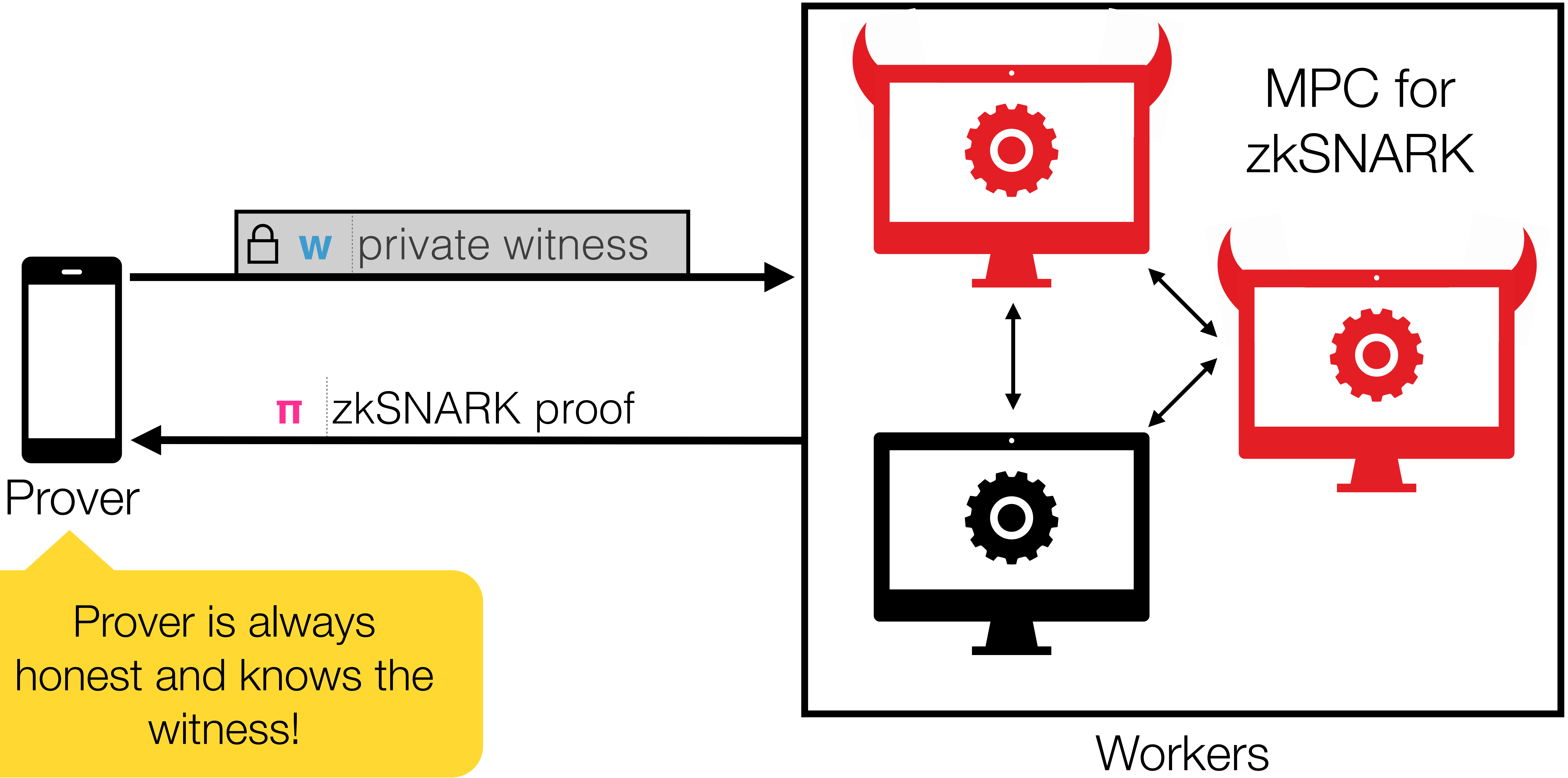
Idea: Extend the circuit model [SA19, OB22]!

- Add gates for $\mathbb{G}$-ops and random oracle calls
- New, efficient subcircuits for polynomial arithmetic

Using the Asymmetric Threat Model

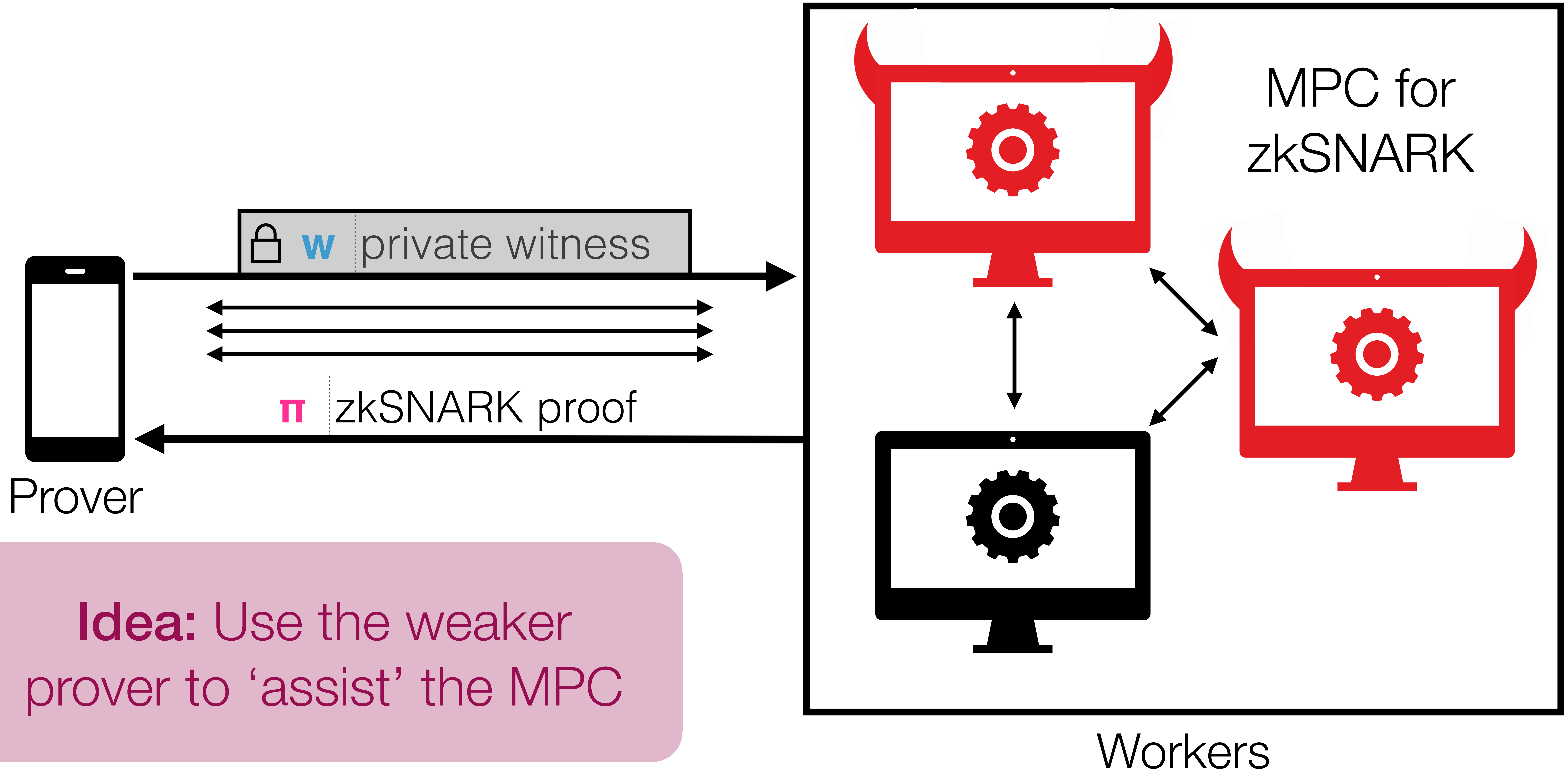


Using the Asymmetric Threat Model



Prover is always honest and knows the witness!

Using the Asymmetric Threat Model



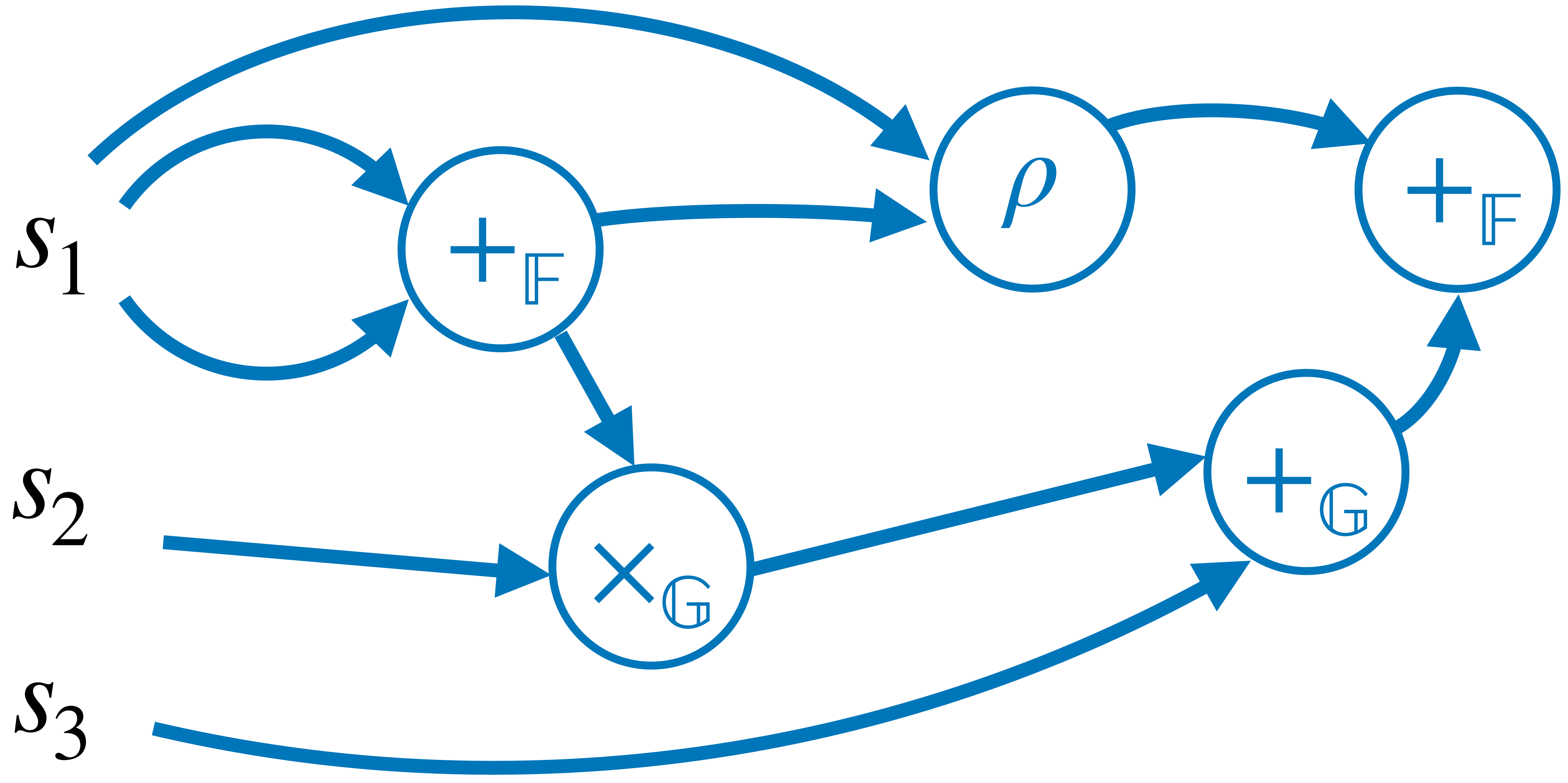
Idea: Use the weaker prover to 'assist' the MPC

