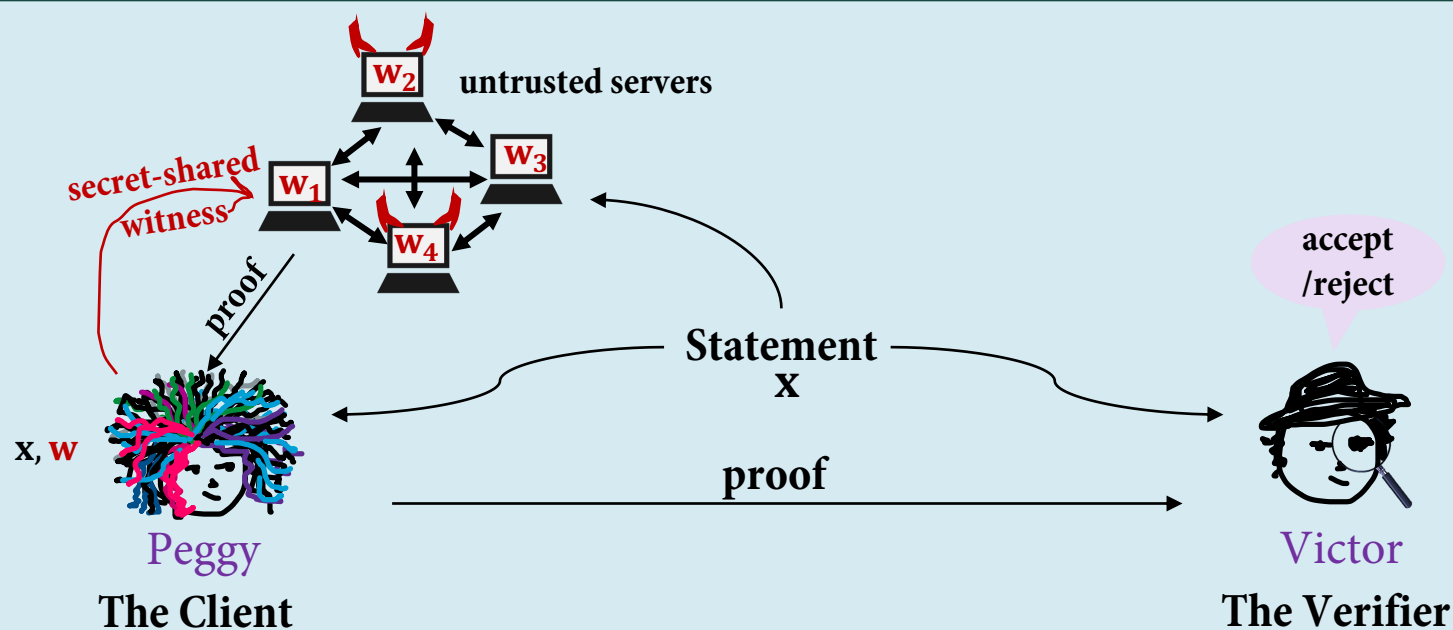




zkSaaS

Zero Knowledge SNARKs as a Service

USENIX Security 2023

Sanjam Garg



Aarushi Goel



Abhishek Jain



Guru Vamsi Policharla



Sruthi Sekar

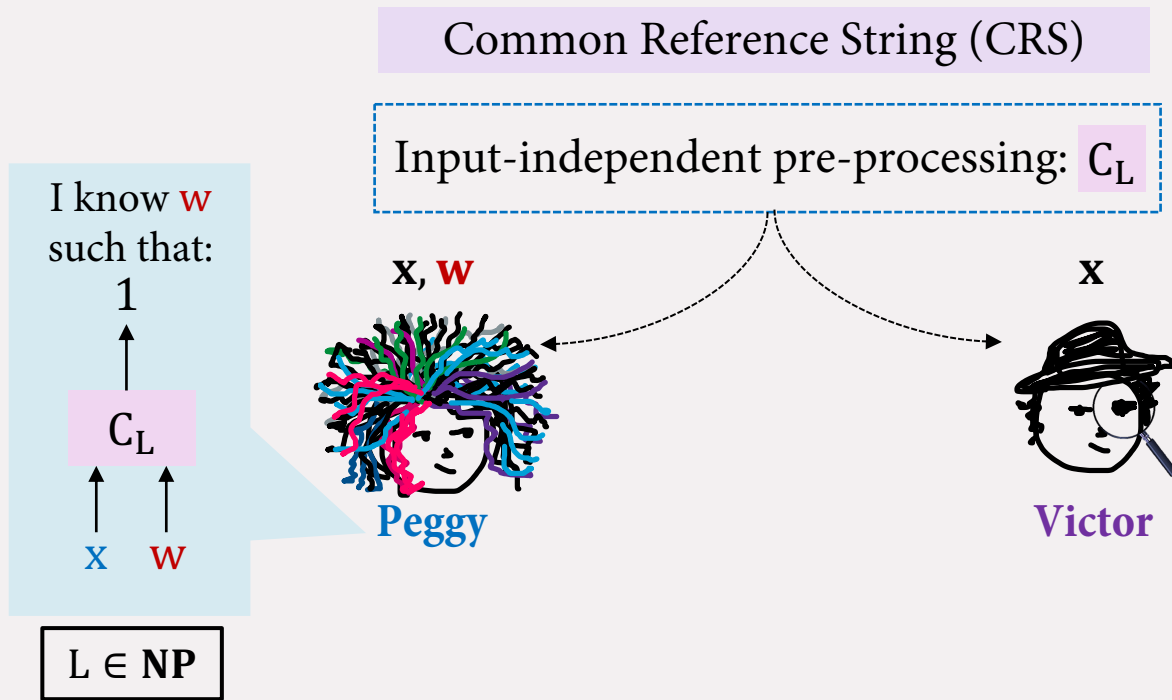


zkSNARKs: Zero-Knowledge Succinct Non-interactive Arguments of Knowledge



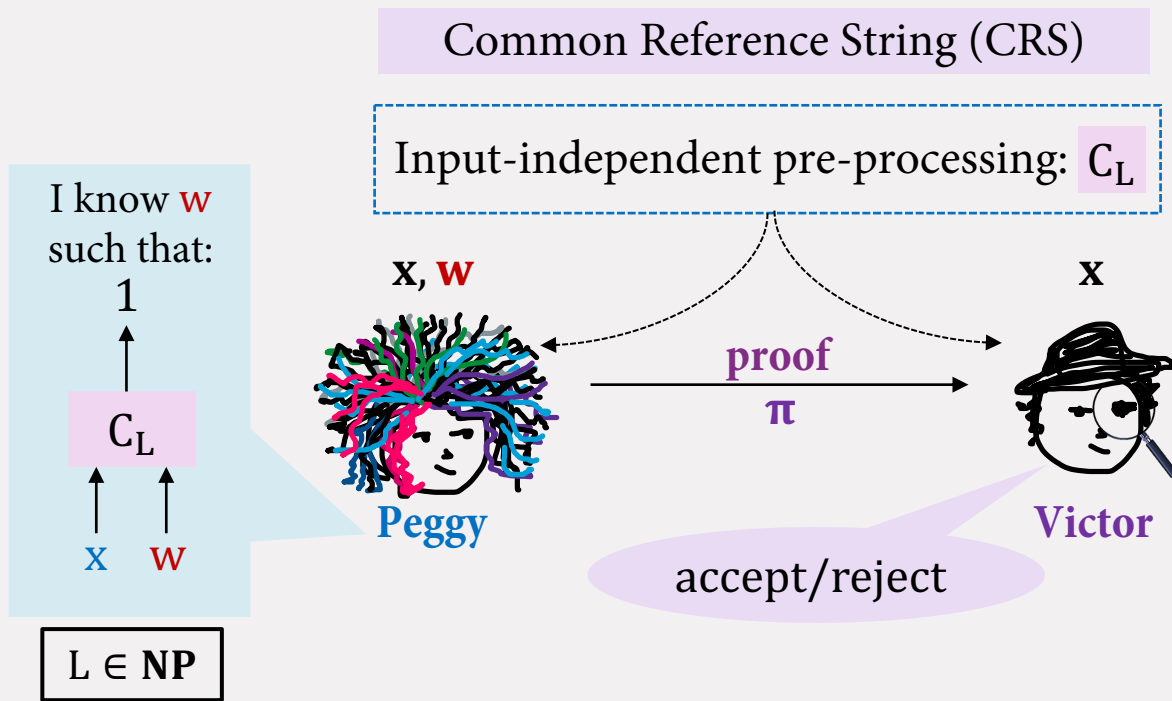
Intuition: *succinct* proof that a certain statement is true.
E.g.: “I know an m such that $\text{SHA256}(m) = 0$ ”

zkSNARKs: Zero-Knowledge Succinct Non-interactive Arguments of Knowledge



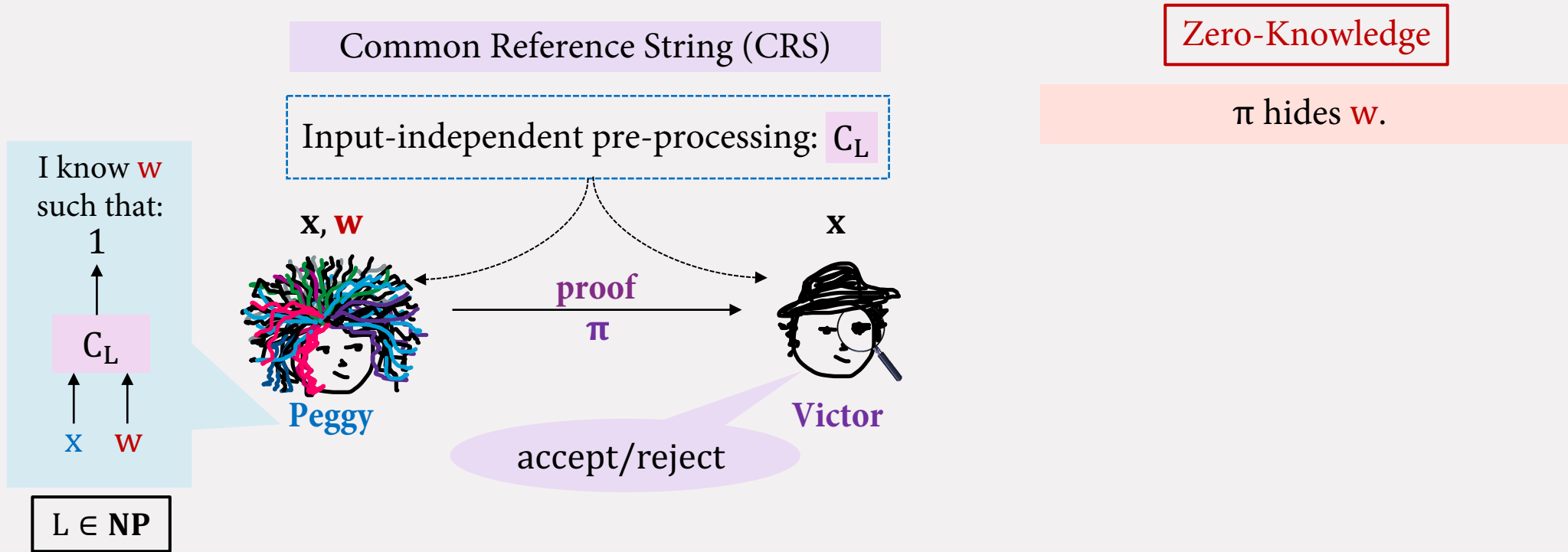
Intuition: *succinct* proof that a certain statement is true.
E.g.: “I know an m such that $SHA256(m) = 0$ ”

