zkSaaS

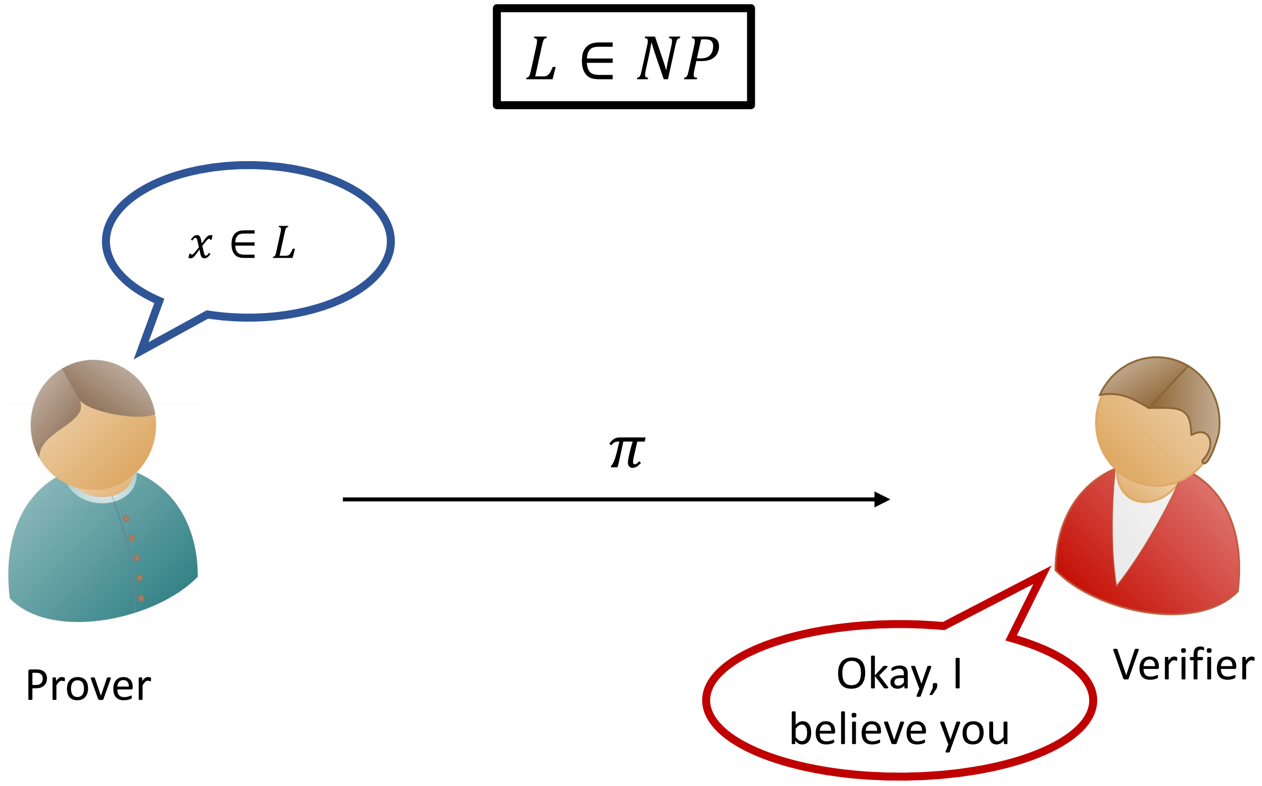
Zero-Knowledge SNARKs as a Service

Sanjam Garg Aarushi Goel Abhishek Jain Guru Vamsi Policharla Sruthi Sekar

NTT Research Johns Hopkins University Berkeley University of California

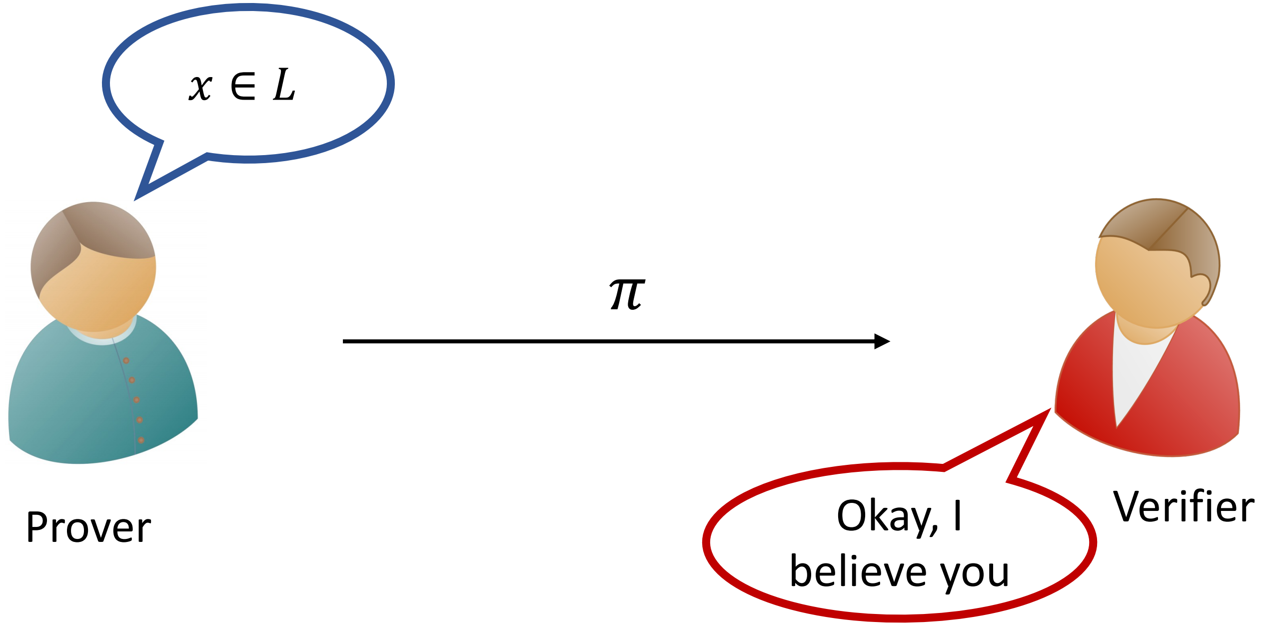
USENIX Security 2023

zk-SNARKs: Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge



zk-SNARKs: Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge

$L \in NP$



Anyone can verify

Zero knowledge

Verifier should not learn the secret witness

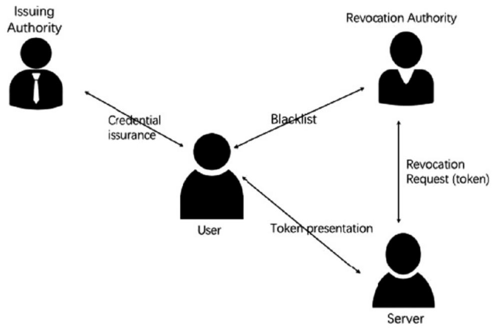
Soundness

Cheating prover cannot give accepting proof if $x \notin L$