Prover-assisted MPC

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit

2) All parties execute circuit gate-by-gate

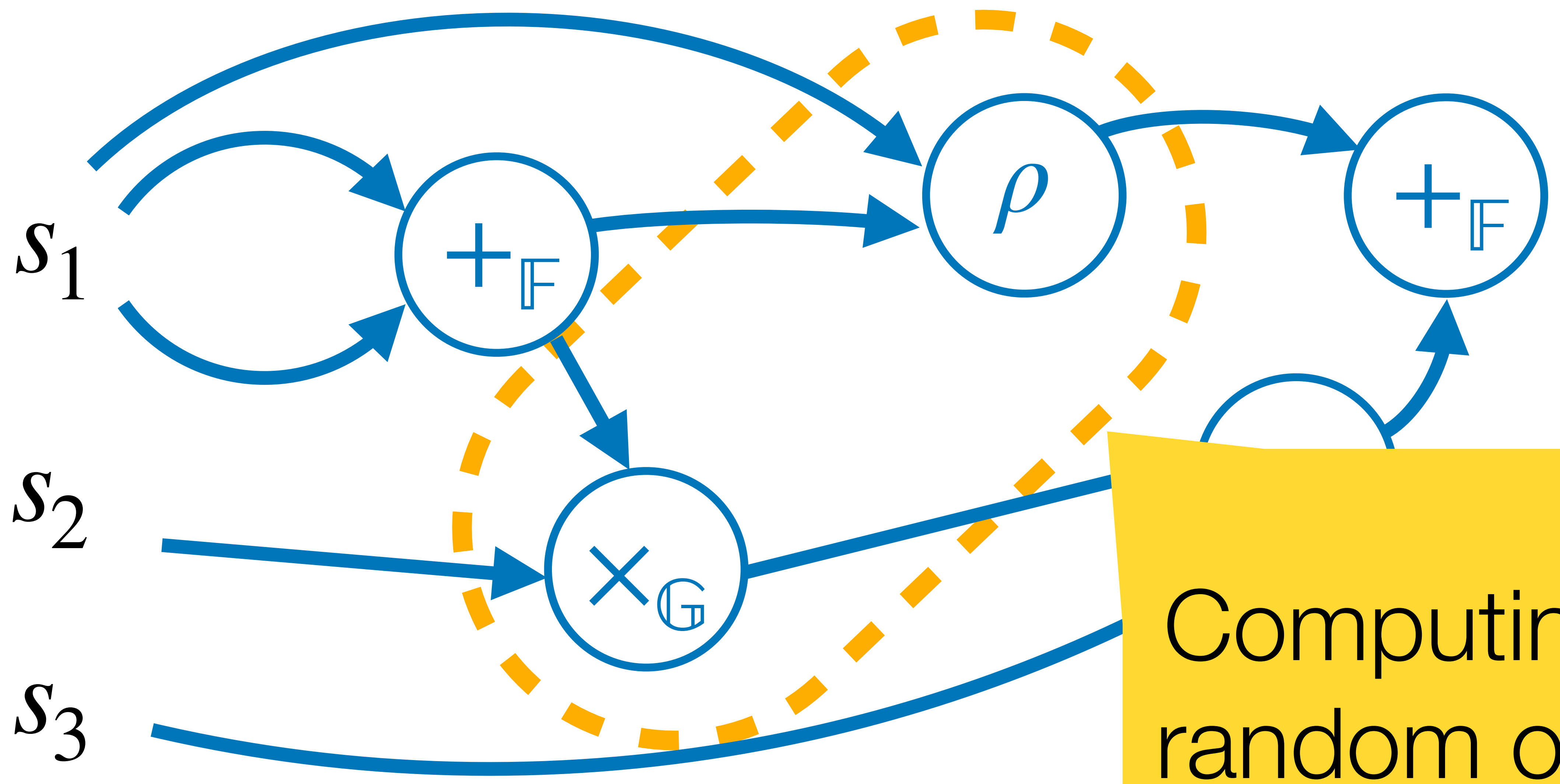


Prover-assisted MPC

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit

2) All parties execute circuit gate-by-gate



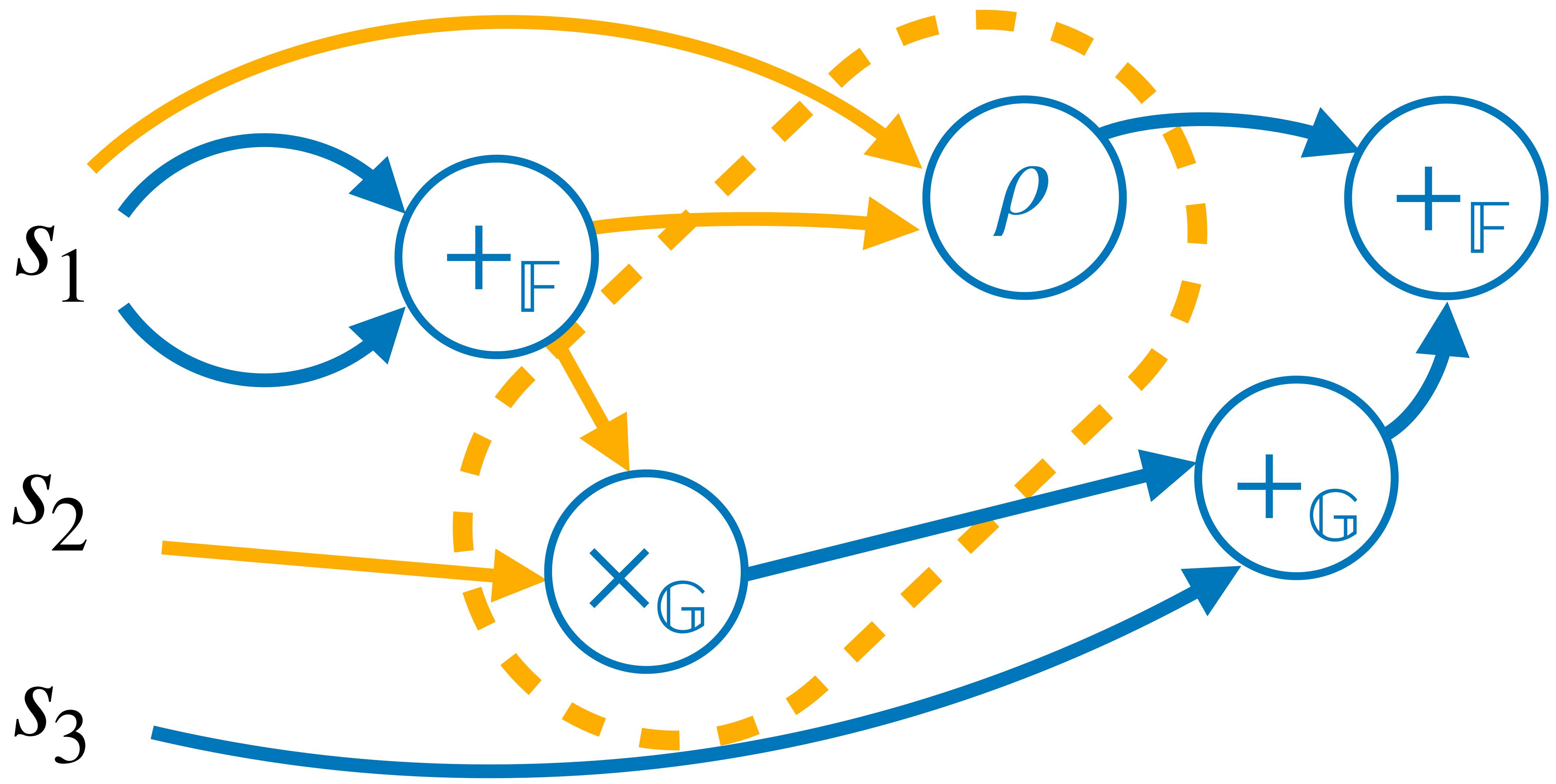
Computing multiplications and random oracles is expensive in MPC but cheap in plaintext

Prover-assisted MPC

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit

2) All parties execute circuit gate-by-gate



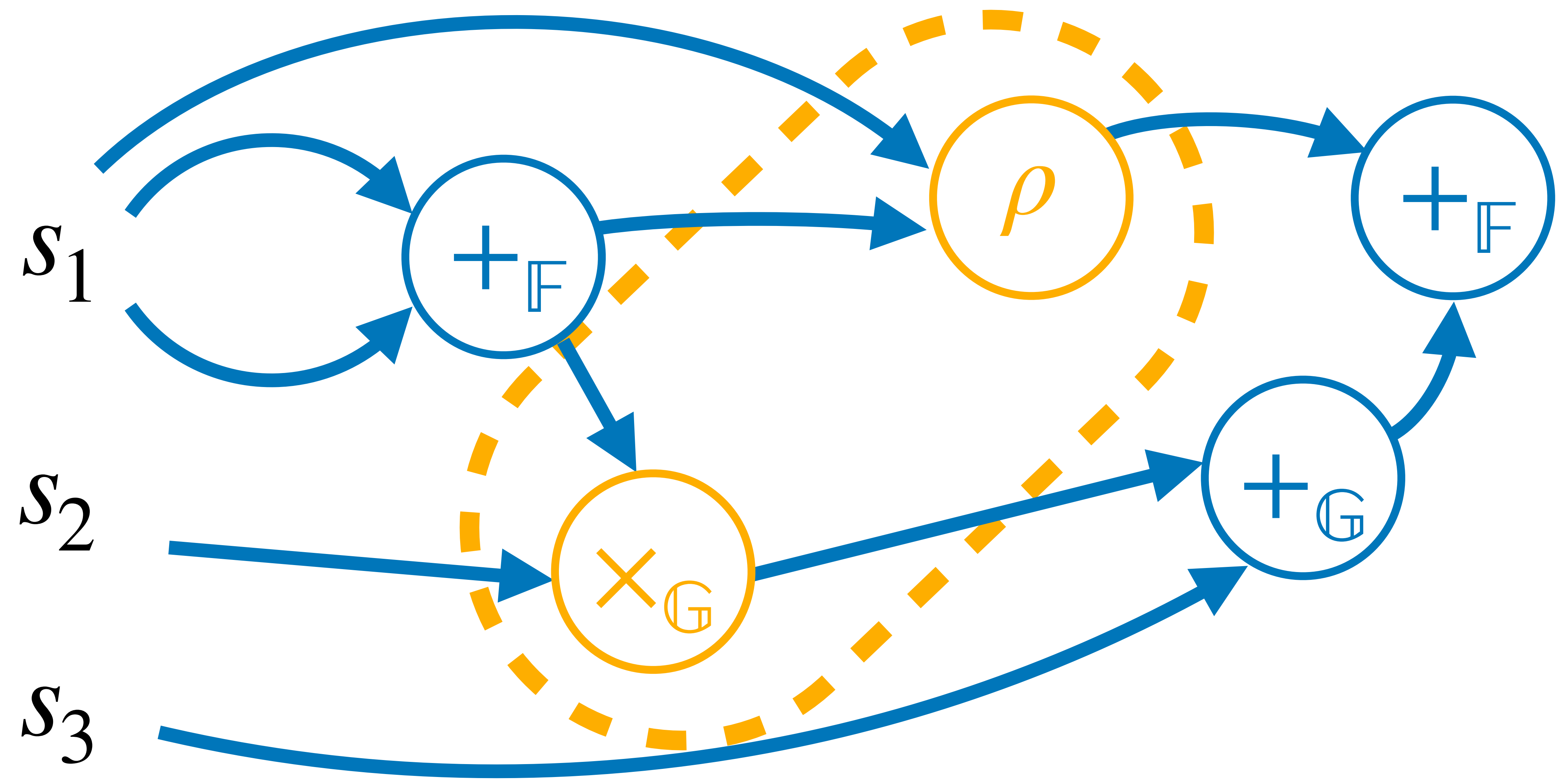
- Workers share subcircuit input wires with prover

