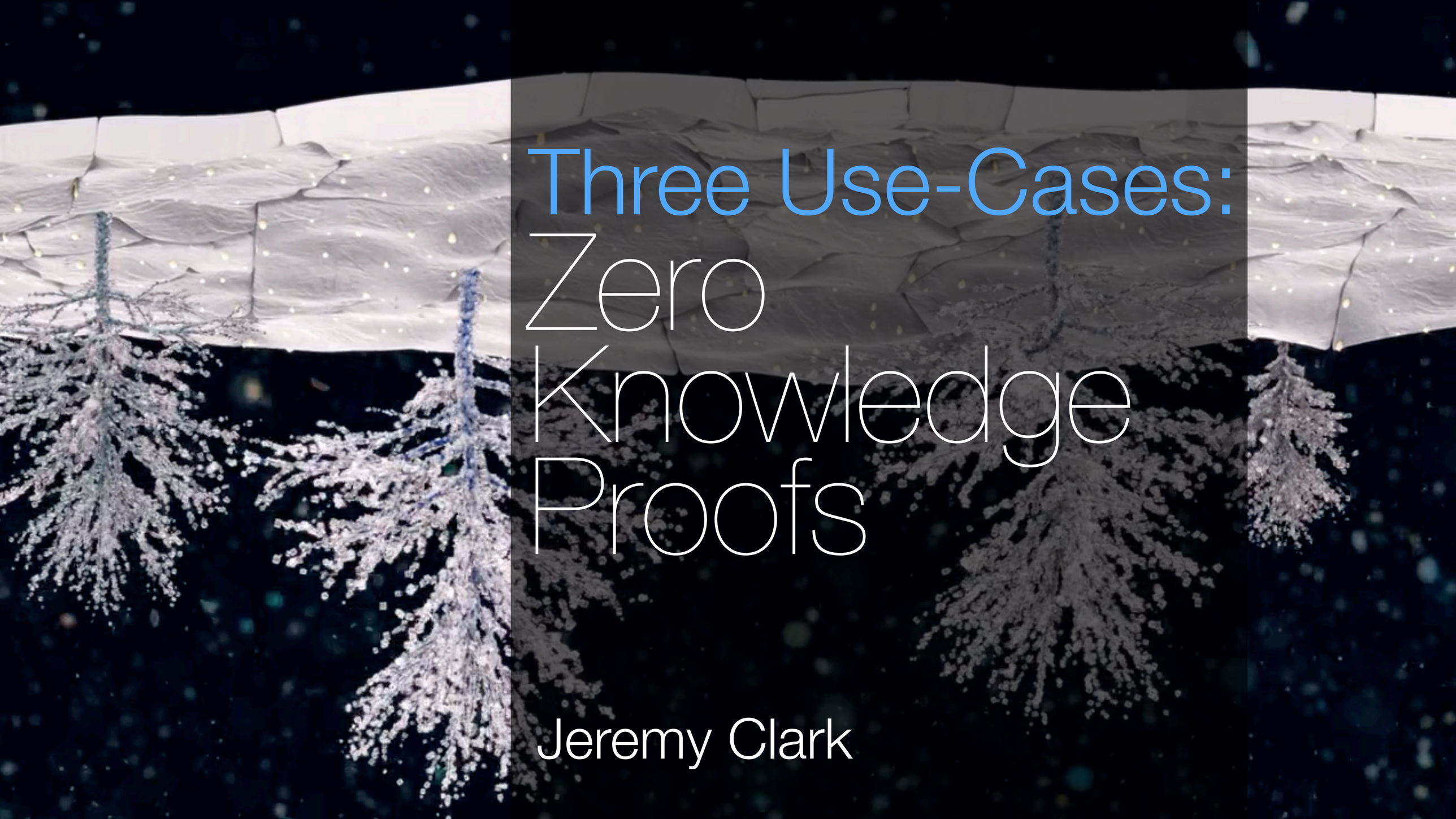


Three Use-Cases: Zero Knowledge Proofs

Jeremy Clark

MADIBA security research group

UNIVERSITÉ Concordia UNIVERSITY

GINA CODY SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

FUNDING & PARTNERS:

NSERC CRSNG

Raymond Chabot Grant Thornton

catallaxy

AUTORITÉ DES MARCHÉS FINANCIERS

Fonds de recherche Nature et technologies Québec

Office of the Privacy Commissioner of Canada

Computation

(or Algorithm / Process / Circuit / Function)

My balance in BTC

My balance in BTC

Bitcoin’s blockchain

My addresses

Total: 1 BTC

My balance in BTC

Bitcoin’s blockchain

My addresses

Total: 1 BTC

Inputs:
Disclose (for now)

Computation:
Disclose

Output:
Disclose

My balance in BTC

Bitcoin’s blockchain

My addresses

Total: 1 BTC

Inputs:
Disclose (for now)

Computation:
Disclose

Output:
Disclose

Check:
(1) Rerun the computation

My balance in BTC

Bitcoin’s blockchain

My addresses

Total: 1 BTC

Inputs:
Disclose (for now)