Succinctness

Time to verify is smaller than time to compute

zk-SNARKs: Numerous Applications



Anonymous Credentials [Chaum82]

Verifiable Inference of Machine Learning [LKKO20]

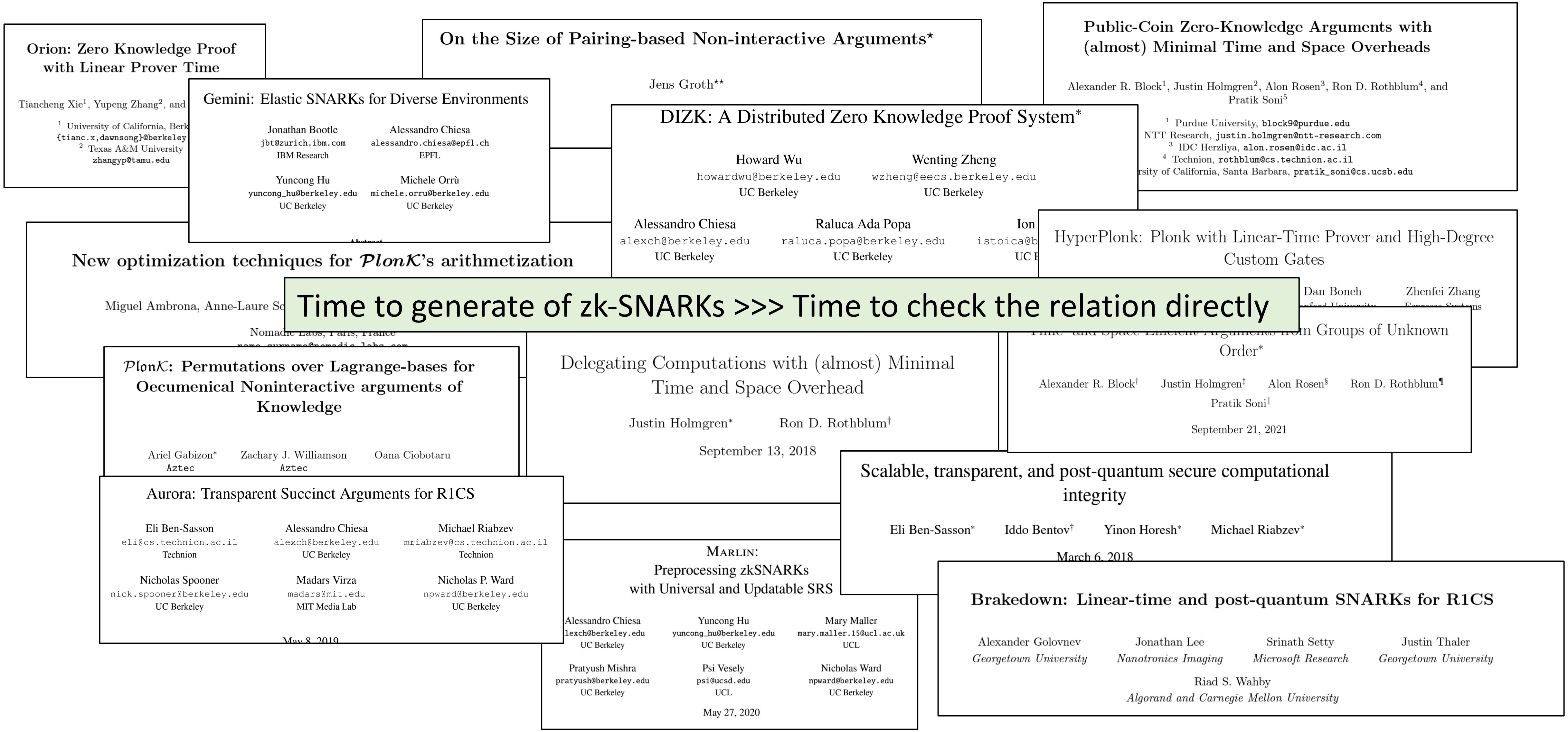
Private Smart Contracts [BCGMMW20]

Proving existence of bugs in code [HK20]

Verifying authenticity of images in media [NT16]

Privacy Respecting Cryptocurrency [BCGGMTV14]

zk-SNARKs: Lots of Work on Improving Efficiency



Gru’s Quest to be a Supervillain

Since Gru has a very long list of despicable achievements, computing a zk-SNARK will take a really long time