Prover-assisted MPC

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit

2) All parties execute circuit gate-by-gate



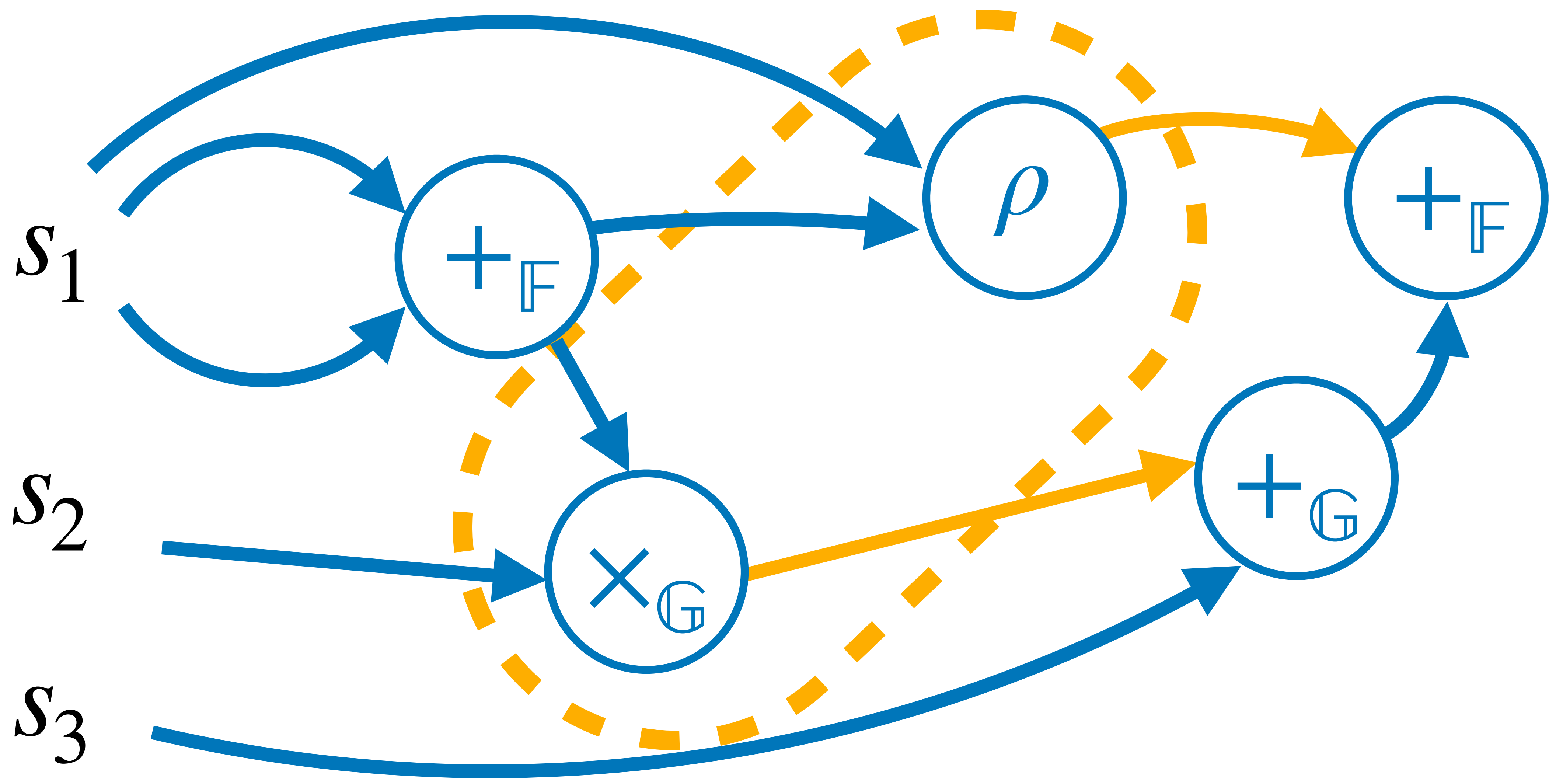
- Workers share subcircuit input wires with prover
- Prover executes the subcircuit in plaintext

Prover-assisted MPC

MPC Protocol [SPDZ]

1) Express function as an arithmetic circuit

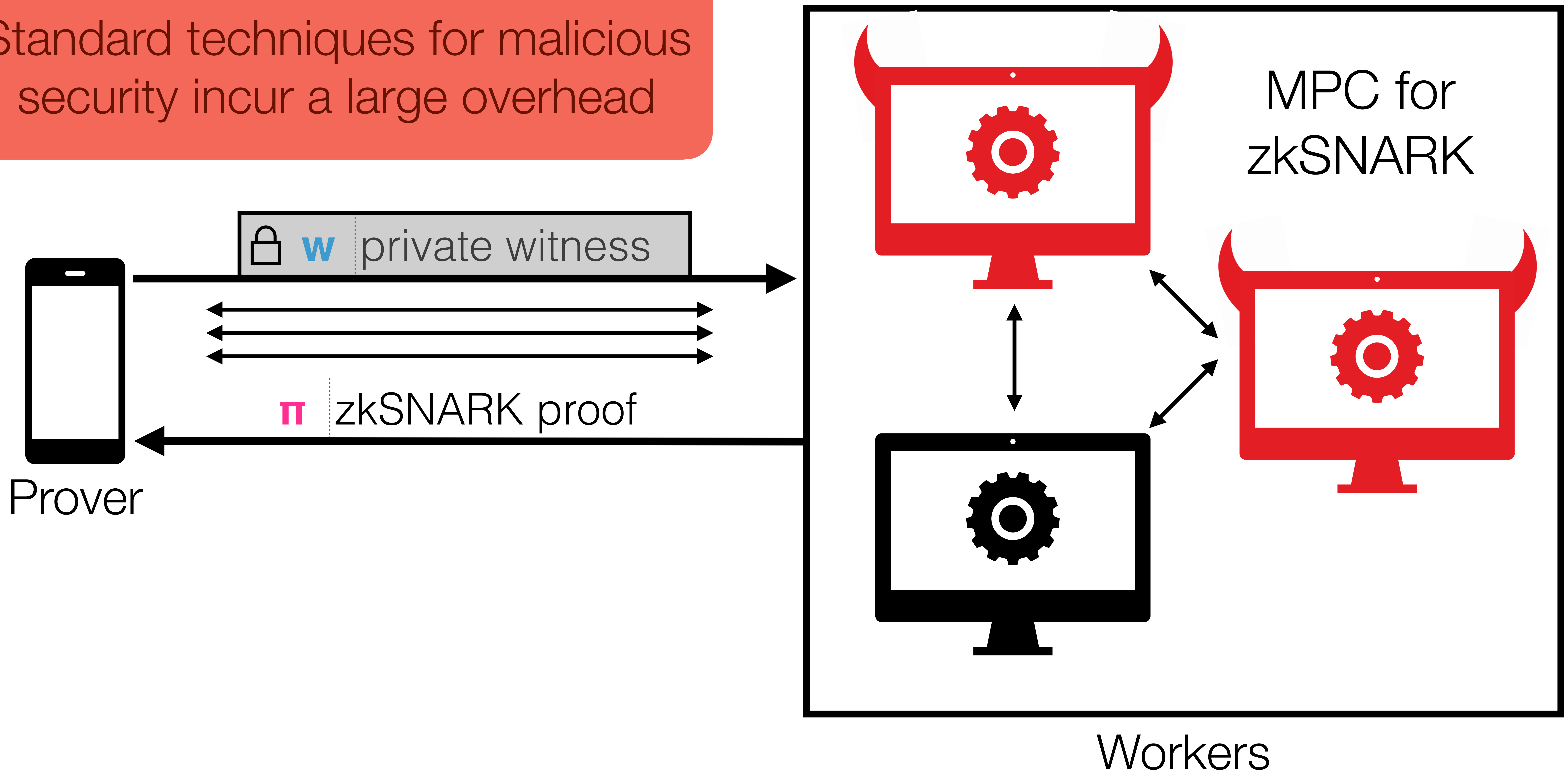
2) All parties execute circuit gate-by-gate



- Workers share subcircuit input wires with prover
- Prover executes the subcircuit in plaintext
- Prover sends subcircuit output wires to workers