zkSNARKs: Zero-Knowledge Succinct Non-interactive Arguments of Knowledge



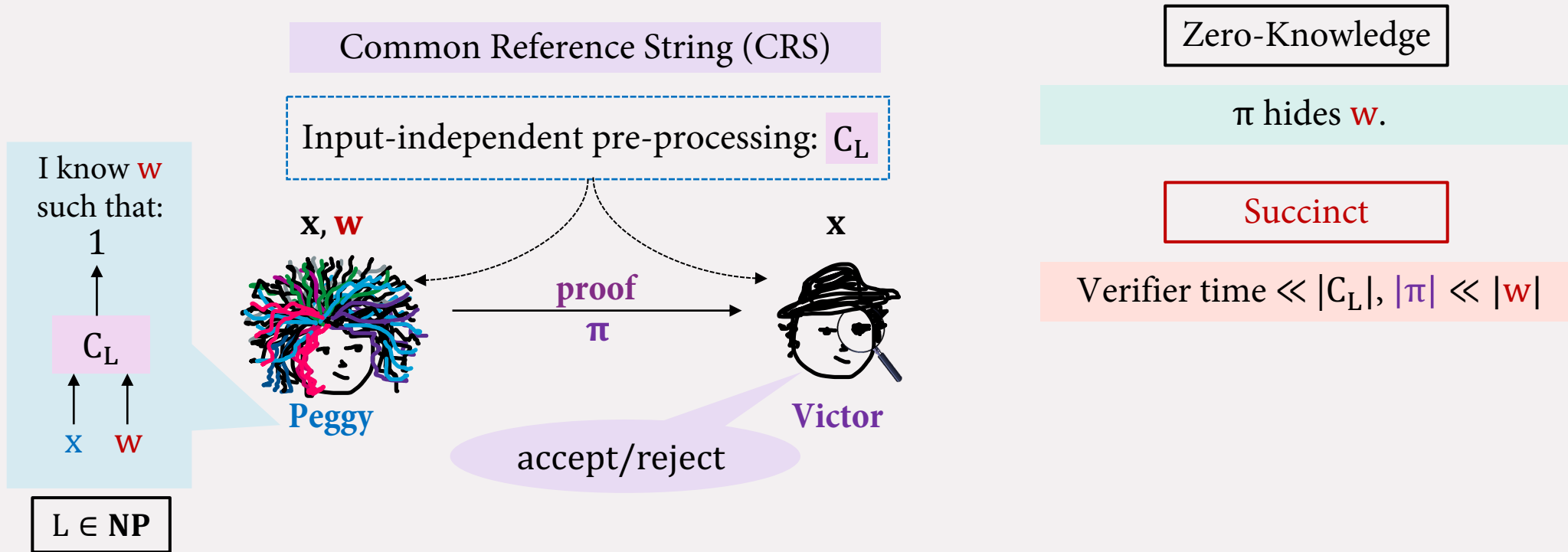
Intuition: *succinct* proof that a certain statement is true.
E.g.: "I know an m such that $SHA256(m) = 0$ "

zkSNARKs: **Zero-Knowledge** Succinct Non-interactive Arguments of Knowledge



Intuition: *succinct* proof that a certain statement is true.
E.g.: "I know an m such that $SHA256(m) = 0$ "

zkSNARKs: Zero-Knowledge **Succinct** Non-interactive Arguments of Knowledge

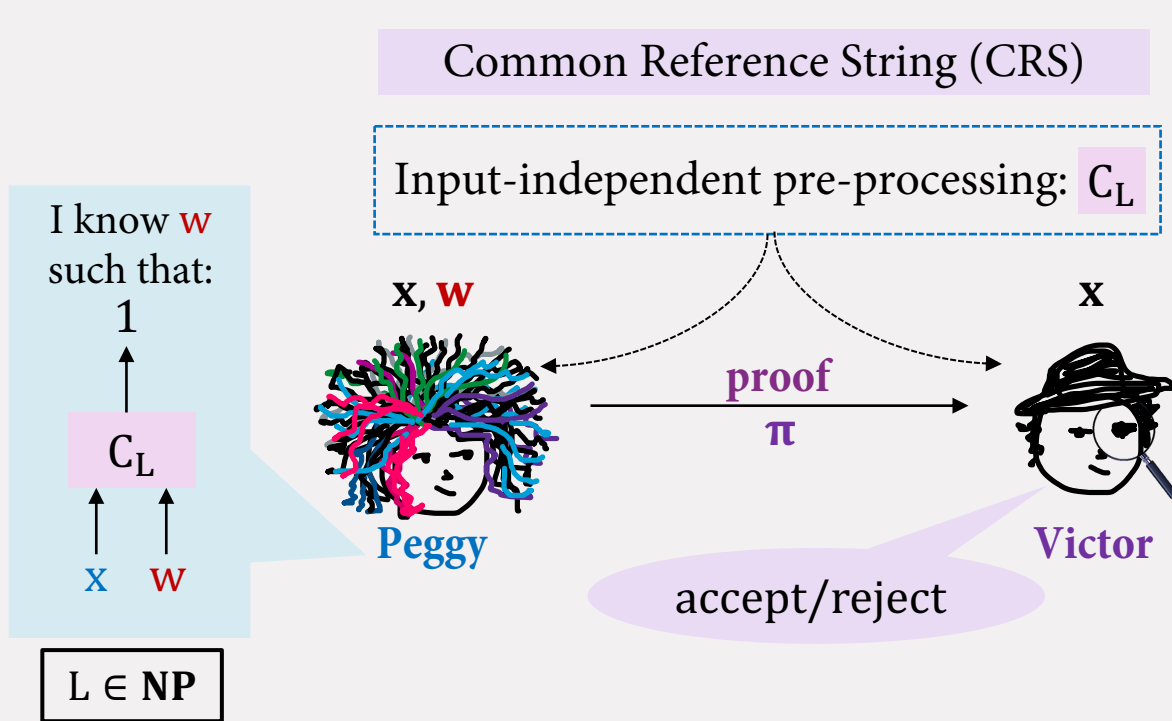


Intuition: *succinct* proof that a certain statement is true.

E.g.: "I know an m such that $\text{SHA256}(m) = 0$ "

→ **Short Proof and fast to verify!**

zkSNARKs: Zero-Knowledge Succinct Non-interactive Arguments of Knowledge



Zero-Knowledge

π hides w .

Succinct

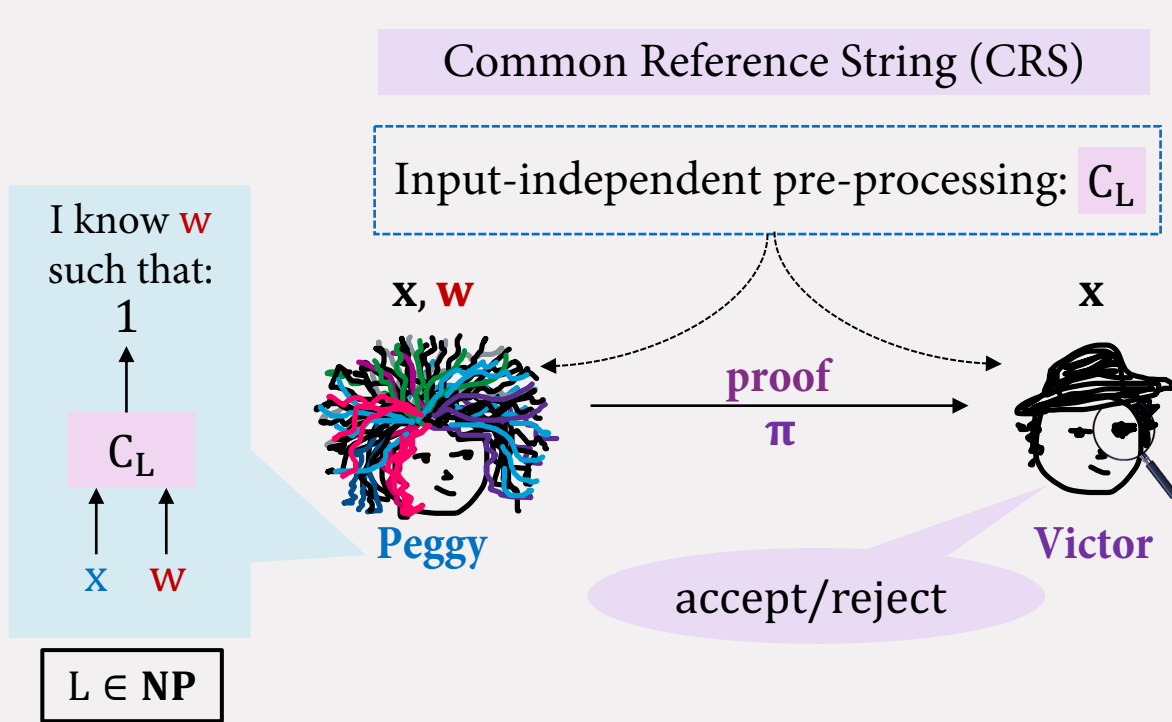
Verifier time $\ll |C_L|, |\pi| \ll |w|$

Non-interactive

Anyone can verify

Intuition: *succinct* proof that a certain statement is true.
E.g.: "I know an m such that $\text{SHA256}(m) = 0$ "

zkSNARKs: Zero-Knowledge Succinct Non-interactive Arguments of Knowledge



Intuition: *succinct* proof that a certain statement is true.
E.g.: "I know an m such that $\text{SHA256}(m) = 0$ "

Zero-Knowledge

π hides w .

Succinct

Verifier time $\ll |C_L|, |\pi| \ll |w|$

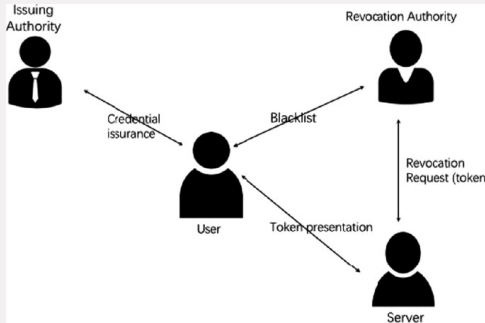
Non-interactive

Anyone can verify

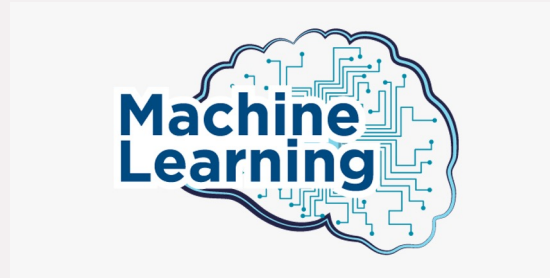
Soundness

π will not be accepted if $\nexists w$.

zk-SNARKs: Applications



Anonymous Credentials
[Chaum82]



Verifiable Inference of
Machine Learning [LKKO20]



Private Smart Contracts
[BCGMMW20]



Proving the existence
of bugs in code [HK20]



Fighting Disinformation
[NT16]



Privacy Respecting
Cryptocurrency [BCGGMTV14]

Flurry of Research on Efficient zkSNARKs

Orion: Zero Knowledge Proof with Linear Prover Time

Tiancheng Xie¹, Yupeng Zhang², and Dawn Song¹

¹ University of California, Berkeley

Gemini: Elastic SNARKs for Diverse Environments

Jonathan Bootle
jbt@zurich.ibm.com
IBM Research

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Yuncong Hu
yuncong_hu@berkeley.edu
UC Berkeley

Michele Orrù
michele.orrù@berkeley.edu
UC Berkeley

PlonK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge

Ariel Gabizon*
Aztec

Zachary J. Williamson
Aztec

Oana Ciobotaru

August 17, 2022

On the Size of Pairing-based Non-interactive Arguments*

Jens Groth**

University College London, UK
j.groth@ucl.ac.uk

Public-Coin Zero-Knowledge Arguments with (almost) Minimal Time and Space Overheads

Alexander R. Block¹, Justin Holmgren², Alon Rosen³, Ron D. Rothblum⁴, and Pratik Soni⁵

¹ Purdue University, block9@purdue.edu

² NTT Research, justin.holmgren@ntt-research.com

³ IDC Herzliya, alon.rosen@idc.ac.il

⁴ Technion, ron@technion.ac.il

⁵ Princeton University, pratiksoni@princeton.edu

New optimization techniques for *PlonK*'s arithmetization

Miguel Ambrona, Anne-Laure Schmitt, Raphael R. Toledo, and Danny Willems

Delegating Computations with (almost) Minimal Time and Space Overhead

Justin Holmgren*

Ron D. Rothblum†

September 13, 2018

DIZK: A Distributed Zero Knowledge Proof System*

Howard Wu
howardwu@berkeley.edu
UC Berkeley

Wenting Zheng
wzheng@eecs.berkeley.edu
UC Berkeley

Alessandro Chiesa
alexch@berkeley.edu

Raluca Ada Popa
raluca.popa@berkeley.edu

Ion Stoica
istoica@berkeley.edu

Aurora: Transparent Succinct Arguments for R1CS

Eli Ben-Sasson
eli@cs.technion.ac.il
Technion

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Michael Riabzev
mriabzev@cs.technion.ac.il
Technion