Prove this in zero-knowledge

I am a great villain and deserve to be part of Vicious 6

Gru: Rising Villain

Vicious 6: A prolific set of Supervillains

Can Gru Delegate zk-SNARK Computation?

aws

Violates Privacy!!: Requires leaking the entire witness to AWS

Details of his despicable achievements

π

zk-SNARK attesting to his achievements

π

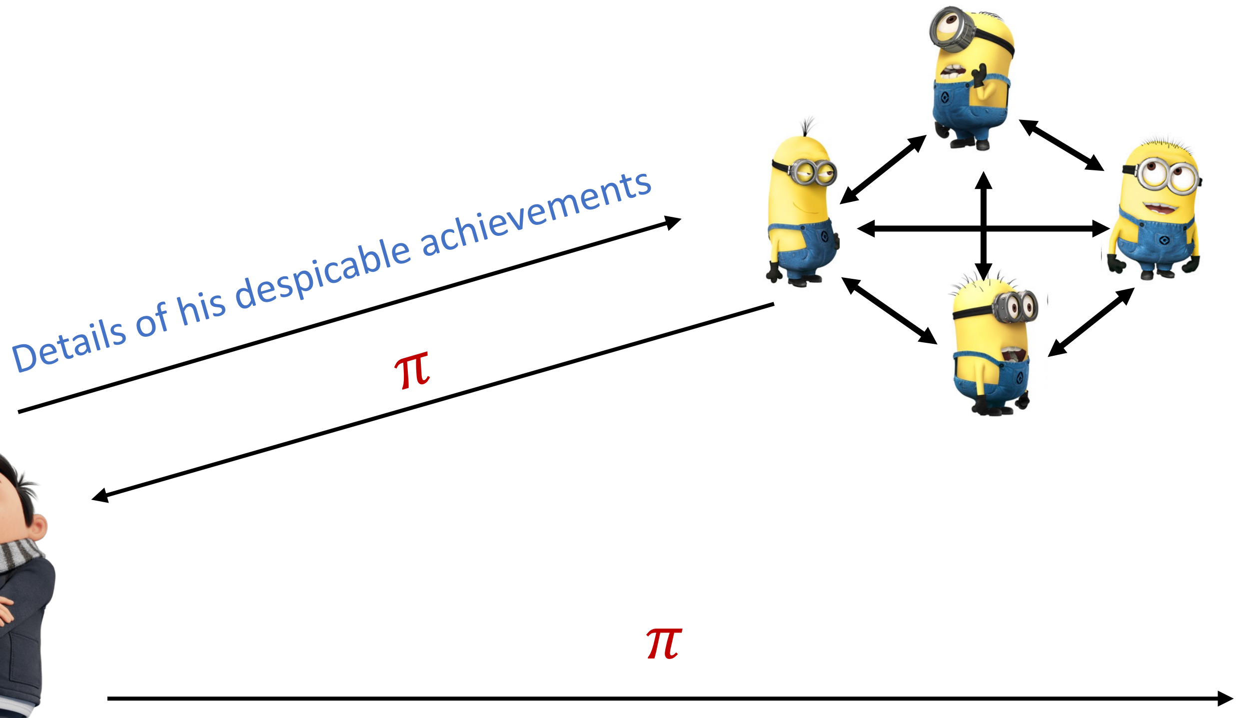
Gru: Rising Villain

Vicious 6: A prolific set of Supervillains

Can Gru Delegate zk-SNARK Computation?

Delegate to a group of minions who run an MPC to compute the zk-SNARK

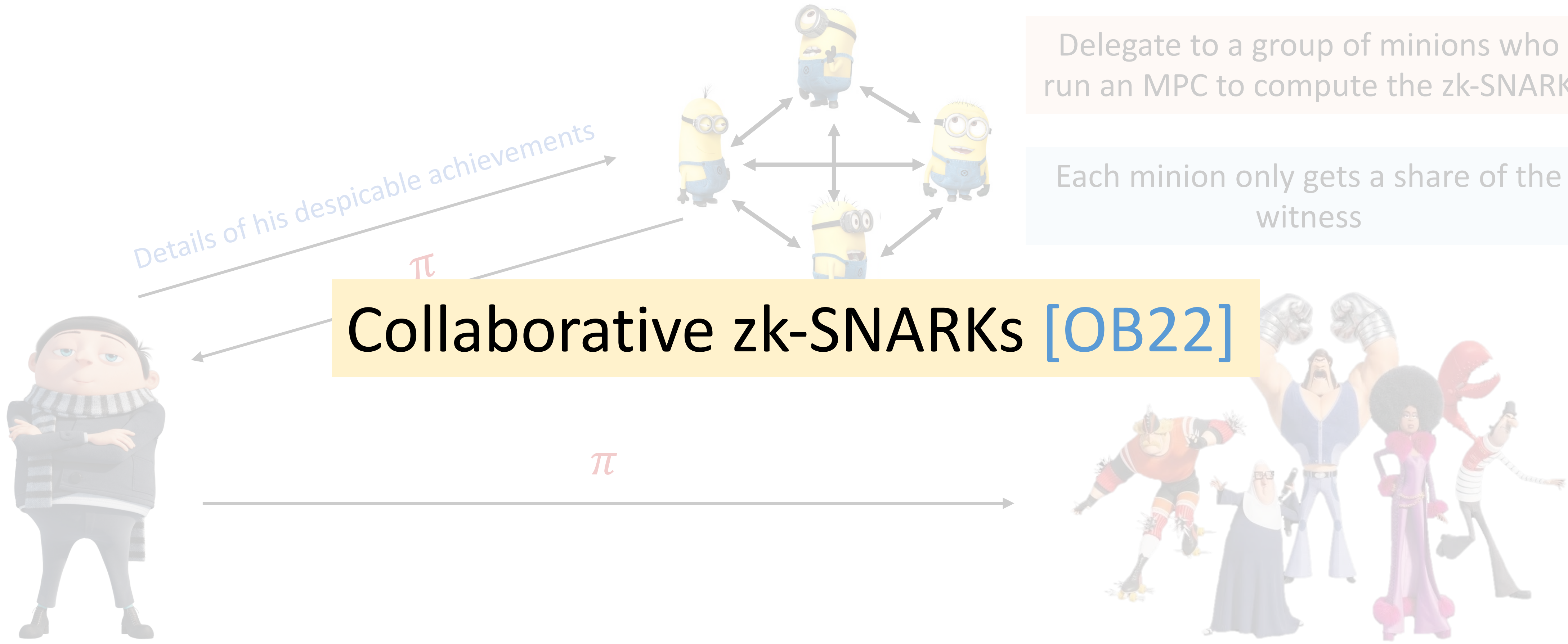
Each minion only gets a share of the witness