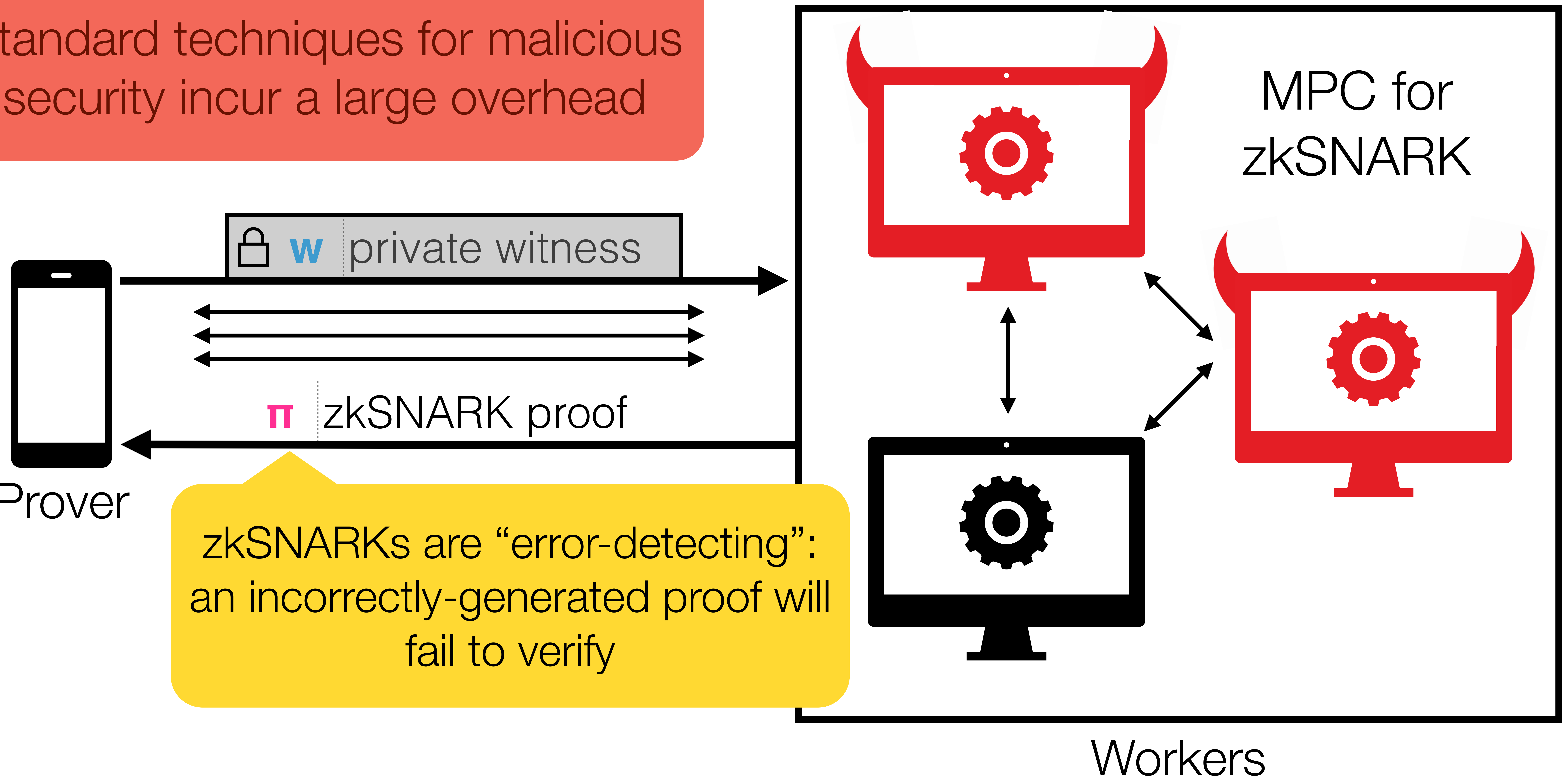
Achieving Malicious Security

Standard techniques for malicious security incur a large overhead



Achieving Malicious Security

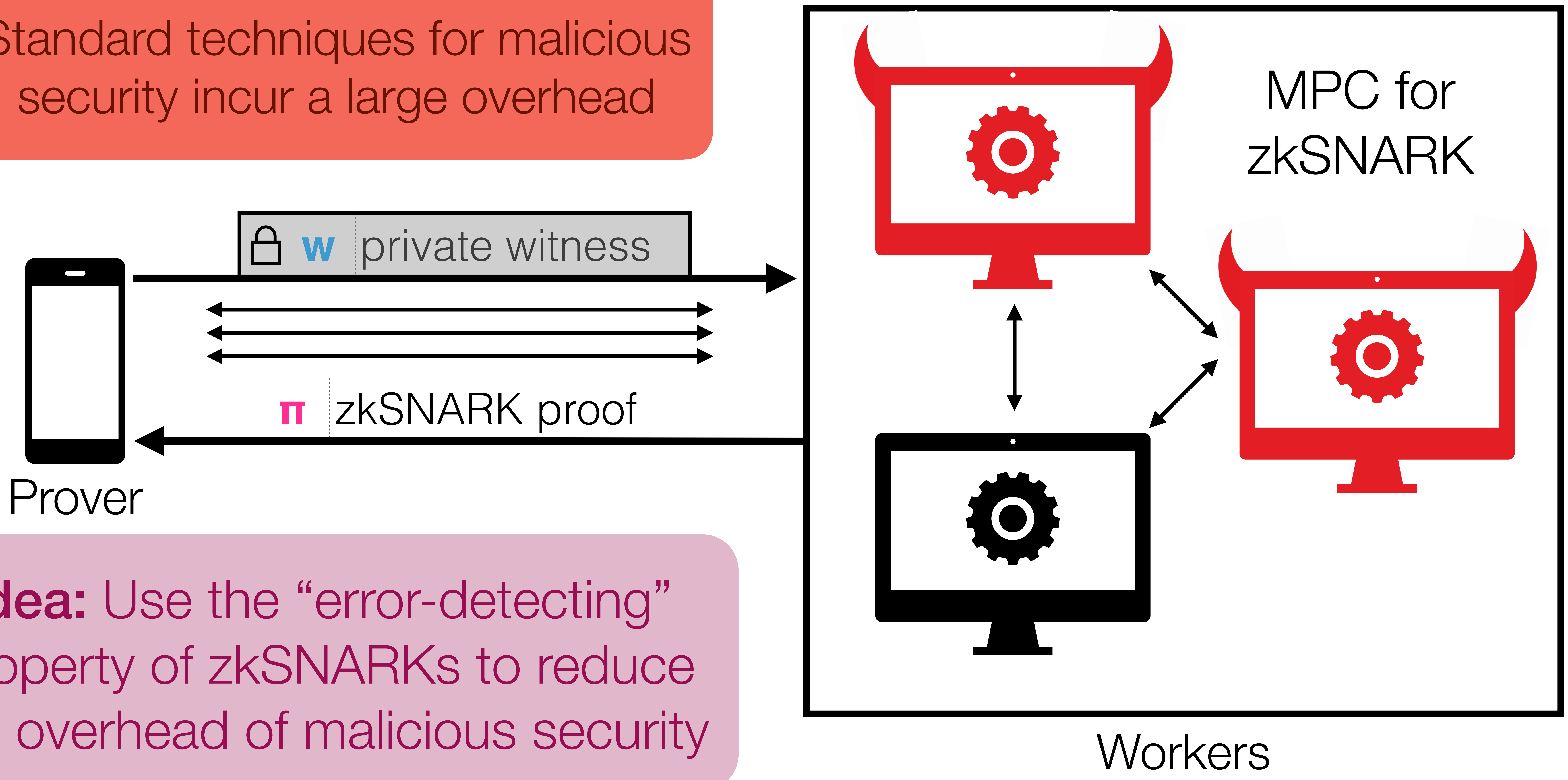
Standard techniques for malicious security incur a large overhead



zkSNARKs are "error-detecting": an incorrectly-generated proof will fail to verify