Nicholas Spooner
nick.spooner@berkeley.edu
UC Berkeley

Madars Virza
madars@mit.edu
MIT Media Lab

Nicholas P. Ward
npward@berkeley.edu
UC Berkeley

May 8, 2019

HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates

Binyi Chen
Espresso Systems

Benedikt Bünz
Stanford University,

Dan Boneh
Stanford University

Zhenfei Zhang
Espresso Systems

Time- and Space-Efficient Arguments from Groups of Unknown Order*

Alexander R. Block†

Justin Holmgren‡

Alon Rosen§

Ron D. Rothblum¶

Pratik Soni‡

September 21, 2021

Scalable, transparent, and post-quantum secure computational integrity

Eli Ben-Sasson*

Ido Bentov†

Yinon Horesh*

Michael Riabzev*

MARLIN: Preprocessing zkSNARKs with Universal and Updatable SRS

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Yuncong Hu
yuncong_hu@berkeley.edu
UC Berkeley

Mary Maller
mary.maller.15@ucl.ac.uk
UCL

Pratyush Mishra
pratyush@berkeley.edu
UC Berkeley

Psi Vesely
psi@ucl.ac.uk
UCL

Nicholas Ward
npward@berkeley.edu
UC Berkeley

May 27, 2020

Brakedown: Linear-time and post-quantum SNARKs for R1CS

Alexander Golovnev
Georgetown University

Jonathan Lee
Nanotronics Imaging

Srinath Setty
Microsoft Research

Justin Thaler
Georgetown University

Riad S. Wahby
Algorand and Carnegie Mellon University

Flurry of Research on Efficient zkSNARKs

Orion: Zero Knowledge Proof with Linear Prover Time

Tiancheng Xie¹, Yupeng Zhang², and Dawn Song¹

¹ University of California, Berkeley

Gemini: Elastic SNARKs for Diverse Environments

Jonathan Bootle
jbt@zurich.ibm.com
IBM Research

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Yuncong Hu
yuncong_hu@berkeley.edu
UC Berkeley

Michele Orrù
michele.orrù@berkeley.edu
UC Berkeley

PlonK: Permutations over Lagrange-basis for Oecumenical Noninteractive arguments of Knowledge

Ariel Gabizon*
Aztec

Zachary J. Williamson
Aztec

Oana Ci

August 17, 2022

On the Size of Pairing-based Non-interactive Arguments*

Jens Groth**

University College London, UK
j.groth@ucl.ac.uk

New optimization techniques for *PlonK*'s arithmetization

Main parameters of interest

- Prover Time (>> Relation circuit time)
- Verifier Time/ Proof size
- Setup (structured/transparent)

Public-Coin Zero-Knowledge Arguments with (almost) Minimal Time and Space Overheads

Alexander R. Block¹, Justin Holmgren², Alon Rosen³, Ron D. Rothblum⁴, and Pratik Soni⁵

¹ Purdue University, block9@purdue.edu

² NTT Research, justin.holmgren@ntt-research.com

³ IDC Herzliya, alon.rosen@idc.ac.il

⁴ MIT, ron@csail.mit.edu

⁵ UC Berkeley, pratiksoni@berkeley.edu

HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates

Benedikt Bünz
Stanford University,

Dan Boneh
Stanford University

Zhenfei Zhang
Espresso Systems

Efficient Arguments from Groups of Unknown Order*

Justin Holmgren[†]

Alon Rosen[§]

Ron D. Rothblum[¶]

Pratik Soni[‡]

September 21, 2021

Scalable, transparent, and post-quantum secure computational integrity

Eli Ben-Sasson*

Iddo Bentov[†]

Yinon Horesh*

Michael Riabzev*

MARLIN: Preprocessing zkSNARKs with Universal and Updatable SRS

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Yuncong Hu
yuncong_hu@berkeley.edu
UC Berkeley

Mary Maller
mary.maller.15@ucl.ac.uk
UCL

Pratyush Mishra
pratyush@berkeley.edu
UC Berkeley

Psi Vesely
psi@ucsd.edu
UCL

Nicholas Ward
npward@berkeley.edu
UC Berkeley

May 27, 2020

Aurora: Transparent Succinct Arguments for R1CS

Eli Ben-Sasson
eli@cs.technion.ac.il
Technion

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Michael Riabzev
mriabzev@cs.technion.ac.il
Technion

Nicholas Spooner
nick.spooner@berkeley.edu
UC Berkeley

Madars Virza
madars@mit.edu
MIT Media Lab

Nicholas P. Ward
npward@berkeley.edu
UC Berkeley

May 8, 2019

Brakedown: Linear-time and post-quantum SNARKs for R1CS

Alexander Golovnev
Georgetown University

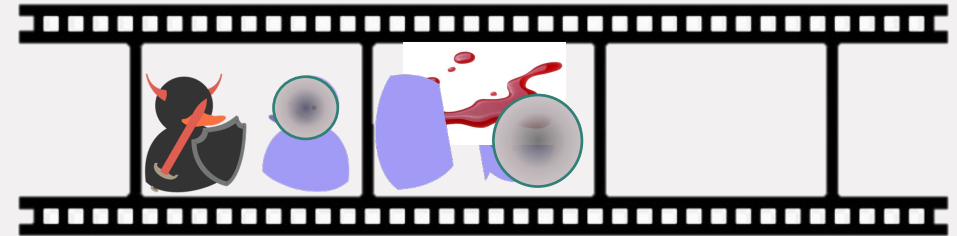
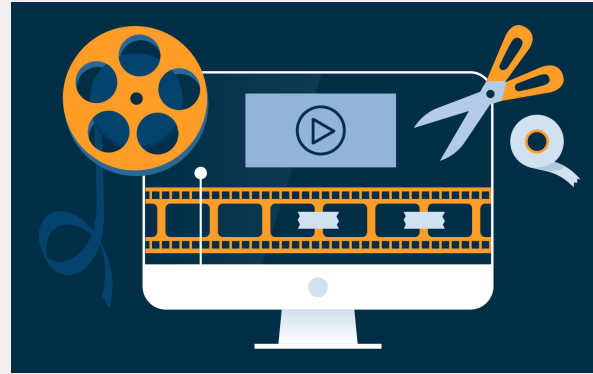
Jonathan Lee
Nanotronics Imaging

Srinath Setty
Microsoft Research

Justin Thaler
Georgetown University

Riad S. Wahby
Algorand and Carnegie Mellon University

An Example Application of Fighting Disinformation



Edit: blurring sensitive part

Only “allowed” set of edits have been made to this video

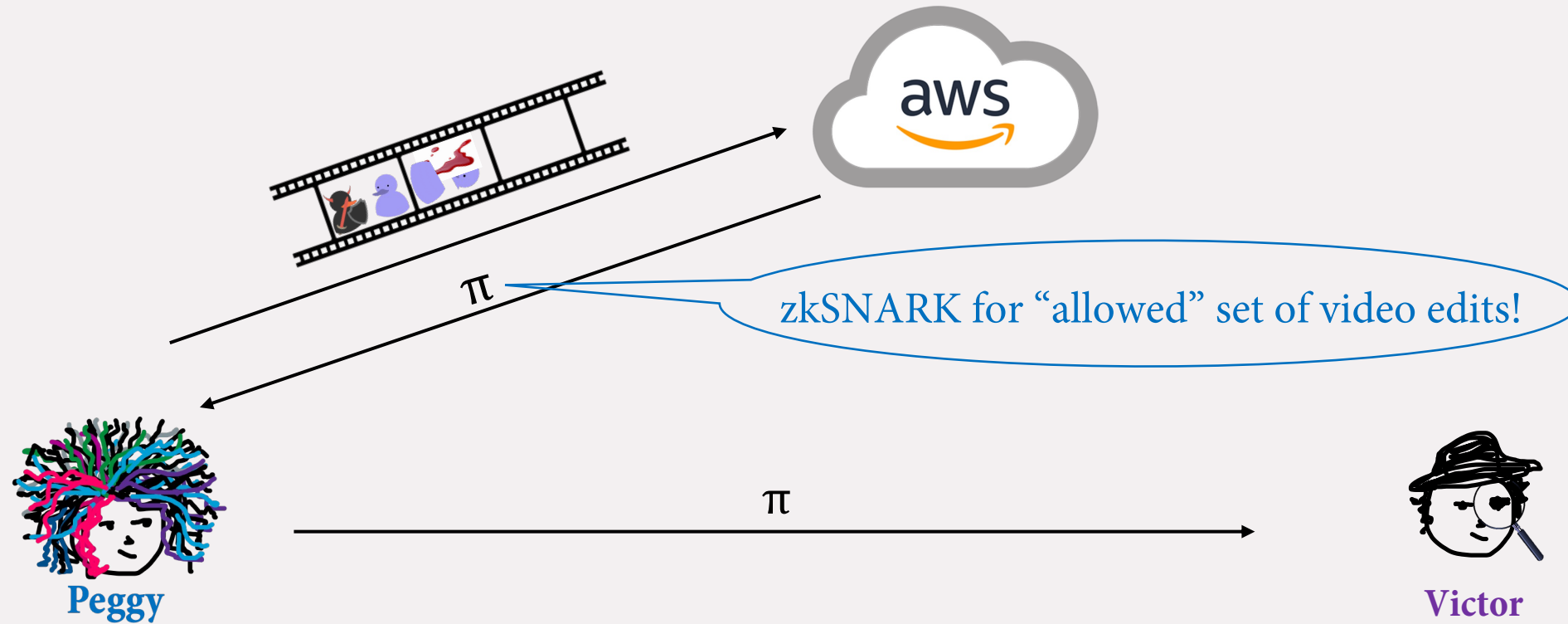


π



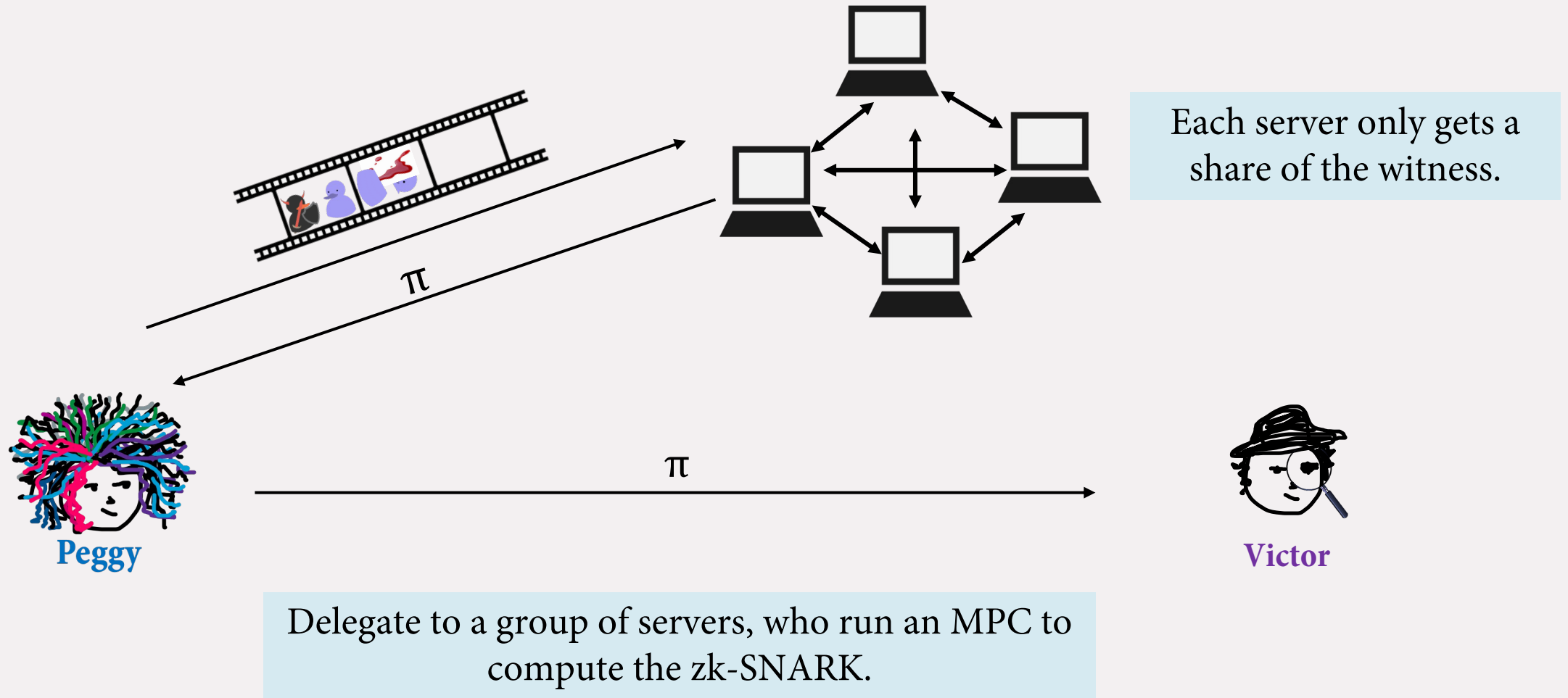
Proof needs to be zero-knowledge.
Large videos $\Rightarrow$ SNARK generation time long.