Gru: Rising Villain

Vicious 6: A prolific set of Supervillains

Can Gru Delegate zk-SNARK Computation?



Delegate to a group of minions who run an MPC to compute the zk-SNARK

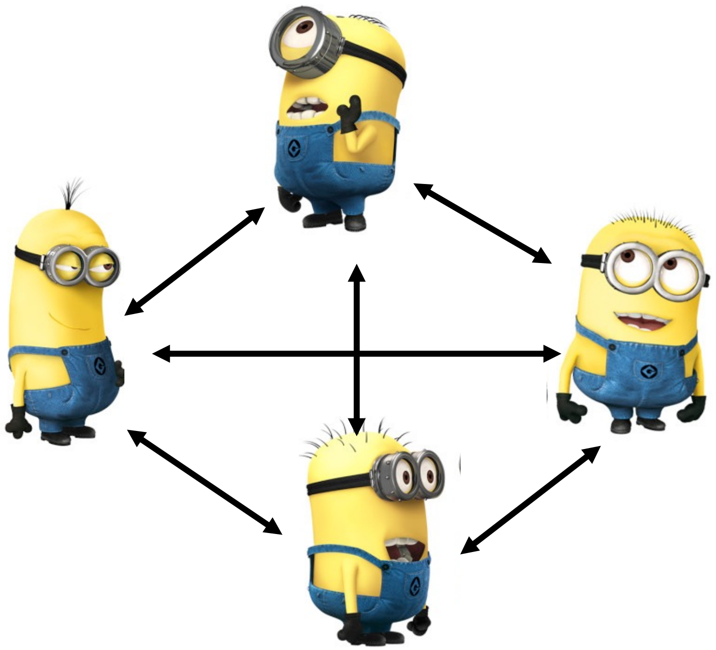
Each minion only gets a share of the witness

Collaborative zk-SNARKs [OB22]

Gru: Rising Villain

Vicious 6: A prolific set of Supervillains

Collaborative zk-SNARKs [OB22]



Efficient MPC for computing zk-SNARKs

Privacy

Each party does work proportional to a single prover

Wasteful

[WZCAS18] leverage parallelism to distribute work across machines in a compute cluster to get faster proof generation

Not Privacy Preserving

Our Goal

Better utilization of resources of the parties in collaborative zk-SNARKs, for faster proof generation, in a privacy-preserving manner

Our Results: zkSaaS

Framework

For privacy preserving delegation of zk-SNARK computation.
Each servers is expected to run for a shorter duration than a single local prover.

Design

Design zkSaaS for Groth16 [Gro16], Marlin [CHMMVW20] and Plonk [GWC19].

Implementation

Implement a prototype of zkSaaS for Groth16, Plonk and get
$\approx$ $22\times$ speed-up when run with 128 parties for $2^{21} - 2^{25}$ constraints

zkSaaS Framework

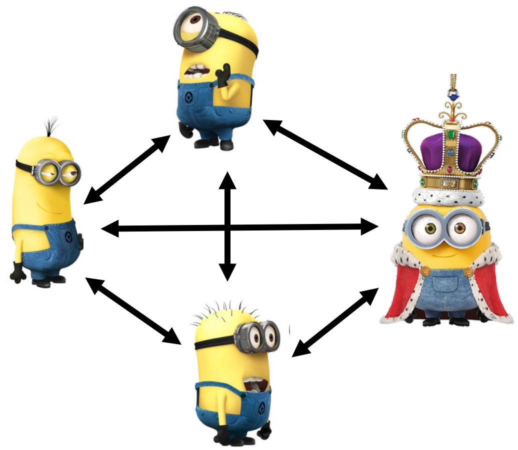
Typical zk-SNARKs

Step 1:	Computing Extended Witness
Step 2:	Generating Proof (Cryptographic Operations + Field Operations)