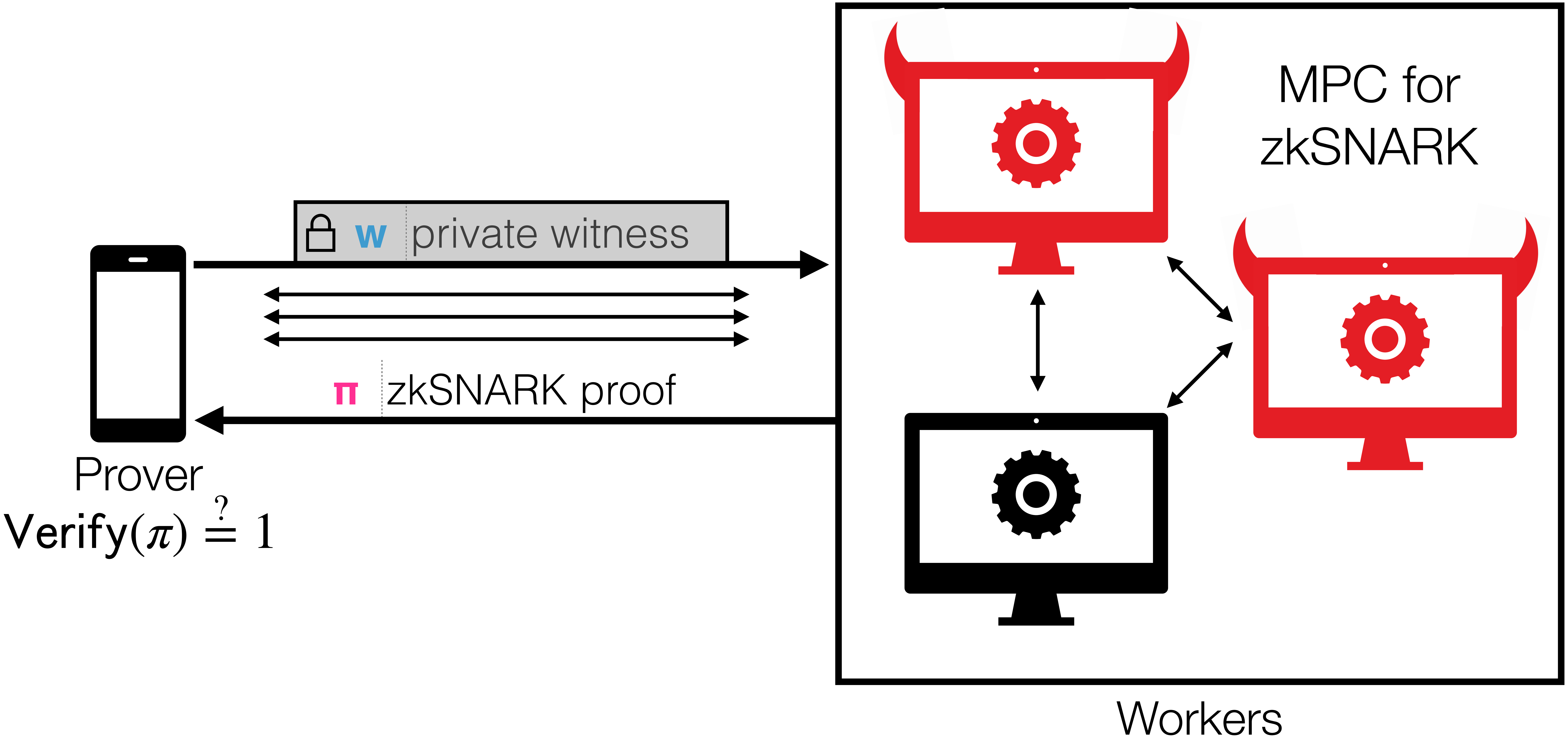
Achieving Malicious Security

Standard techniques for malicious security incur a large overhead

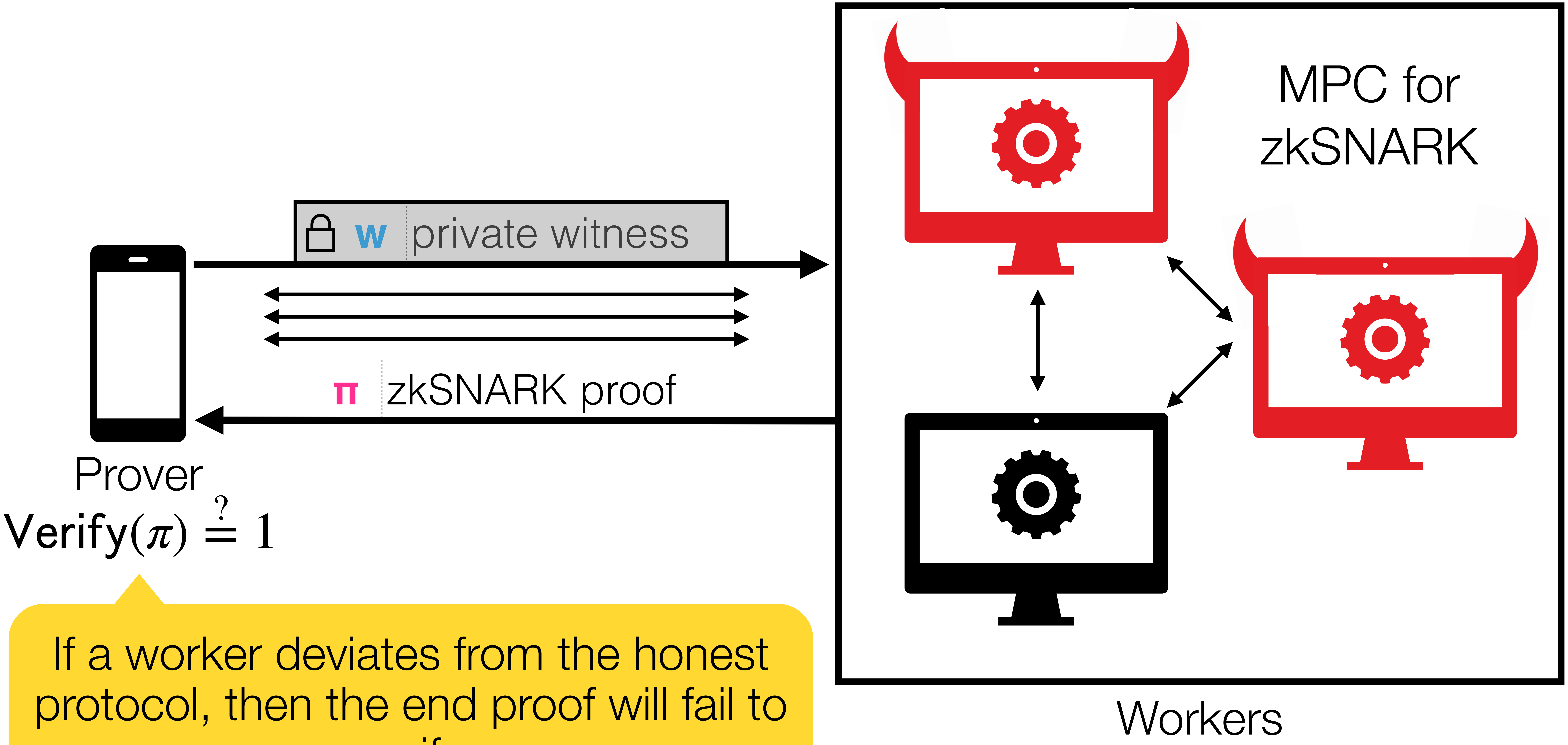


Idea: Use the "error-detecting" property of zkSNARKs to reduce the overhead of malicious security

Achieving Malicious Security



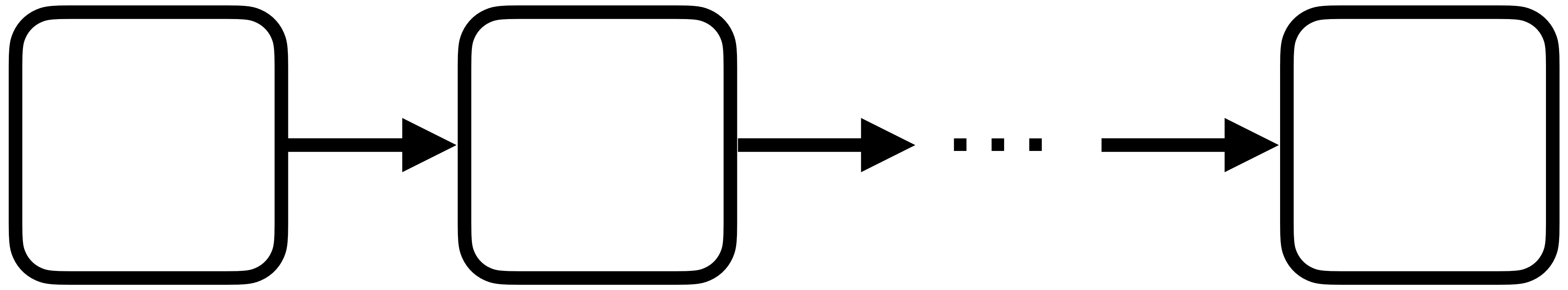
Achieving Malicious Security

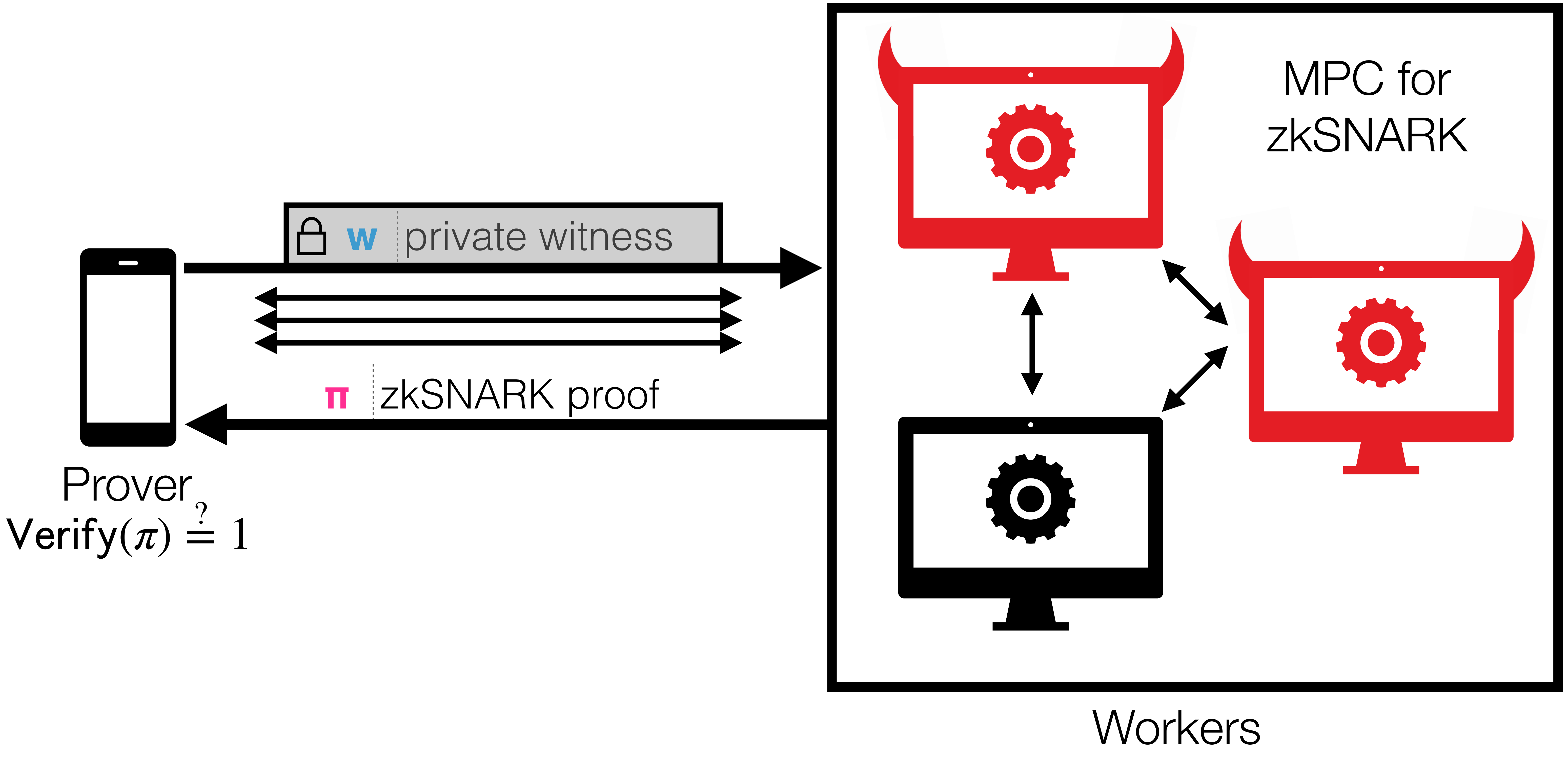


Verify(π) $\stackrel{?}{=}$ 1

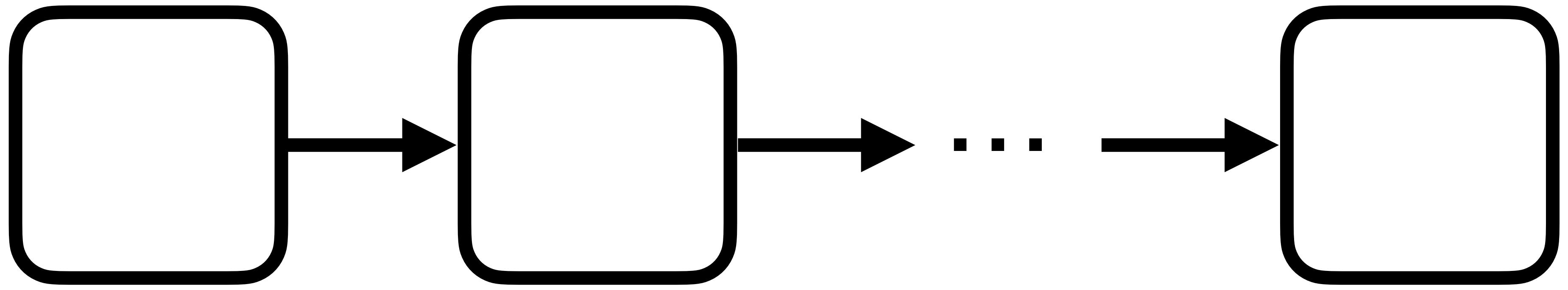
If a worker deviates from the honest protocol, then the end proof will fail to verify