Can the zkSNARK computation be delegated?

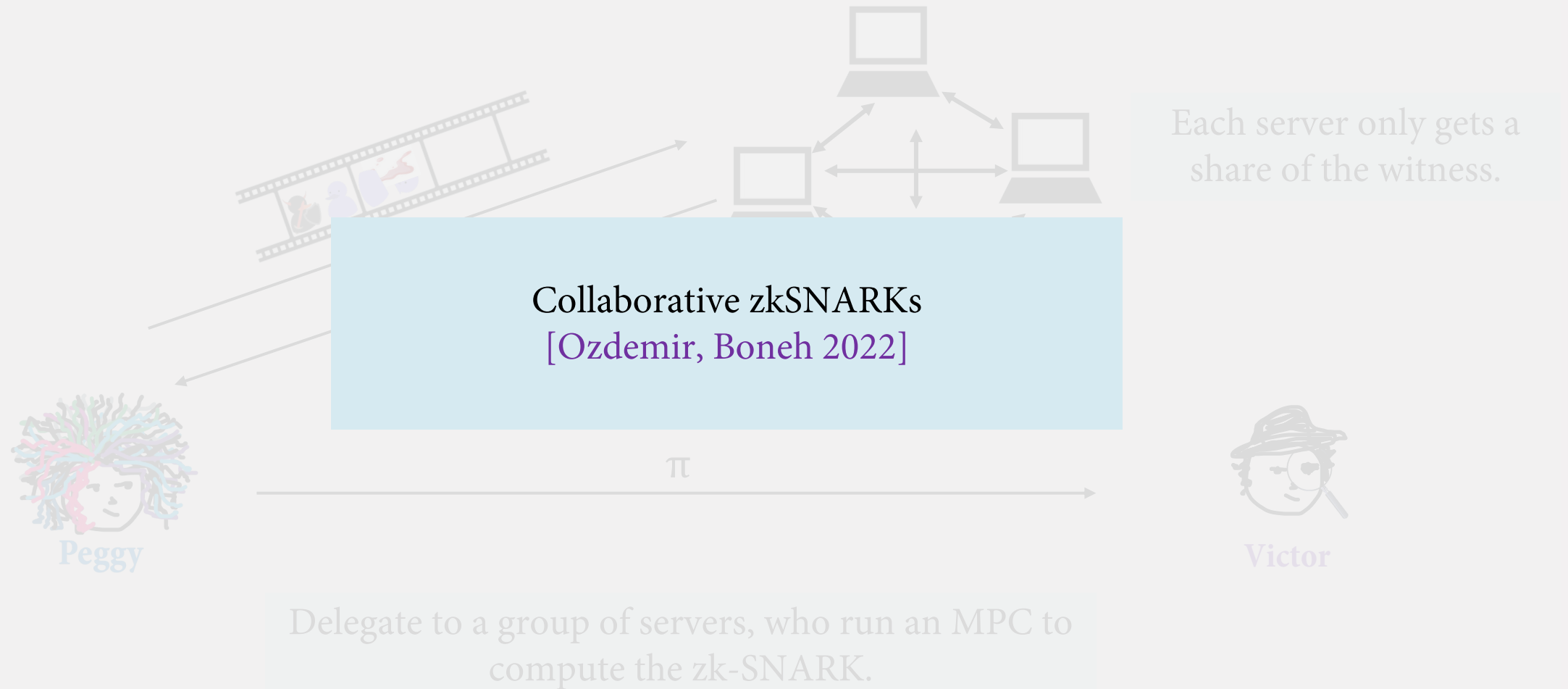


Violates Privacy!!!: Requires leaking the entire sensitive video to AWS

Can the zkSNARK computation be delegated?



Can the zkSNARK computation be delegated?



Prior Works

Collaborative zkSNARKs [OB22]

Efficient MPC to compute the zkSNARK proof –privacy for client's witness.

Each server in this MPC takes as much computation time as a single prover!

Distributed Zero Knowledge Proof System [WZCAS18]

Distribute work across machines in a compute cluster to get faster proof generation.

Leaks witness to the clusters -no privacy!

Our Results

OUR GOAL: Faster proof generation + Privacy-preserving guarantees of Collaborative zk-SNARKs

Framework

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

Design

Design **zkSaaS** to obtain faster runtimes than local provers for widely used zk-SNARKs, such as Groth16 [Gro16], Marlin [CHMMVW20] and Plonk [GWC19].

Implementation

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

Concurrent Work: EOS [CLMZ23]

- Also achieve our goal: private delegation of zkSNARKs for faster proof generation.
- Model differs from ours:

EOS [CLMZ23]

Client always online, throughout the
zkSNARK computation.
MPC involving (honest) client + servers

Reduces client's work in this model
(doesn't divide the work amongst all servers)

zkSaaS

Client shares the extended witness.
After this, entire zkSNARK computation is
delegated to the servers

Delegates all of clients work to servers
(post sharing of the extended witness)

The image features a light gray background with several semi-transparent colored circles of various sizes in the corners. In the top-left, there are circles in teal, orange, light blue, and dark blue. In the top-right, there are circles in purple, orange, and light blue. In the bottom-right, there are circles in pink, teal, dark blue, and a large purple one. In the bottom-left, there are small circles in pink, teal, and dark blue.

zkSaaS Framework

zkSNARKs: Typical Design Blueprint

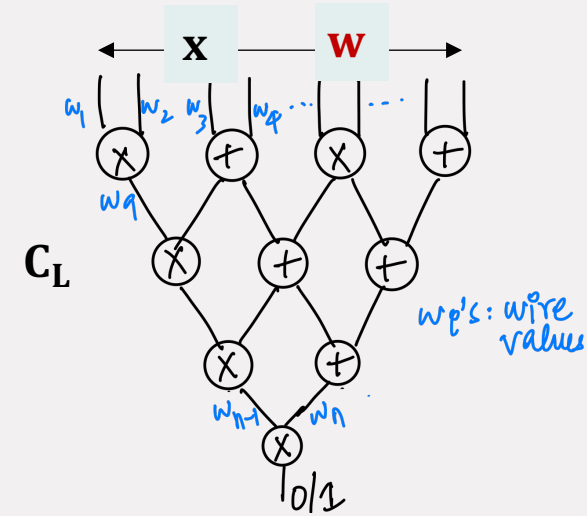
Step I (Extended Witness Generation)

Step II (Proof Generation)

zkSNARKs: Typical Design Blueprint

Step I (Extended Witness Generation)

Extended witness has the wire values corresponding to the evaluation of C_L on statement $\mathbf{x}$ and witness $\mathbf{w}$.

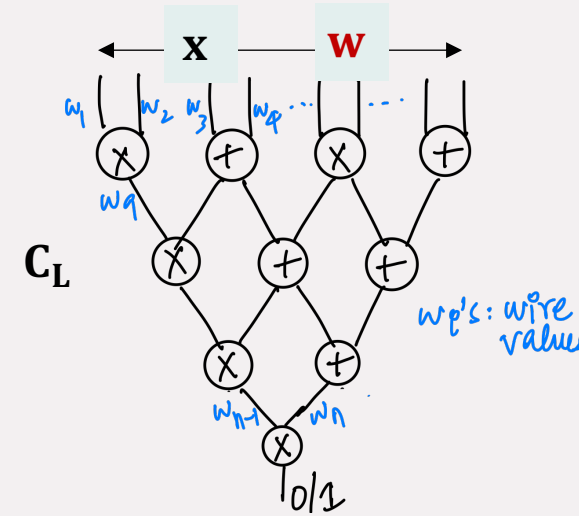


Step II (Proof Generation)

zkSNARKs: Typical Design Blueprint

Step I (Extended Witness Generation)

Extended witness has the wire values corresponding to the evaluation of C_L on statement $\mathbf{x}$ and witness $\mathbf{w}$.



Step II (Proof Generation)

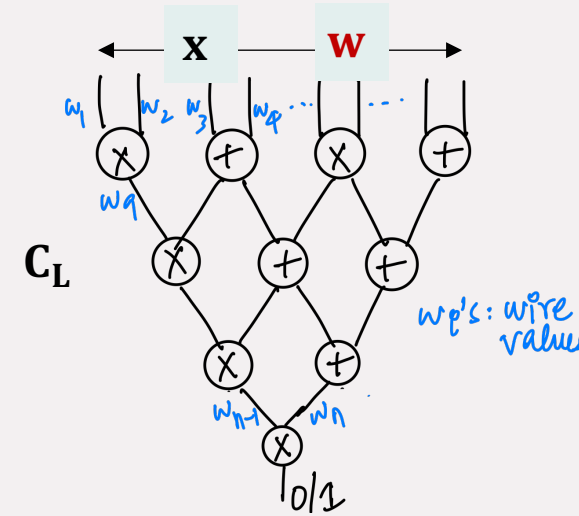
Cryptographic Operations (more expensive)
+ Field operations

zk-SNARK

zkSNARKs: Typical Design Blueprint

Step I (Extended Witness Generation)

Extended witness has the wire values corresponding to the evaluation of C_L on statement $\mathbf{x}$ and witness $\mathbf{w}$.



Step II (Proof Generation)

Cryptographic Operations (more expensive)
+ Field operations

zk-SNARK



Fiat-Shamir (using RO)

An zk-succinct argument of knowledge



information-theoretic
Interactive Oracle Proofs (IOP)

+

cryptographic
Functional Commitment Scheme

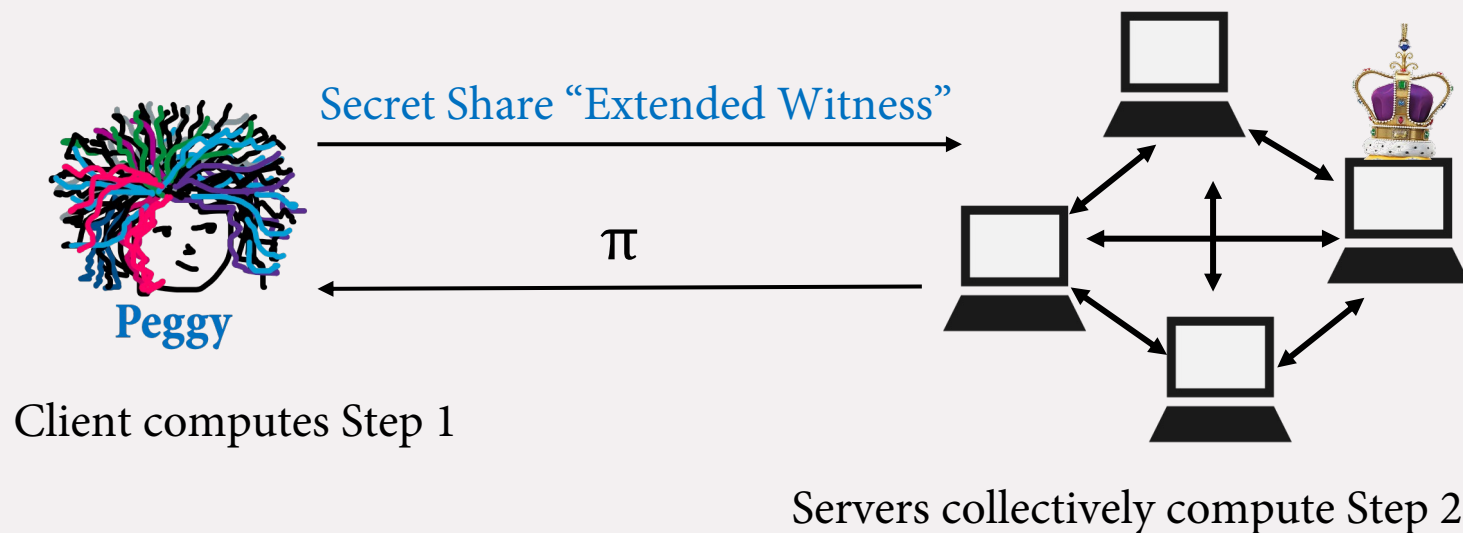
zkSaaS Framework

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



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**.