Pre-Processing: each server gets a part of the correlated randomness

Secret Shares "Extended Witness" →

← π



Client computes Step 1

Servers collectively compute Step 2

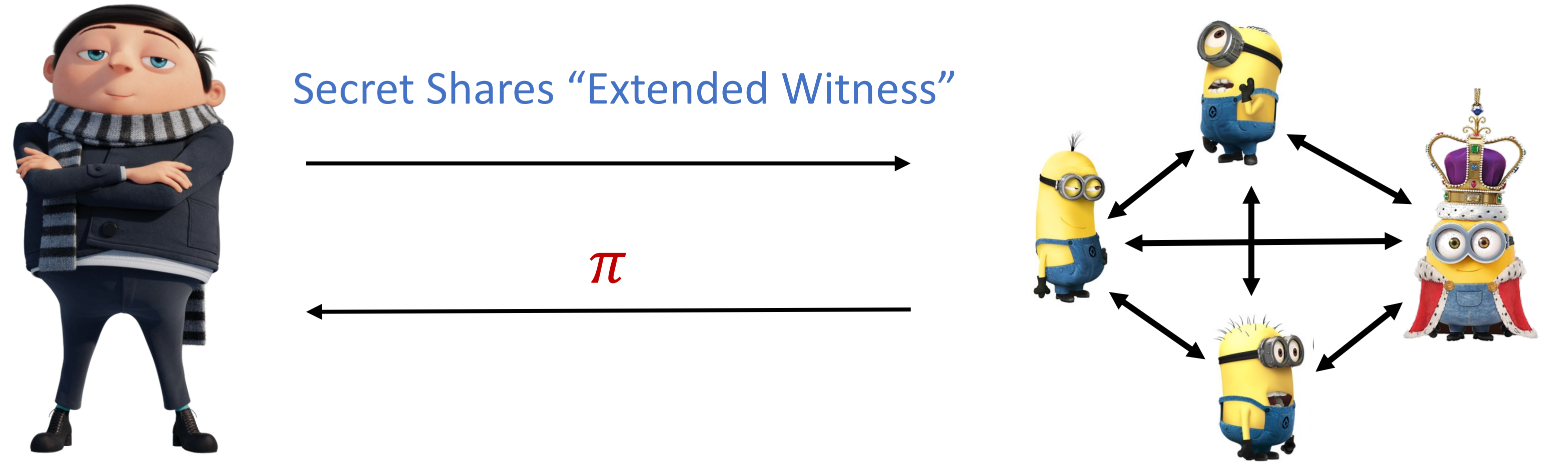
zkSaaS Framework

<u>Typical zk-SNARKs</u>

Step 1: Computing Extended Witness

Step 2: Generating Proof
(Cryptographic Operations + Field Operations)

<u>Pre-Processing:</u> each server gets a part of the correlated randomness



Cryptographic Operations get equally divided amongst all servers

Field Operations get equally divided amongst small servers. King does work linear in the number of field operations.

Client computes Step 1

Servers collectively compute Step 2

Applicability of **zkSaaS**

To aid users with small devices

For extremely large computations

Designing zkSaaS

General Template [OB22]

Identify basic building blocks in zk-SNARKs

+

Design custom MPC protocols for each building block with the required efficiency

Combine them to get a **zkSaaS** for the corresponding zk-SNARK

Building Blocks in Groth16, Marlin, Plonk

Multi-Scalar Multiplications (MSM)

$$F(g_1, \alpha_1, \dots, g_m, \alpha_m) = \prod_{i \in [m]} {g_i}^{\alpha_i}$$

Partial Products

$$F(x_1, \dots, x_m) = \left(\prod_{i \in [j]} x_i \right)_{j \in [m]}$$

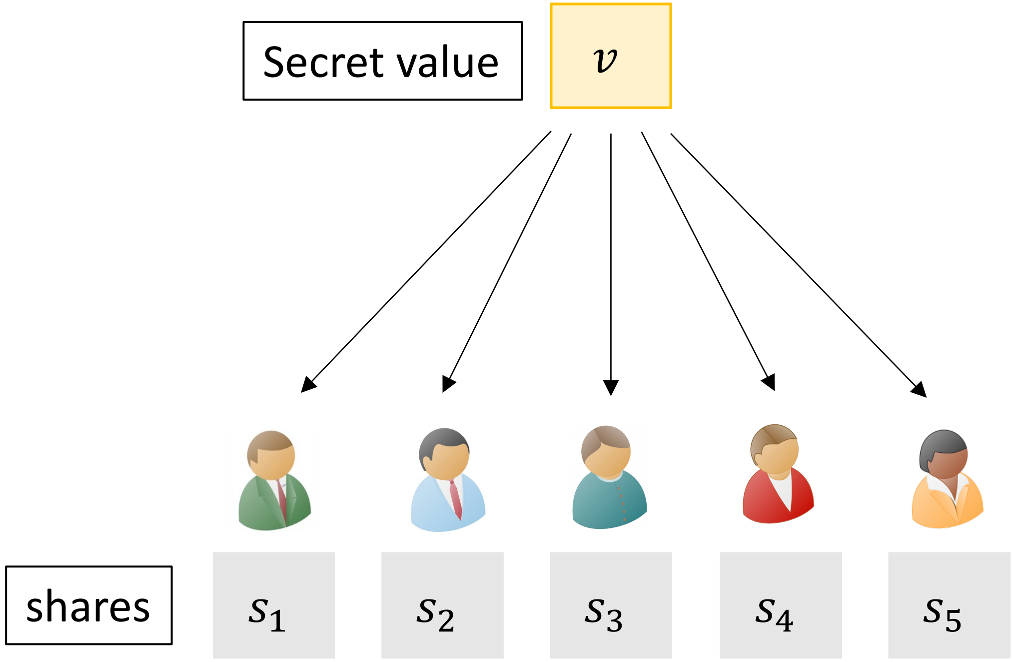
Fast Fourier Transform (FFT)

For converting between coefficient and evaluation representation of polynomials

Polynomial Multiplication and Division

A combination of addition, multiplication and FFT operations

Packed Secret Sharing (PSS) [FY92]