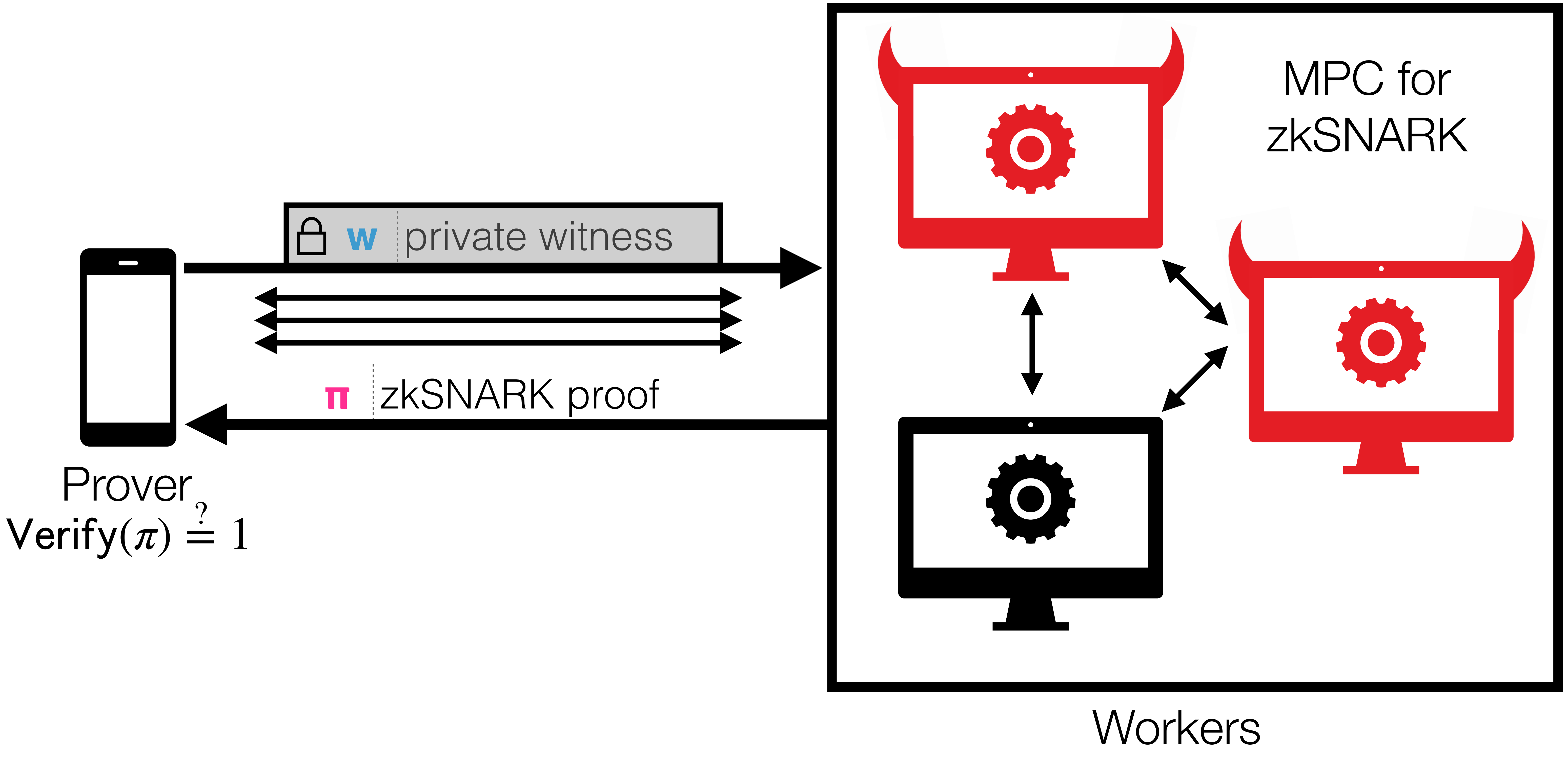
Achieving Malicious Security



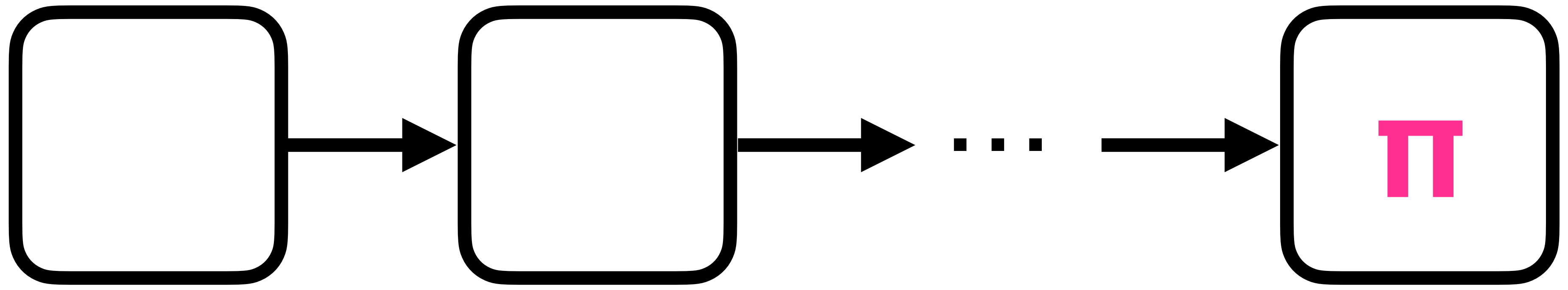


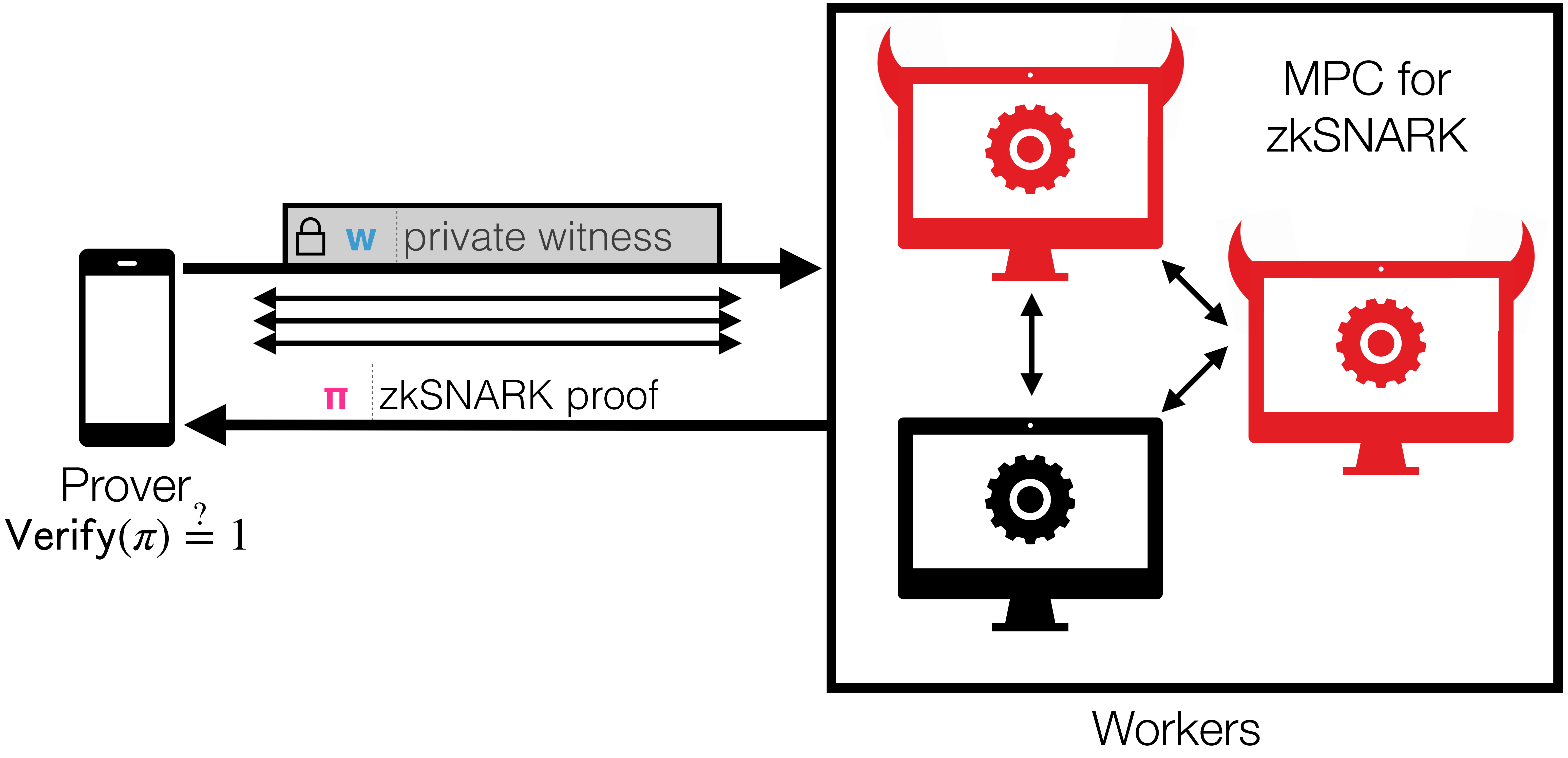
Achieving Malicious Security



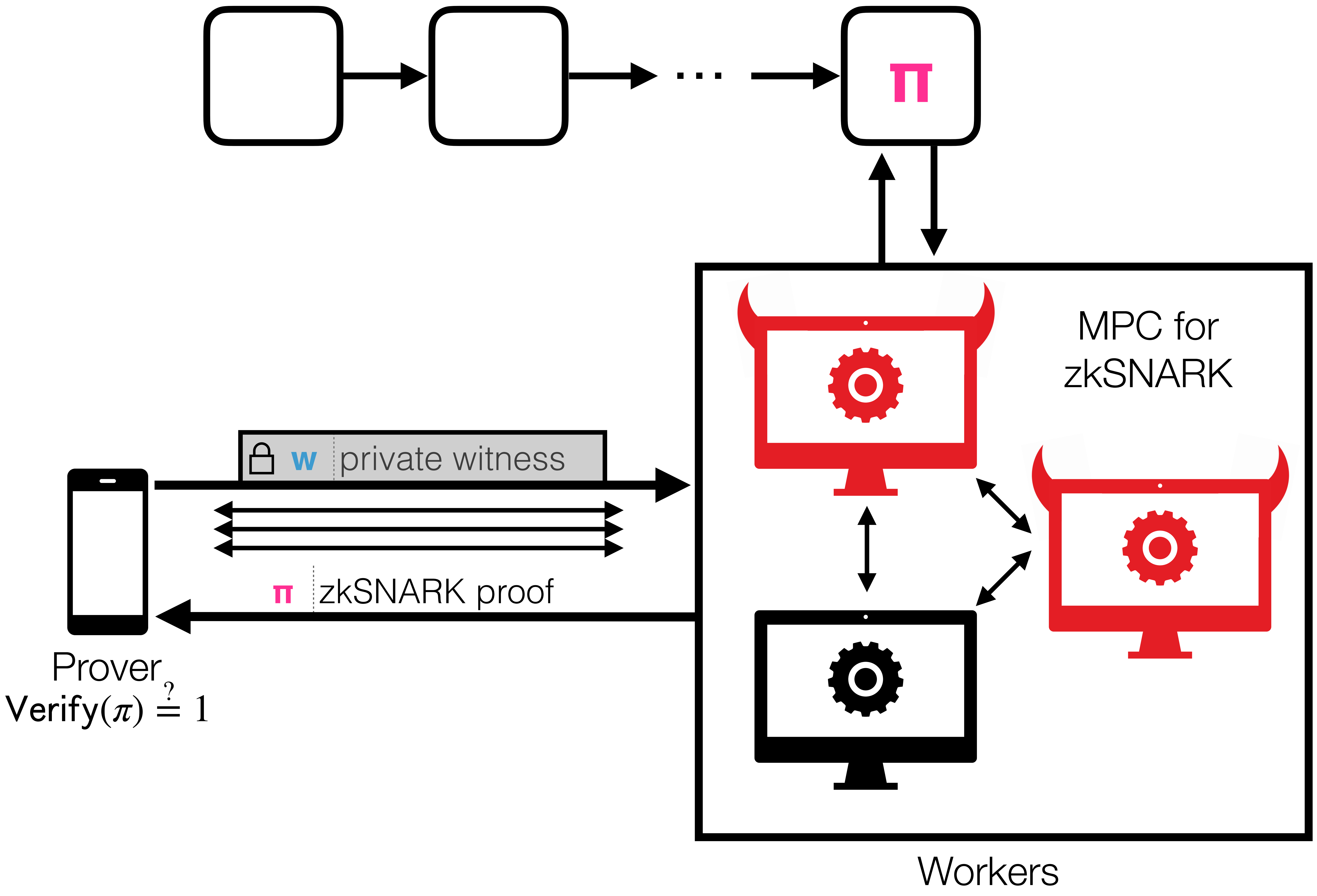


Achieving Malicious Security

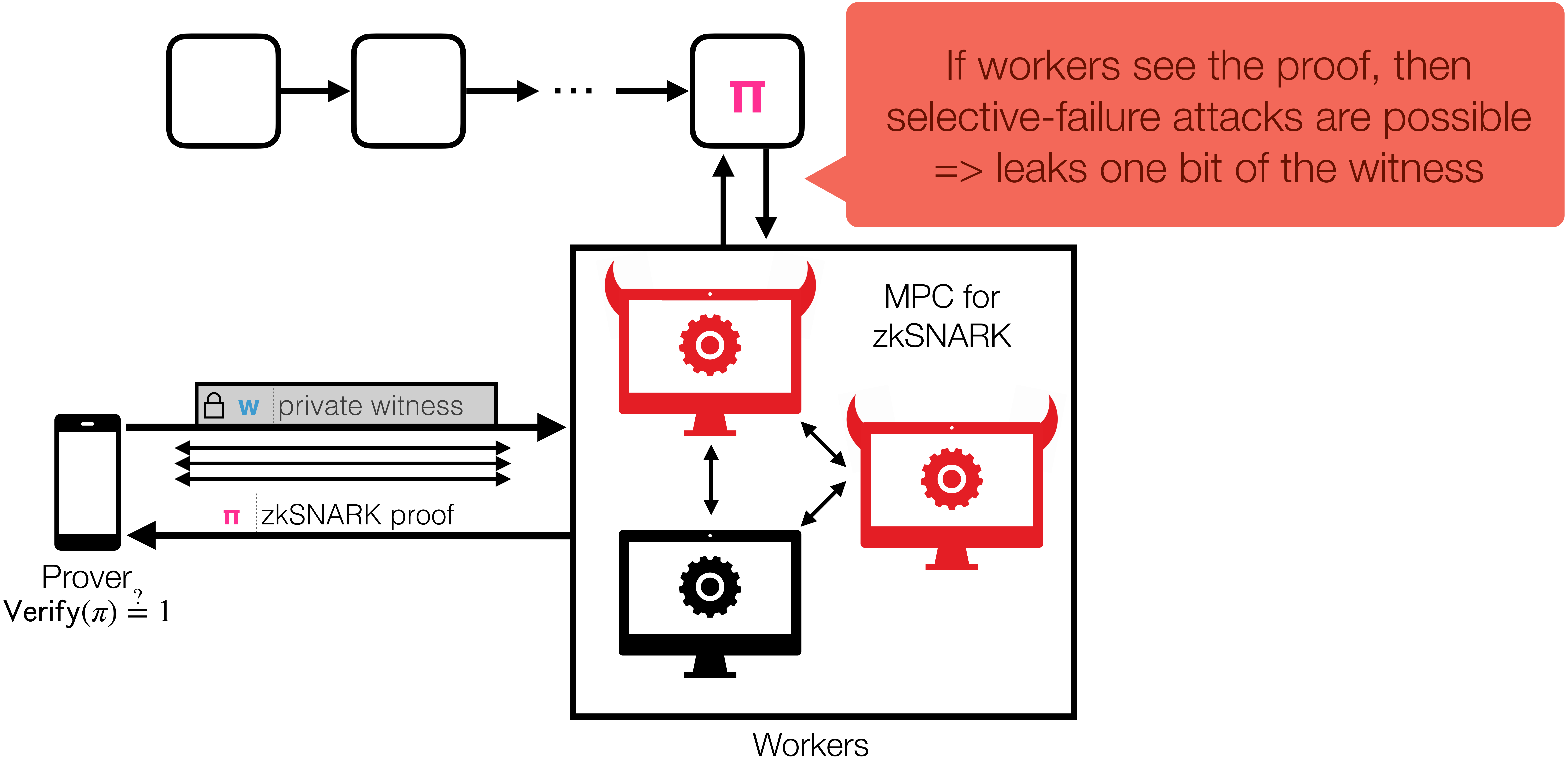




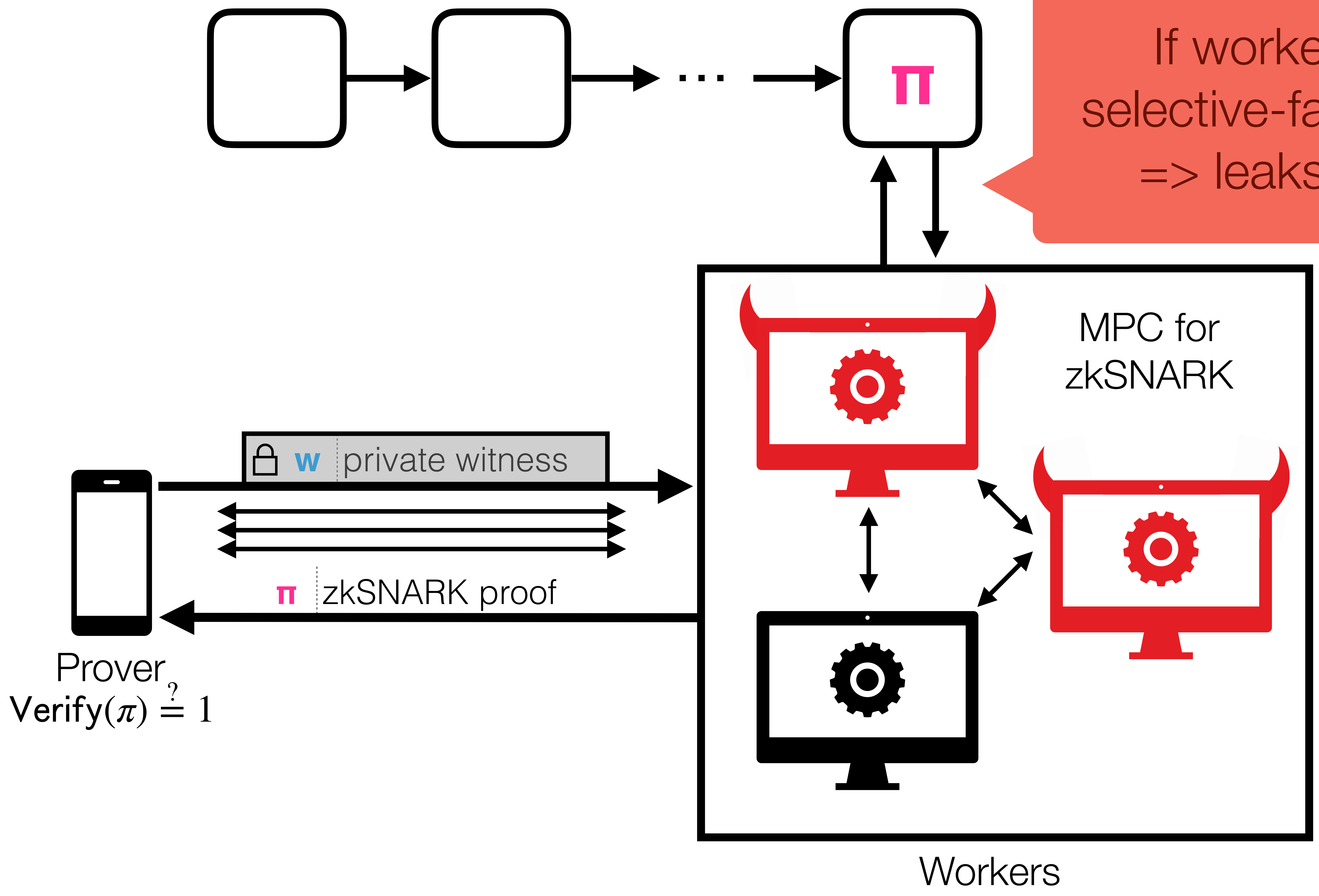
Achieving Malicious Security



Achieving Malicious Security



Achieving Malicious Security



If workers see the proof, then selective-failure attacks are possible => leaks one bit of the witness

Idea:
Use additional properties of algebraic zkSNARKs to eliminate these attacks*

Implementation

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

- We implemented Eos as a Rust library in the `arkworks` ecosystem
- Eos produces a delegation protocol for any “algebraic” zkSNARK