zkSaaS Framework: Example Applications

To aid users with mobile devices

Private Transactions and Smart Contracts

Prove general computations
on smart contracts

Statements involving Ethereum Wallets

E.g. “I own an Eth wallet out of a
set of 1024 wallets”

For extremely large computations

Fighting Disinformation

Prove that only valid set of edits
were made to image/video

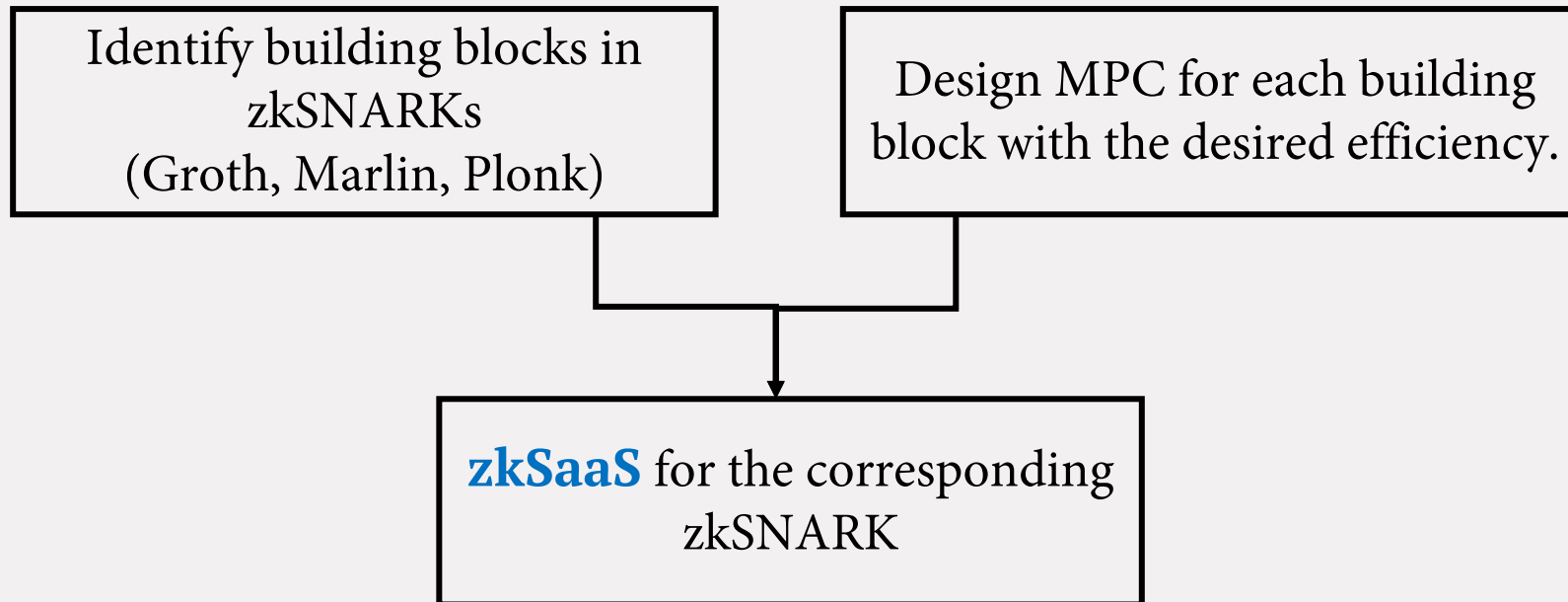
Verifiable Private ML Inference

Proof of inference on a learning model

The slide features a light gray background with decorative elements in the corners. The top-left corner contains a large light teal circle, a small orange circle, a small light blue circle, a medium blue circle, and a tiny dark blue dot. The top-right corner has a large purple circle, a small orange circle, and a medium blue circle. The bottom-right corner is decorated with a small pink circle, a medium teal circle, a small dark blue circle, a large purple circle, and a tiny light blue circle.

Designing zkSaaS

Blueprint of Design [OB22]



Building Blocks in Groth16, Marlin, Plonk

Multi-Scalar Multiplications (MSM)

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

$$g_i \in \mathbb{G}, \alpha_i \in \mathbb{F}, \forall i$$

Partial Products

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

$$x_i \in \mathbb{F}, \forall i$$

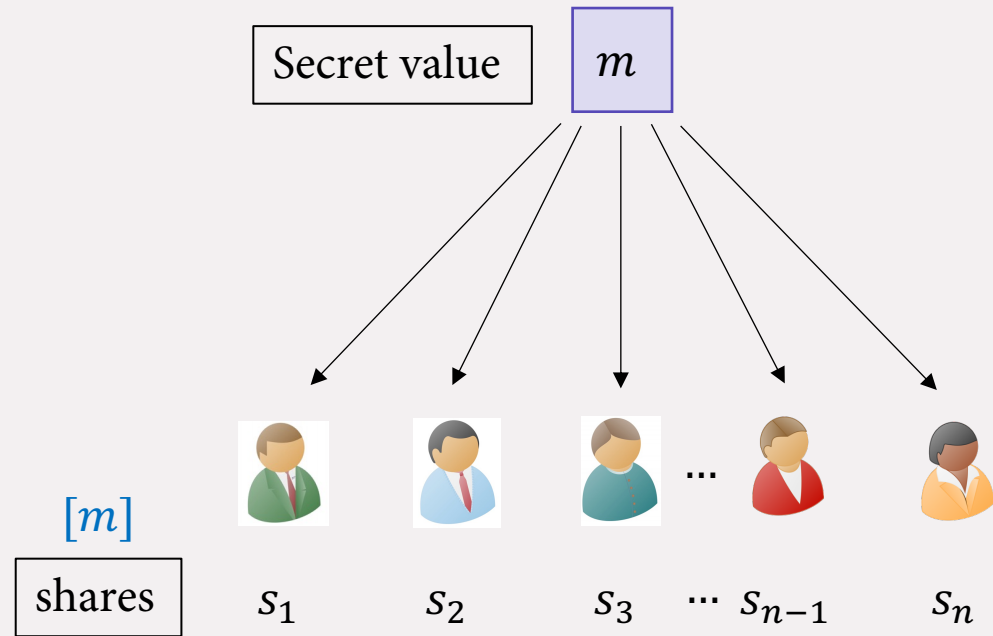
Fast Fourier Transform (FFT)

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]

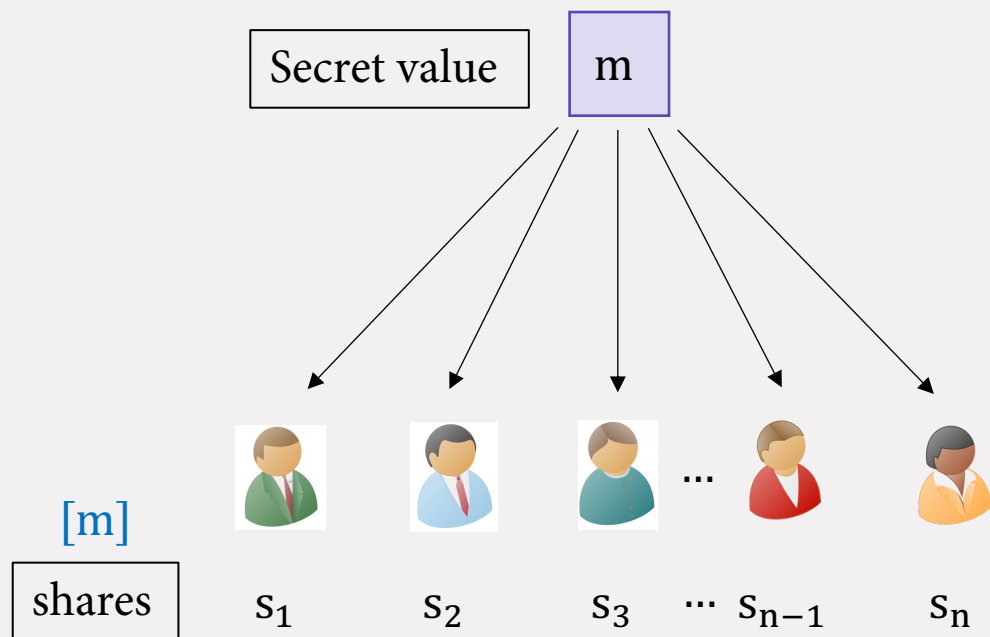


Regular Secret Sharing

1 Value $\rightarrow n$ shares

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

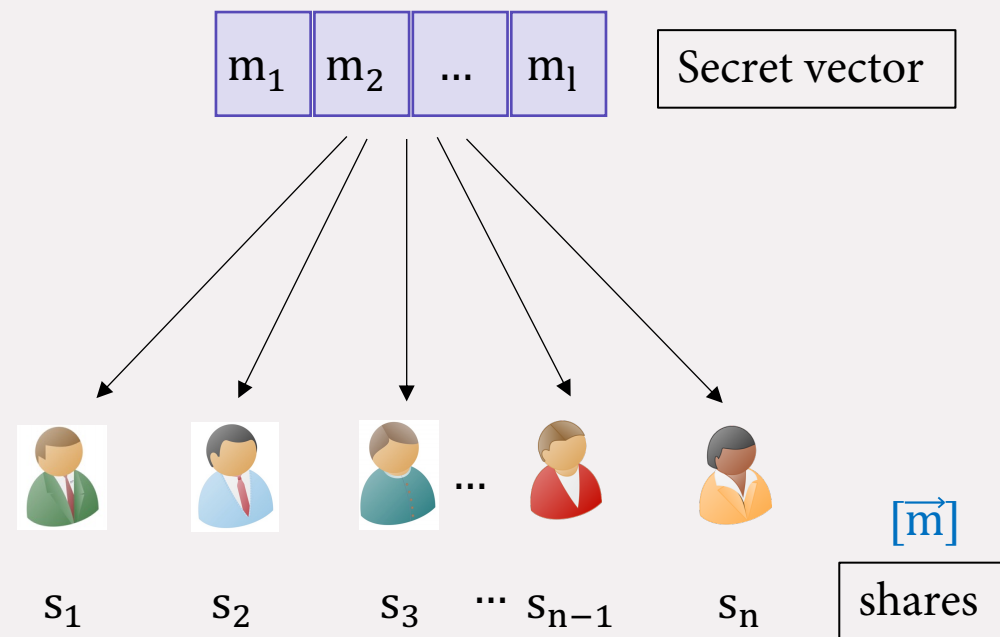
Packed Secret Sharing (PSS) [FY92]