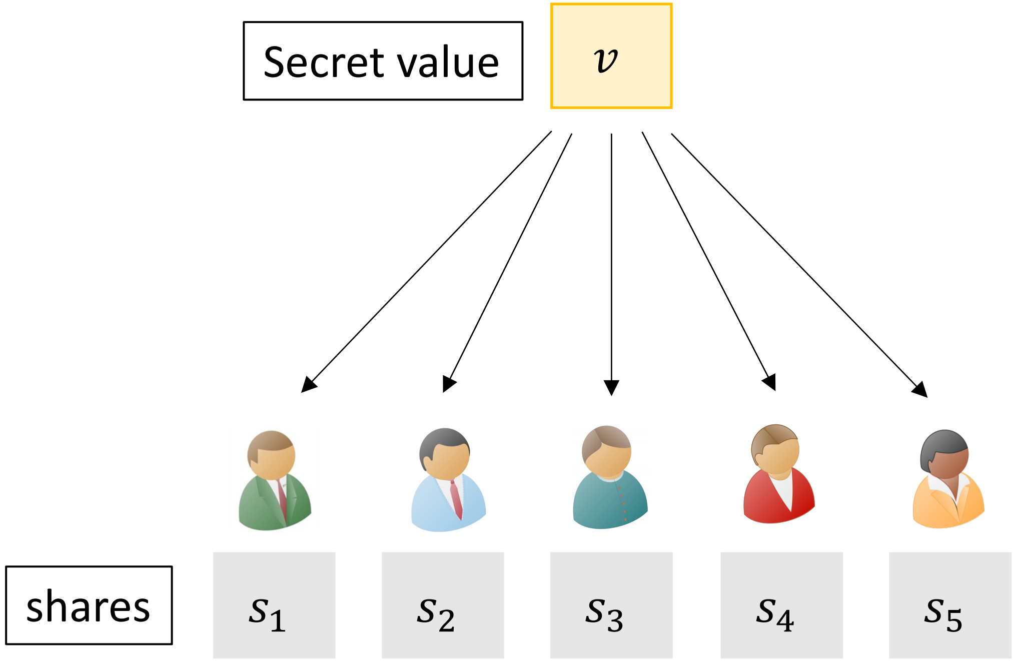


Regular Secret Sharing

1 Value → n shares

Corruption threshold: $t < \frac{n}{2}$

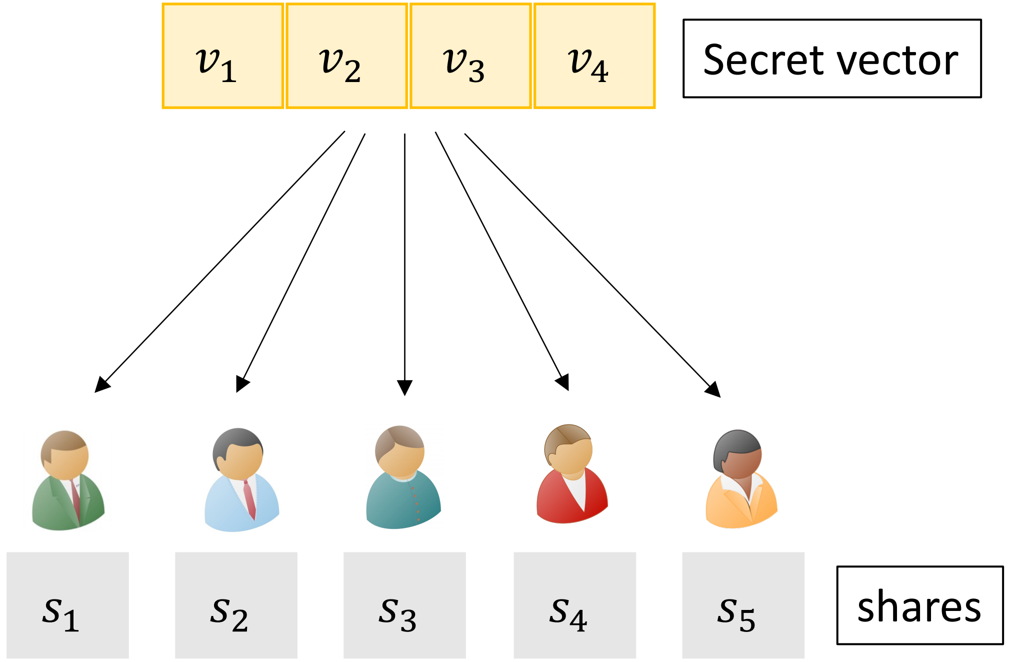
Packed Secret Sharing (PSS) [FY92]