Experimental Setup

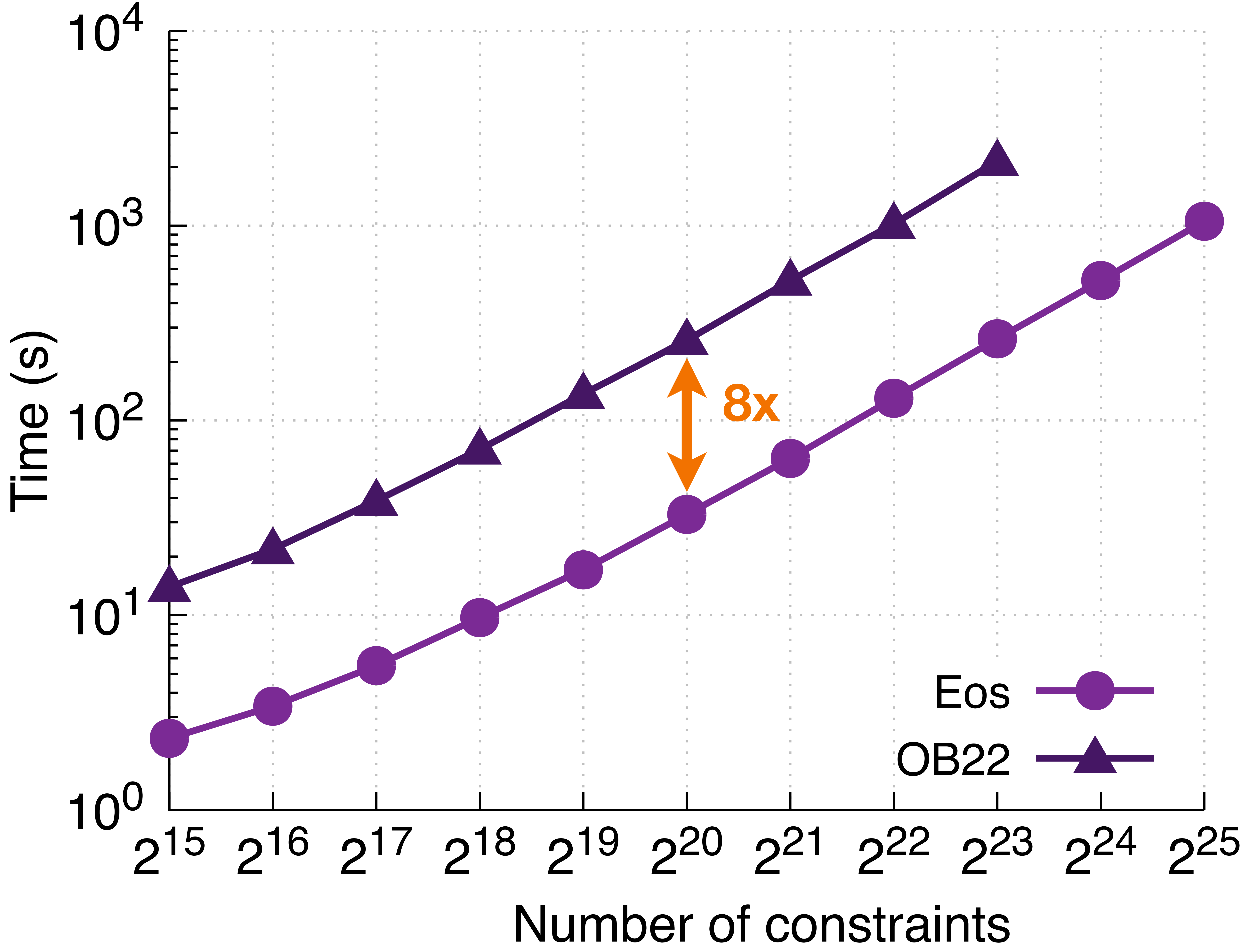
- We evaluated our protocols for the Marlin zkSNARK [CHMMVW20]
- 2 workers (AWS c5.24xlarge) in `us-west-1` and `us-east-1` regions

Eos speeds up proving time by 26x for mobile-phones

Prover	Network Throughput	Speedup	Memory reduction
r4.xlarge (AWS)	3 Gbps	9x	256x
r4.xlarge (AWS)	350 Mbps	6x	256x
Pixel 4A	350 Mbps	26x	256x

Eos vs. local proving for 2^{20} constraints

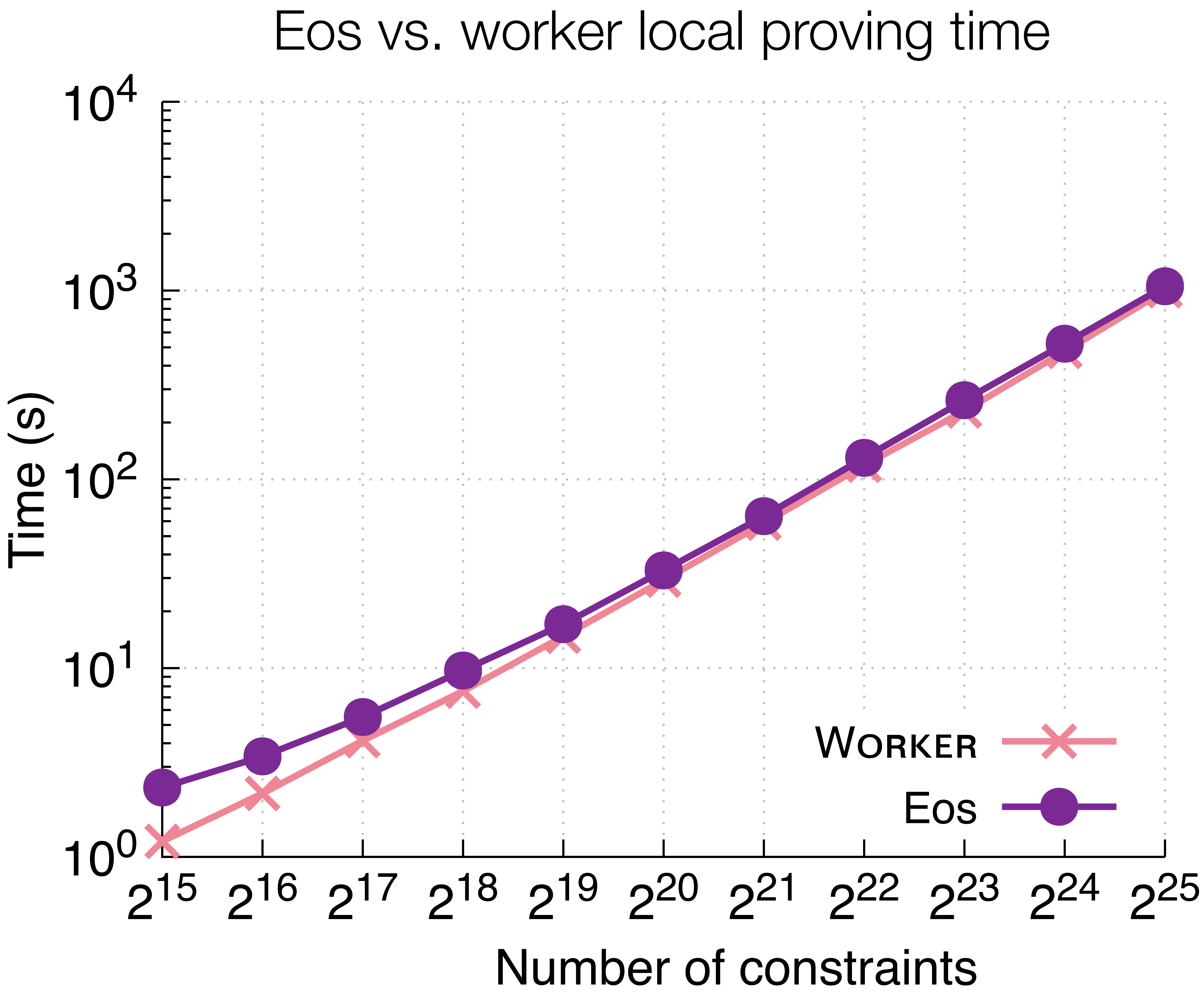
Eos is 8x faster than existing techniques



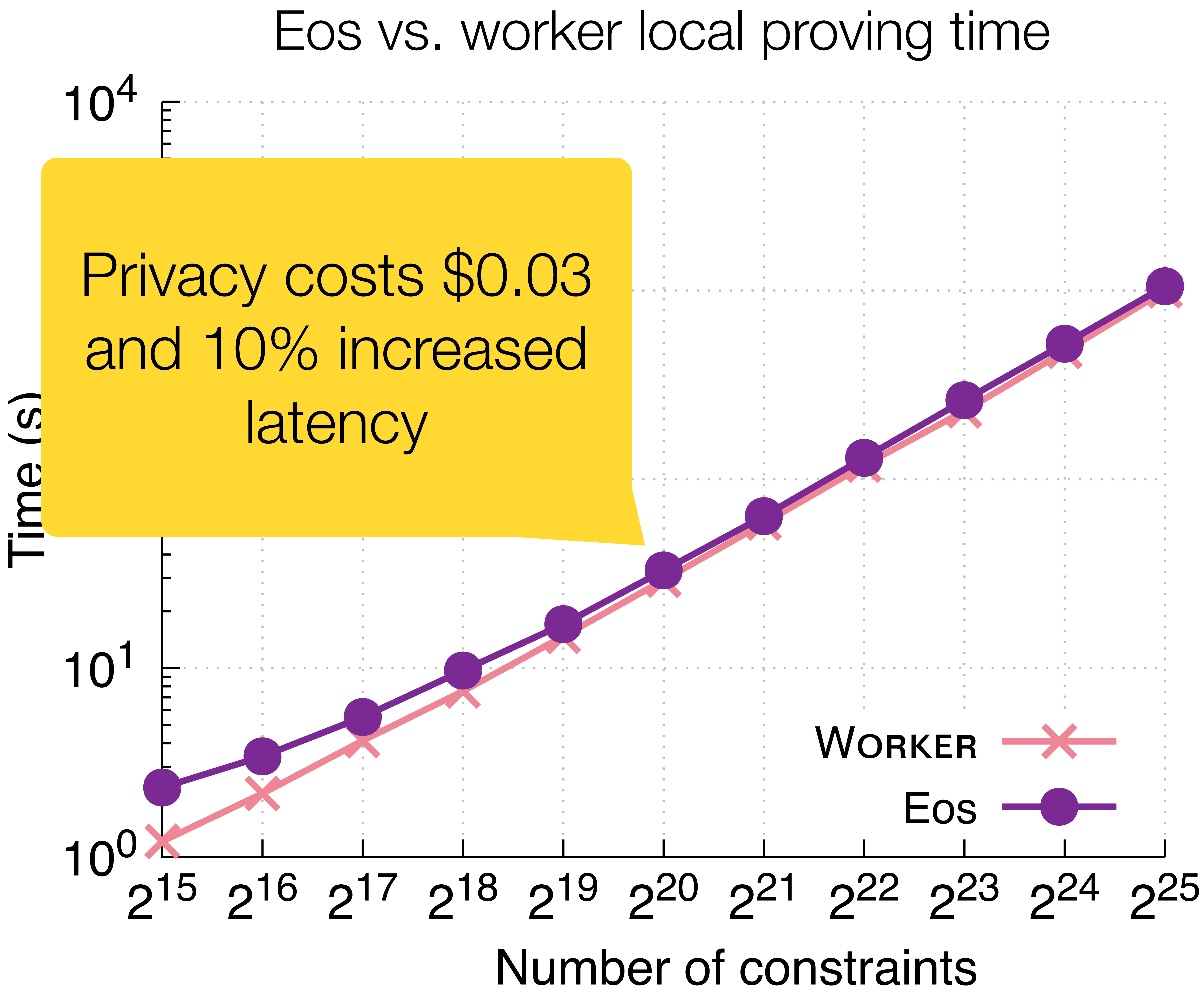
Eos is only 10% slower than insecure delegation

Eos vs. worker local proving time

Eos is only 10% slower than insecure delegation



Eos is only 10% slower than insecure delegation



Privacy costs $0.03 and 10% increased latency

Thank You!

Paper/code: www.usenix.org/conference/usenixsecurity23/presentation/chiesa

(Updated version coming soon to ePrint)

Ryan Lehmkuhl
ryanleh@mit.edu