Regular Secret Sharing

1 Value $\rightarrow n$ shares

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

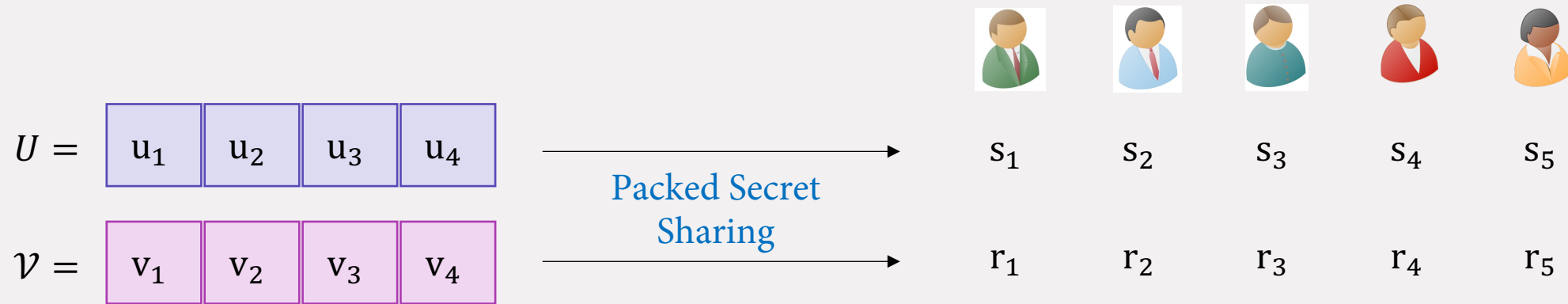


Packed Secret Sharing

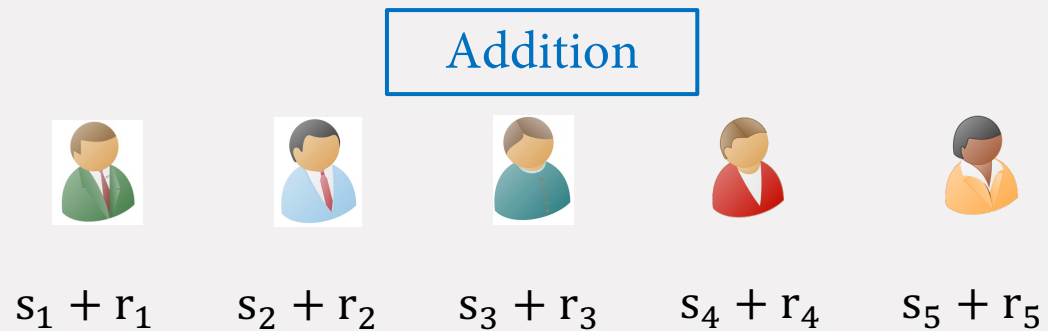
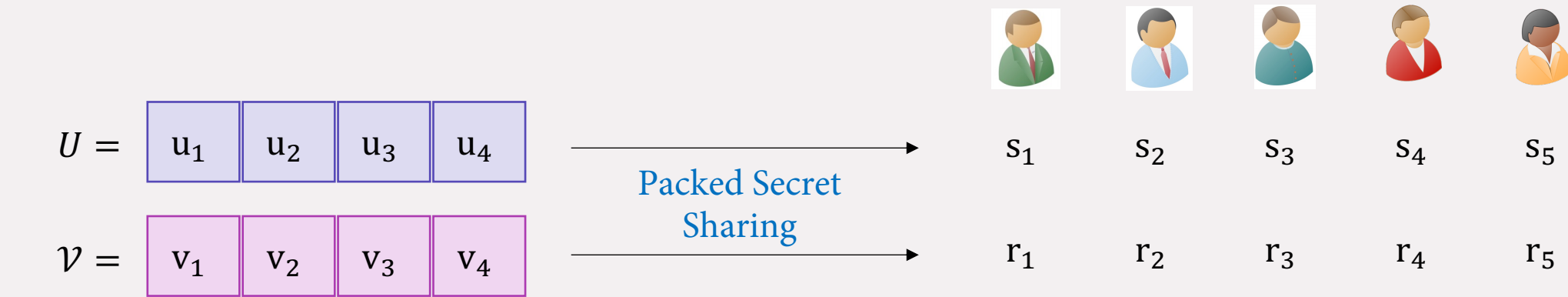
$\ell = O(n)$ Values $\rightarrow n$ shares

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

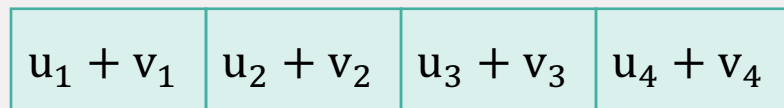
Computing using Packed Secret Sharing



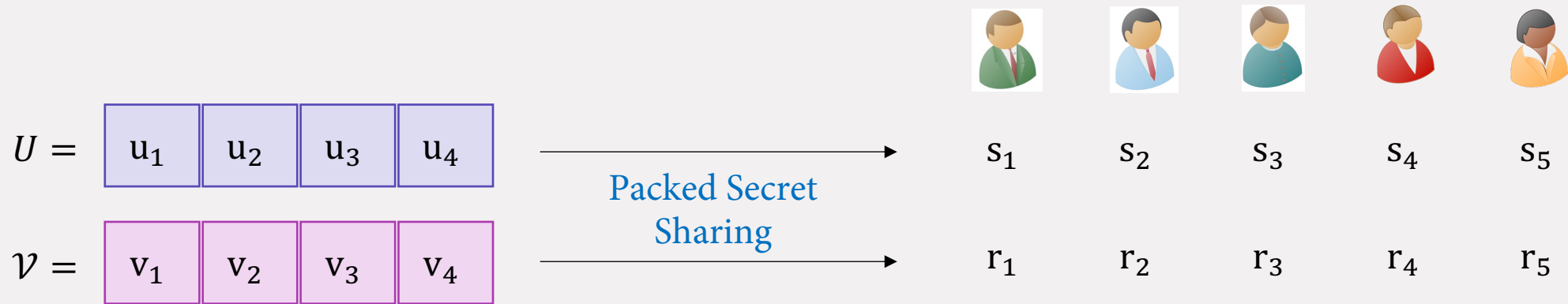
Computing using Packed Secret Sharing



t-out-of-n Packed sharing of:

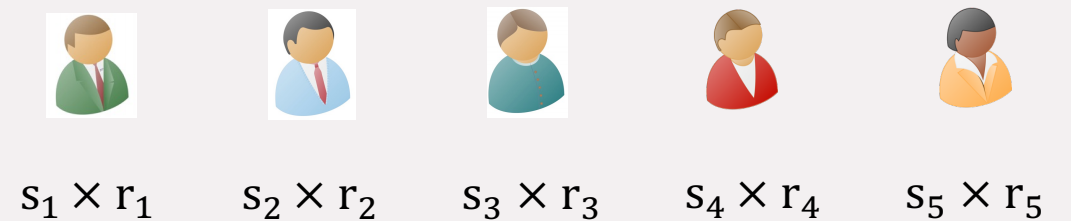
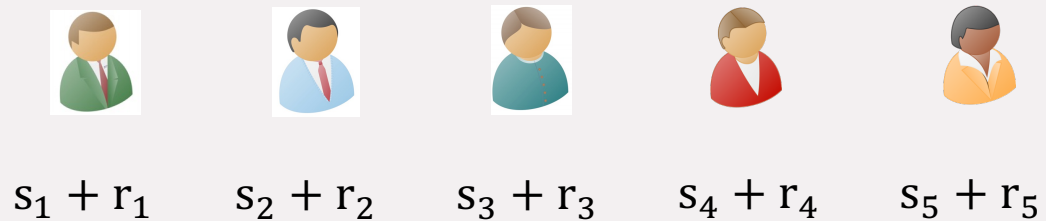


Computing using Packed Secret Sharing

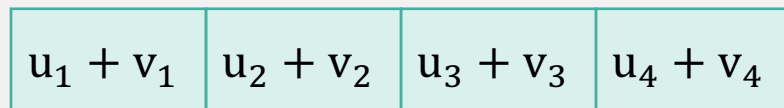


Addition

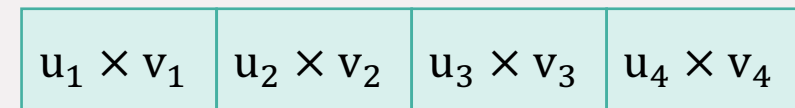
Multiplication



t-out-of-n Packed sharing of:



2t-out-of-n Packed sharing of:



Distributed Protocol for Multi-Scalar Multiplications (MSMs)

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

Distributed Protocol for Multi-Scalar Multiplications (MSMs)

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

Key Observation in [OB22]: For $g \in G, x \in \mathbb{Z}_p$, to compute shares of g^x :

- Compute shares of x : $[x]$
- Compute $g^{[x]}$

Distributed Protocol for Multi-Scalar Multiplications (MSMs)

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

Key Observation in [OB22]: For $g \in G, x \in \mathbb{Z}_p$, to compute shares of g^x :

- Compute shares of x : $[x]$
- Compute $g^{[x]}$

For packed secret shares:

For $g \in G, a, b, c, d \in \mathbb{Z}_p$, to compute shares of

g^a	g^b	g^c	g^d
-------	-------	-------	-------

 :

- Compute shares of $\vec{x} =$

a	b	c	d
-----	-----	-----	-----

 : $[\vec{x}]$
- Compute $g^{[\vec{x}]}$

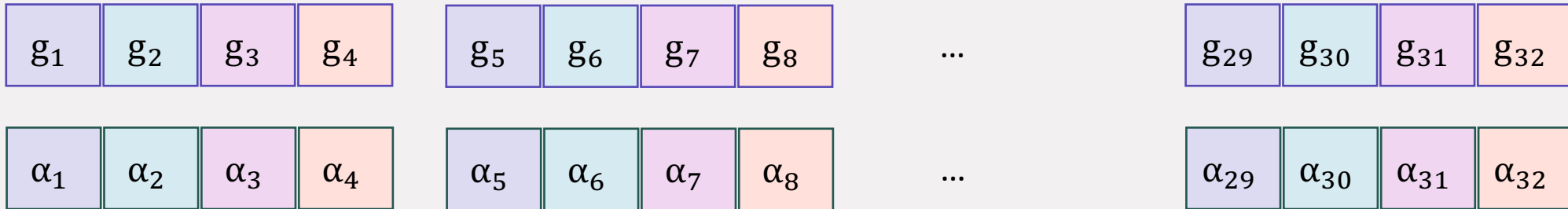
Distributed Protocol for Multi-Scalar Multiplications (MSMs)

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

Distributed Protocol for Multi-Scalar Multiplications (MSMs)

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

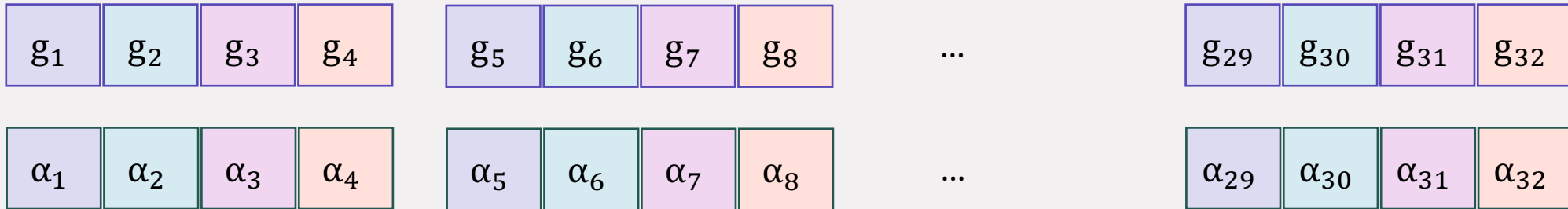
Assume that the parties have packed secret shares of: (take example $m = 32$ and pack vectors with 4 values)



Distributed Protocol for Multi-Scalar Multiplications (MSMs)

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

Assume that the parties have packed secret shares of: (take example $m = 32$ and pack vectors with 4 values)



1. Parties locally compute the MSM functionality (local since $[g_i]$ is actually $g^{[x_i]}$), giving packed

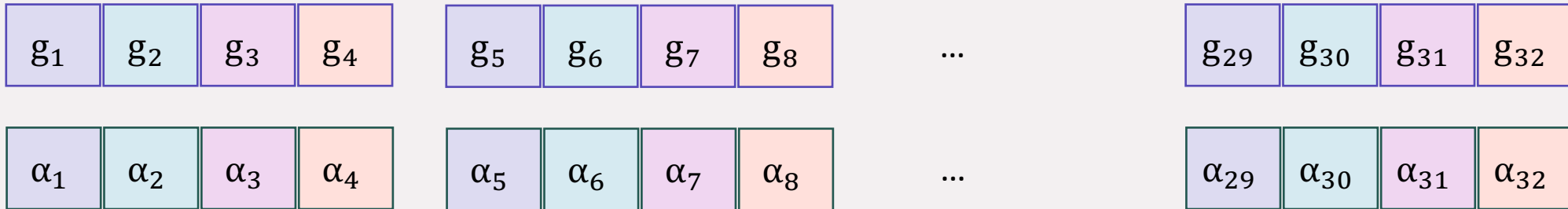
shares of a single vector $\vec{v} =$

$g_1^{\alpha_1} \cdot g_5^{\alpha_5} \dots g_{29}^{\alpha_{29}}$	$g_2^{\alpha_2} \cdot g_6^{\alpha_6} \dots g_{30}^{\alpha_{30}}$	$g_3^{\alpha_3} \cdot g_7^{\alpha_7} \dots g_{31}^{\alpha_{31}}$	$g_4^{\alpha_4} \cdot g_8^{\alpha_8} \dots g_{32}^{\alpha_{32}}$
--	--	--	--

Distributed Protocol for Multi-Scalar Multiplications (MSMs)

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

Assume that the parties have packed secret shares of: (take example $m = 32$ and pack vectors with 4 values)



1. Parties locally compute the MSM functionality (**local since $[g_i]$ is actually $g^{[x_i]}$**), giving packed

shares of a single vector $\vec{v} =$ $g_1^{\alpha_1} \cdot g_5^{\alpha_5} \dots g_{29}^{\alpha_{29}}$ $g_2^{\alpha_2} \cdot g_6^{\alpha_6} \dots g_{30}^{\alpha_{30}}$ $g_3^{\alpha_3} \cdot g_7^{\alpha_7} \dots g_{31}^{\alpha_{31}}$ $g_4^{\alpha_4} \cdot g_8^{\alpha_8} \dots g_{32}^{\alpha_{32}}$

2. Unpack these shares of $\vec{v}$ and get regular shares of $[v_1], [v_2], [v_3], [v_4]$.
3. Locally multiply these shares to get shares of the final output $v_1 \cdot v_2 \cdot v_3 \cdot v_4$.