Regular Secret Sharing

1 Value → n shares

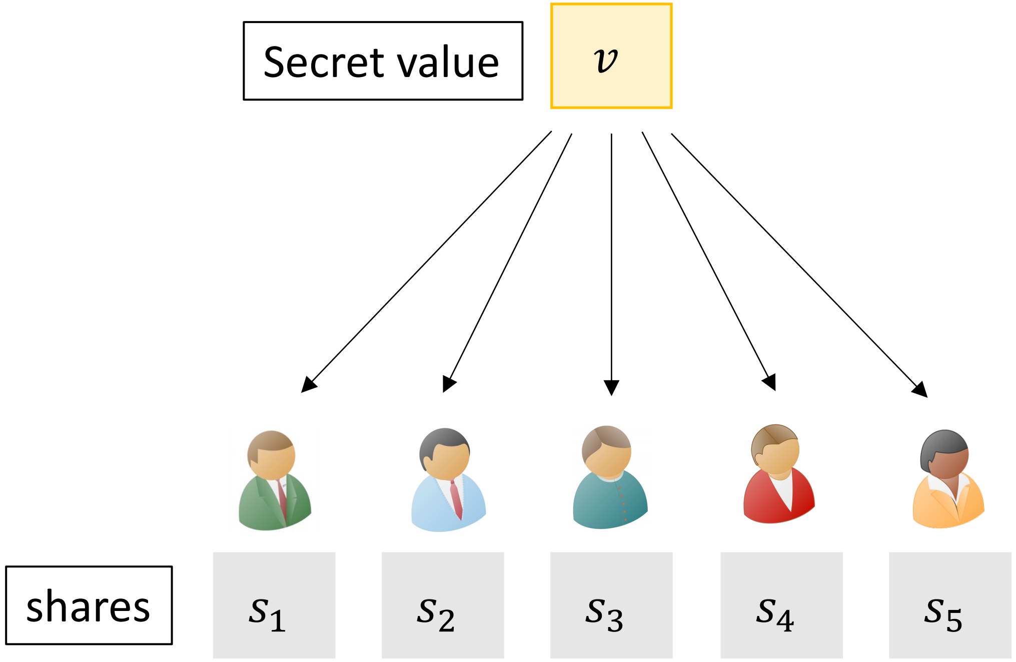
Corruption threshold: $t < \frac{n}{2}$



Packed Secret Sharing

$O(n)$ Values → n shares

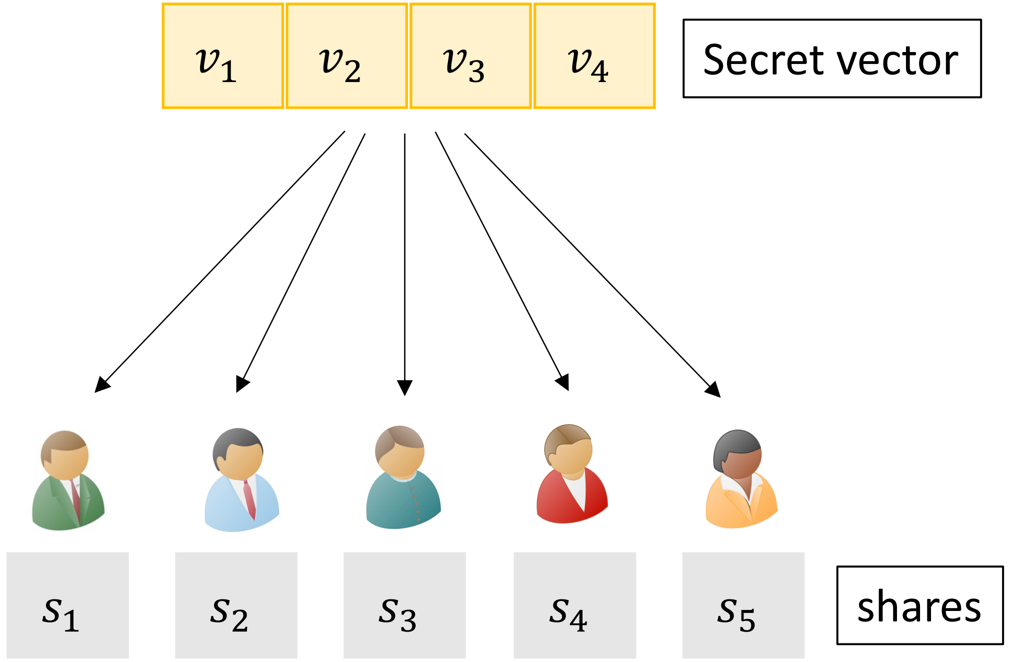
Packed Secret Sharing (PSS) [FY92]



Regular Secret Sharing

1 Value → n shares

Corruption threshold: $t < \frac{n}{2}$



Packed Secret Sharing

$O(n)$ Values → n shares

Corruption threshold $t < n(\frac{1}{2} - \frac{1}{\varepsilon})$

Experimental Results

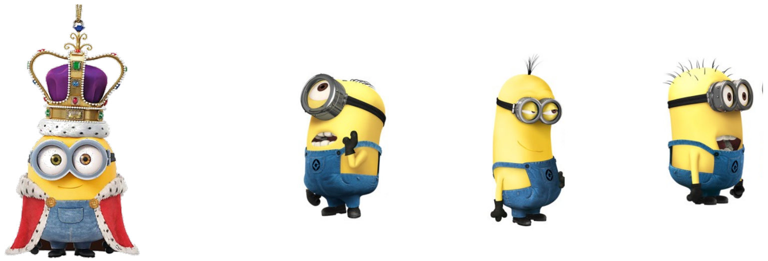
zkSaaS for Groth16: Setup

N1 GCP Instances

Local Prover

1vCPU and 4 GB RAM

zkSaaS Servers



96vCPU and 128 GB RAM

1vCPU and 2 GB RAM each

zkSaaS for Groth16

No. of Servers = 128
Packing constant = 32
No. of corrupt servers =31

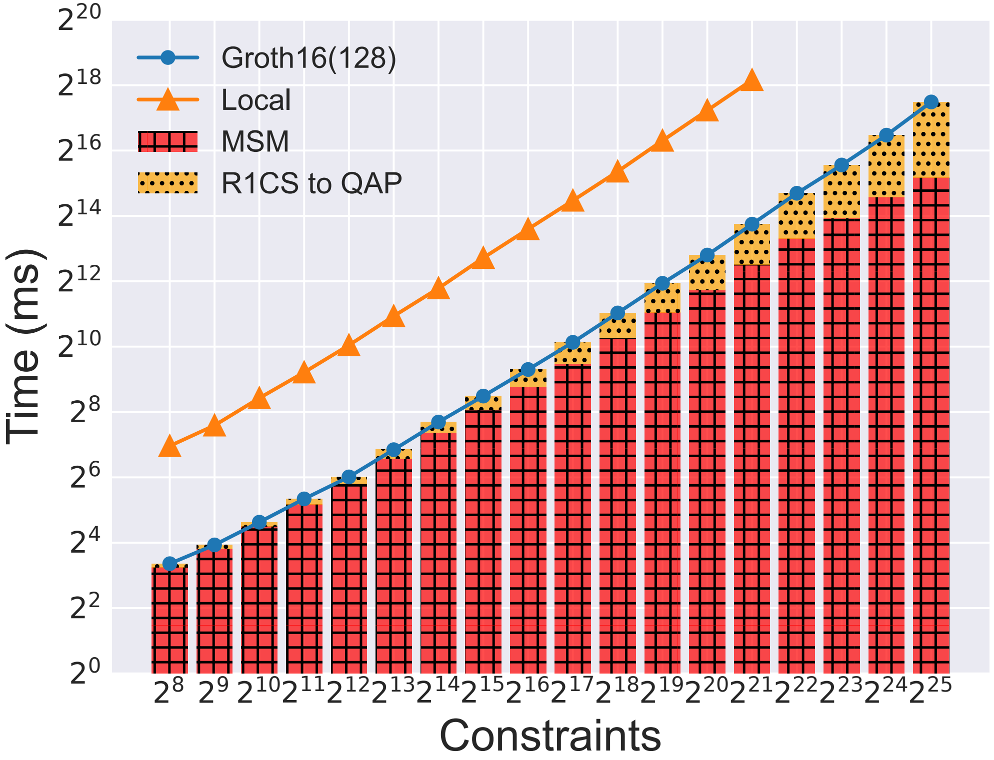
Local Prover

1vCPU and 4 GB RAM

zkSaaS Servers

96vCPU and 128 GB RAM

1vCPU and 2 GB RAM each



Memory Exhaustion

Weak servers can handle 16 times more constraints than consumer machine before running out of memory

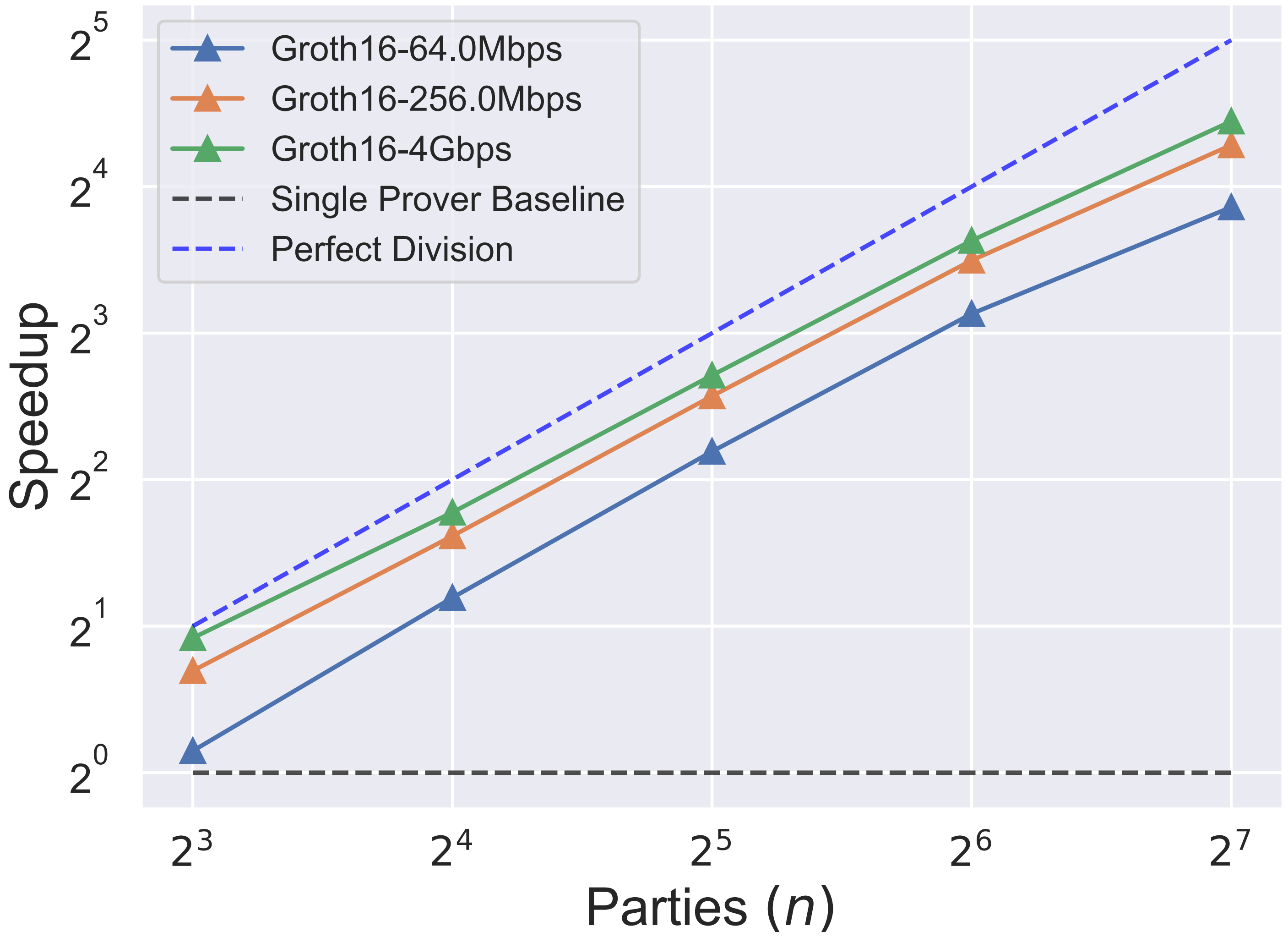
Running Time

We get ≈ 22× speed-up over consumer machine

Why not 32 times?

1. FFT doesn’t achieve equal division of work
2. Sub-optimal use of Pippenger’s algorithm for MSMs

zkSaaS for Groth16



Local Prover

1vCPU and 4 GB RAM

zkSaaS Servers

96vCPU and 128 GB RAM

1vCPU and 2 GB RAM each

No. of Constraints = 2^{19}
Packing constant = n/4
No. of corrupt servers = n/4

Paper

Code

Thanks!

aarushi.goel@ntt-research.com

https://aarushigoel.github.io/