The slide features a light gray background with decorative elements in the corners. The top-left corner contains a cluster of small circles in teal, orange, light blue, and dark blue. The top-right corner features a large purple circle, a small orange circle, and a medium blue circle. The bottom-right corner is decorated with a large purple circle, a medium teal circle, a small pink circle, and a small dark blue circle.

zkSaaS Implementation

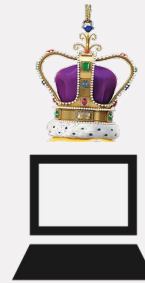
zkSaaS for Groth16 and Plonk: Setup

Local Prover



1vCPU
4 GB RAM

zkSaaS Servers



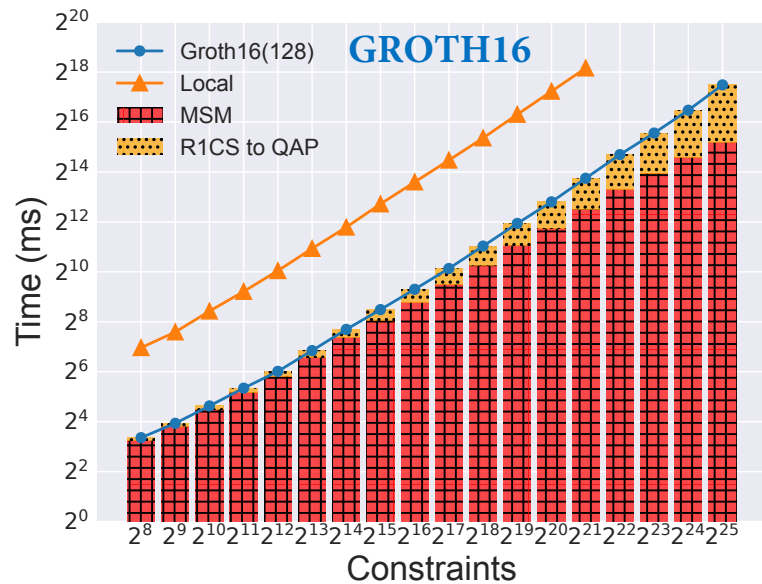
96vCPU
128 GB RAM



1vCPU, 2 GB RAM each

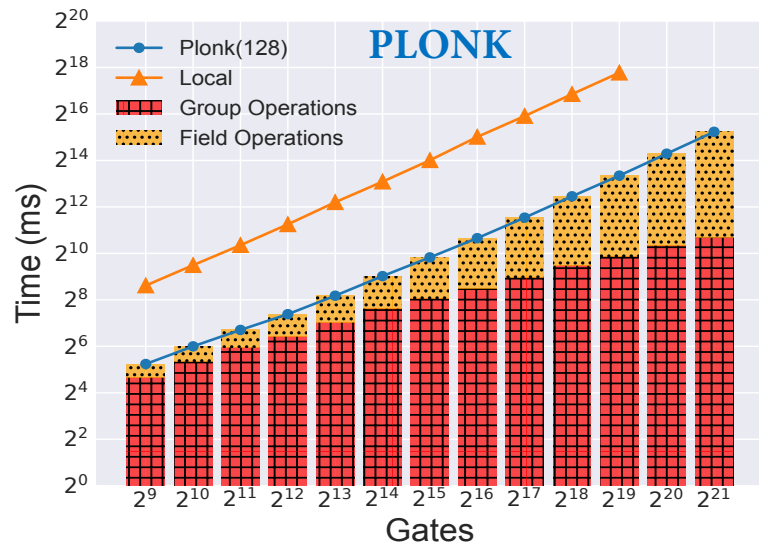
N1 GCP Instances

zkSaaS for Groth16 and Plonk: Evaluation



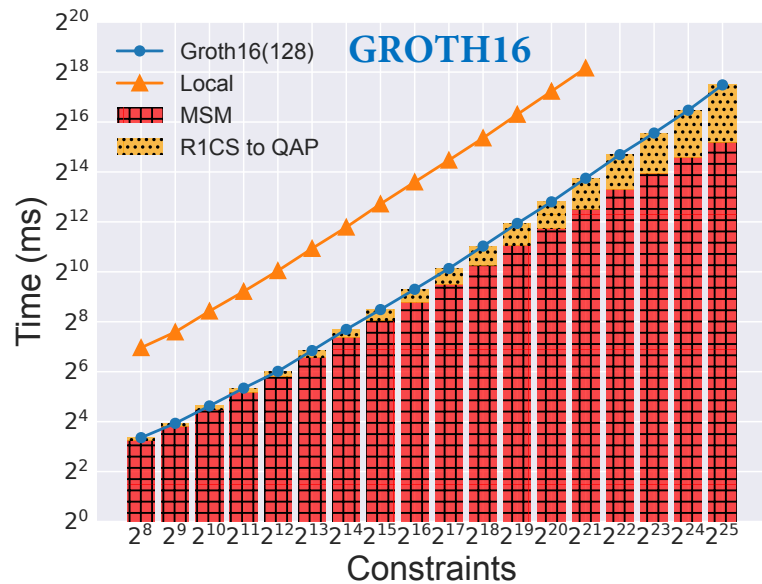
No. of Servers = 128
Packing constant = 32
No. of corrupt servers = 31

Memory Exhaustion



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

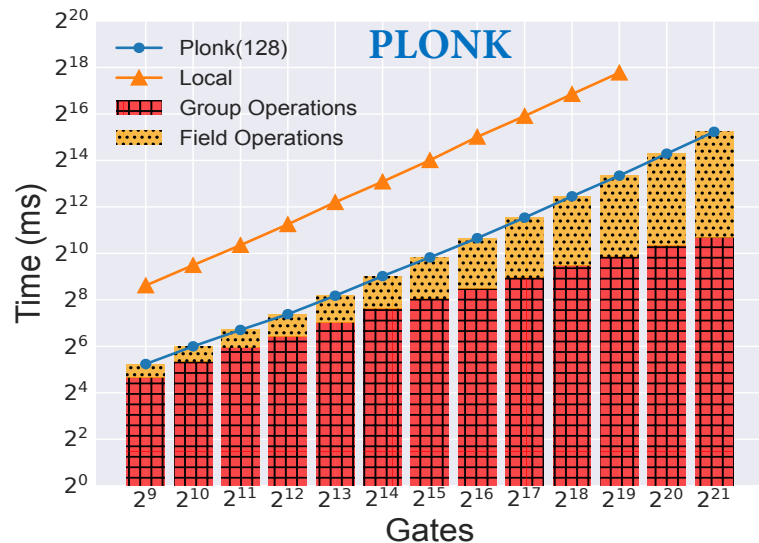
zkSaaS for Groth16 and Plonk: Evaluation



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

Running Time

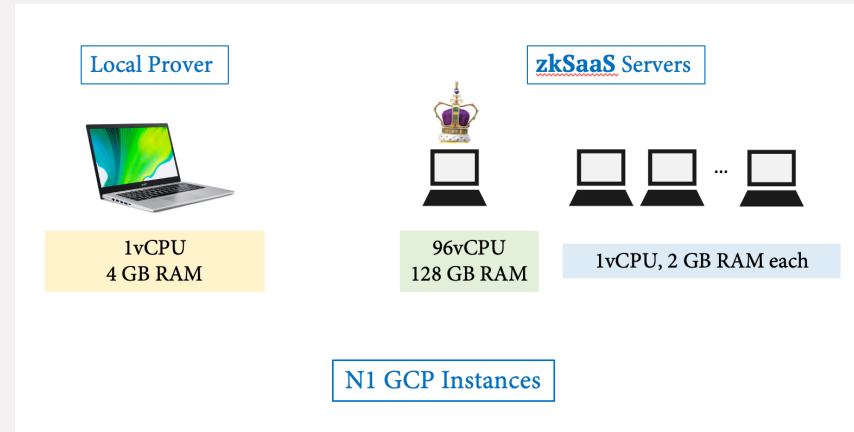
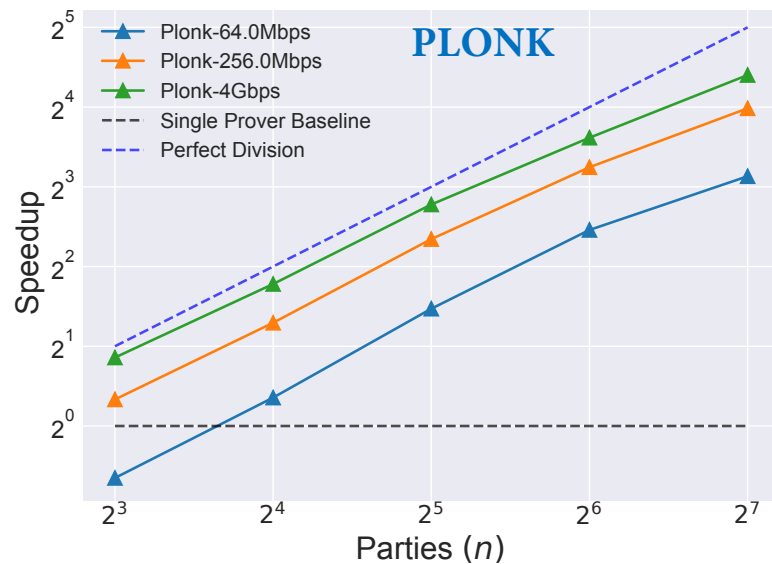
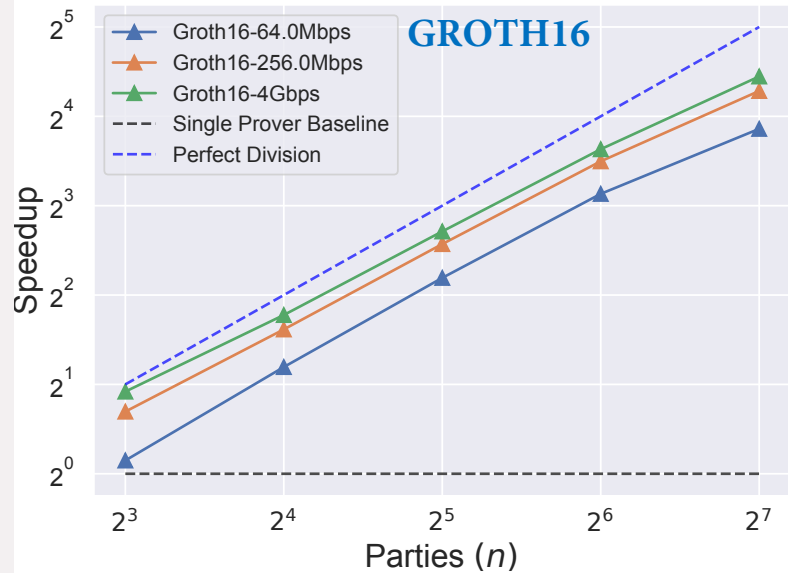
We get $\approx 22\times$ speed-up over consumer machine



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 and Plonk: Evaluation



No. of constraints = 2^{19}
Packing constant = $n/4$
No. of corrupt servers = $n/4 - 1$



THANK YOU

eprint/2023/905

<https://github.com/guruvamsi-policharla/zksaas>

Distributed Protocol for Fast Fourier Transform (FFT)

FFT: Given polynomial $p(x) = a_0 + a_1x + \dots + a_{31}x^{31}$ and $W = \{\omega^0, \omega^1, \dots, \omega^{31}\}$ (roots of unity)

Evaluate $p(x)$ on W .

$$p(x) = a_0 + a_1x + \dots + a_{31}x^{31}$$

$$p_{\text{odd}}(x) = a_1 + a_3x + \dots + a_{31}x^{15}$$

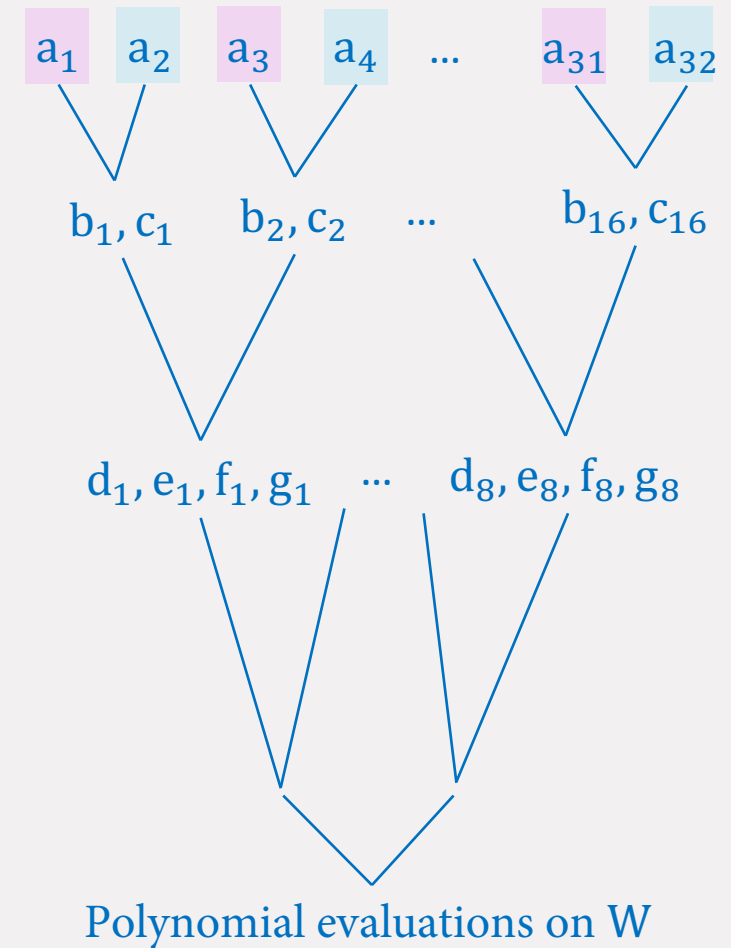
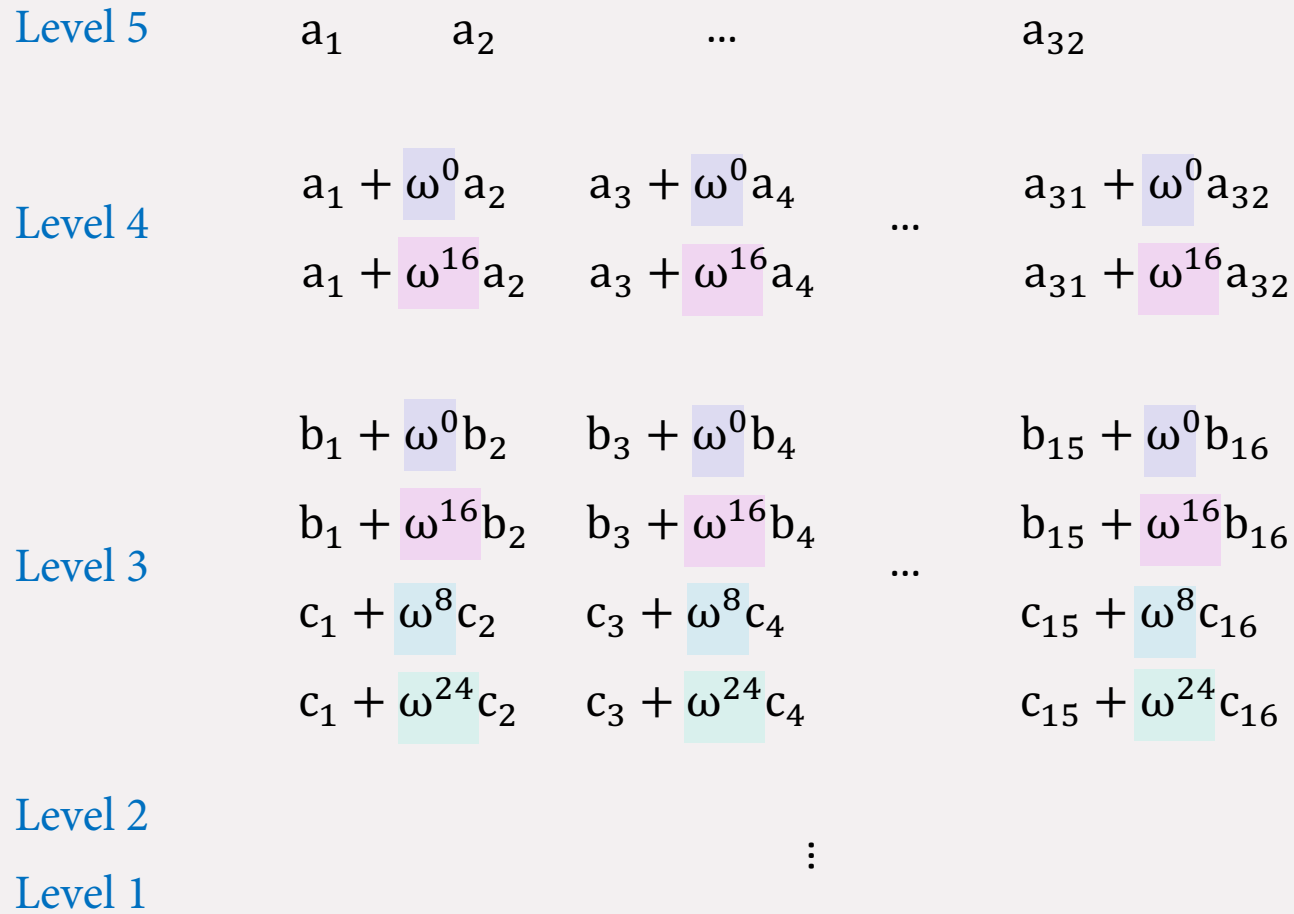
$$p_{\text{even}}(x) = a_0 + a_2x + \dots + a_{30}x^{15}$$

$$p(x) = p_{\text{even}}(x^2) + x \cdot p_{\text{odd}}(x^2)$$

Need to evaluate on $W^2 \longrightarrow 16$ unique values

Recurse for $\log 32 = 5$ steps

Distributed Protocol for Fast Fourier Transform (FFT)

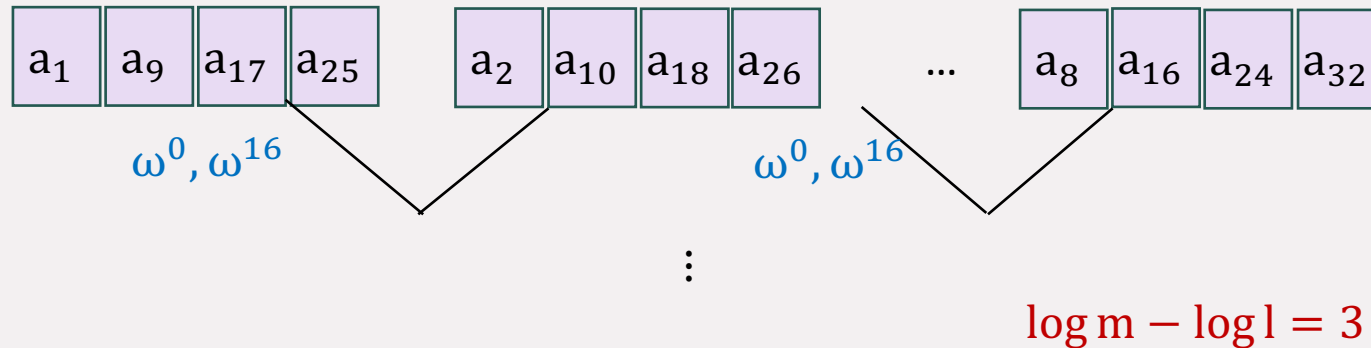


Distributed Protocol for Fast Fourier Transform (FFT)

How do we pack the starting values so that we can compute down the tree on vectors of values?

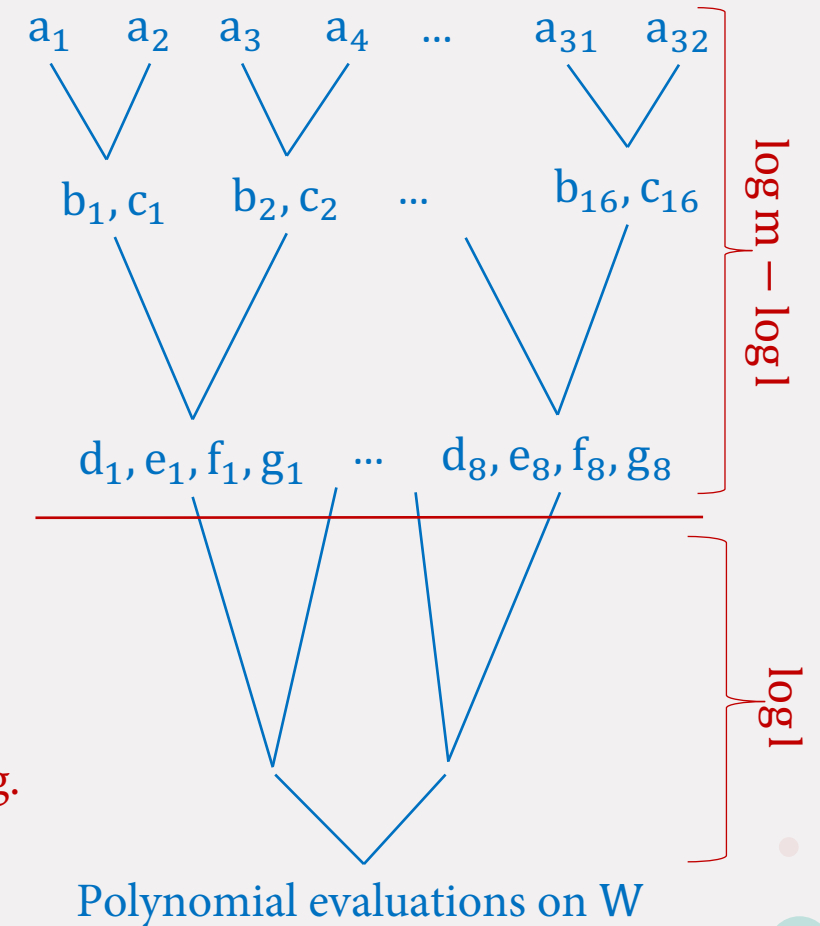
Key Observation: Pack the $a_1, a_2, \dots, a_{32}$ as far away as possible so that we can continue to compute on the PSS.

Packing $l = 4$ values together ($m = 32$)



After 3 steps, the computation needs to be done on values within same packing.

Last $\log l = 2$ levels are computed locally by parties!



Distributed Protocol for Fast Fourier Transform (FFT)

Key Observation: FFT (and each level i) is a linear operation.

$$F_{\text{FFT}}^i((a_1 + r_1), \dots, (a_m + r_m)) = F_{\text{FFT}}^i(a_1, \dots, a_m) + F_{\text{FFT}}^i(r_1, \dots, r_m)$$

For the last $\log l = 2$ levels of computation:

Pre-processing: Assuming packed shares of

$$(s_1, \dots, s_{32}) = F_{\text{FFT}}^1(F_{\text{FFT}}^2(r_1, \dots, r_{32}))$$

The parties do the following:

- Mask the shares at the end of Level 3 using shares of r_i 's.
- King party reconstructs these masked shares, locally computes Levels 2,1 on these masked shares.
- Final value obtained by unmasking using the shares of s_i 's.