Computation:
Disclose

Output:
Disclose

Check:
(1) Rerun the computation
(2) I give you a “proof” and you check the proof

My balance in BTC

Bitcoin’s blockchain

My addresses

Total: 1 BTC

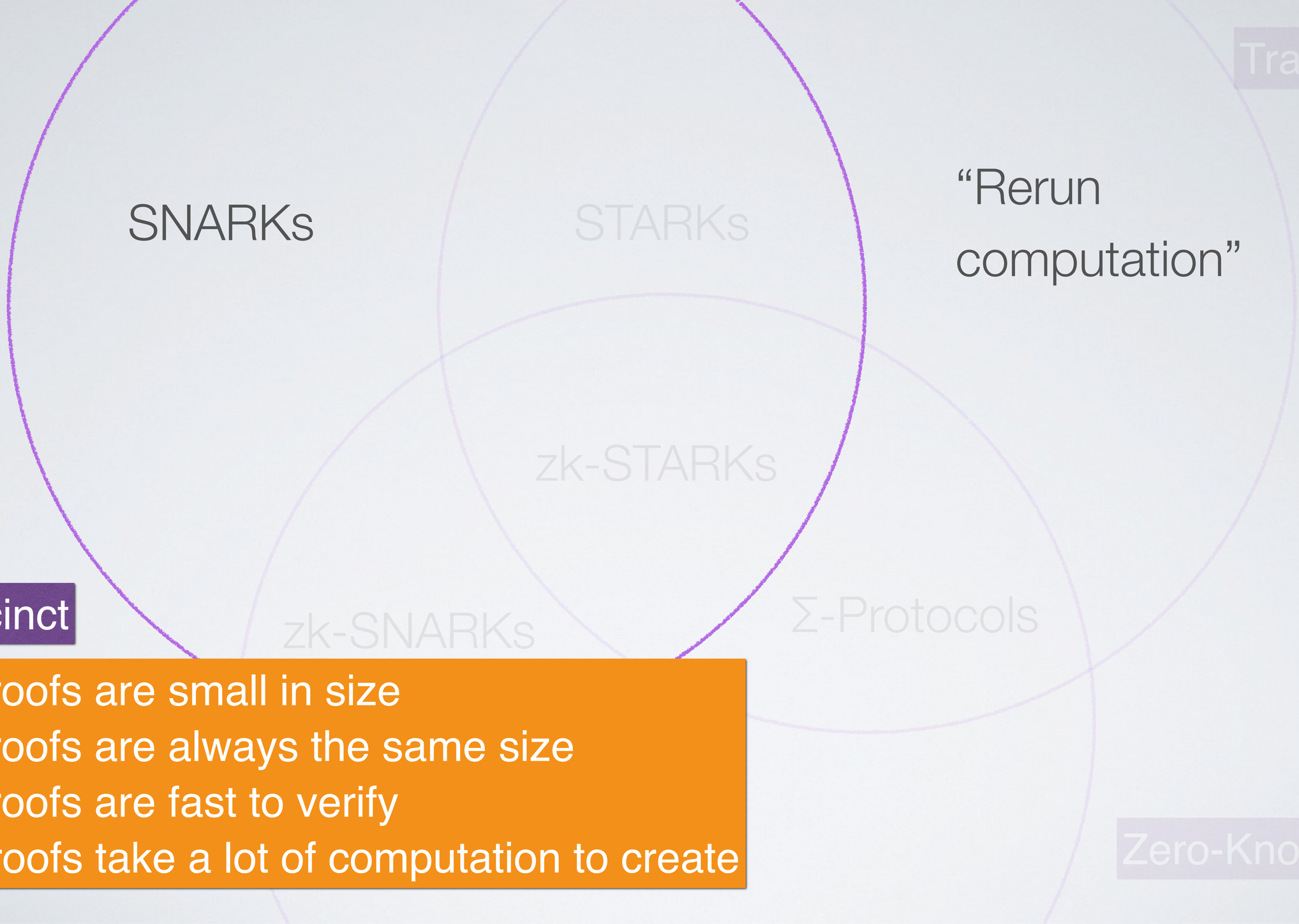
Inputs:
Disclose (for now)

Computation:
Disclose

Output:
Disclose

Check:
(1) Rerun the computation
(2) I give you a “proof” and you check the proof

Why (2)?
Cost or many checkers



Succinct

- ⊕Proofs are small in size
- ⊕Proofs are always the same size
- ⊕Proofs are fast to verify
- ⊖Proofs take a lot of computation to create

My balance in BTC

Bitcoin’s blockchain

My addresses

Total: 1 BTC

My balance in BTC

Bitcoin’s blockchain

~~My addresses~~

My private keys

Total: 1 BTC

Input:
Disclose

My balance in BTC

Bitcoin’s blockchain

~~My addresses~~

My private keys 🔒

Total: 1 BTC

Input:
Hidden

Computation:
Disclose

Output:
Disclose

Input:
Disclose

My balance in BTC

Bitcoin’s blockchain

~~My addresses~~

My private keys 🔒

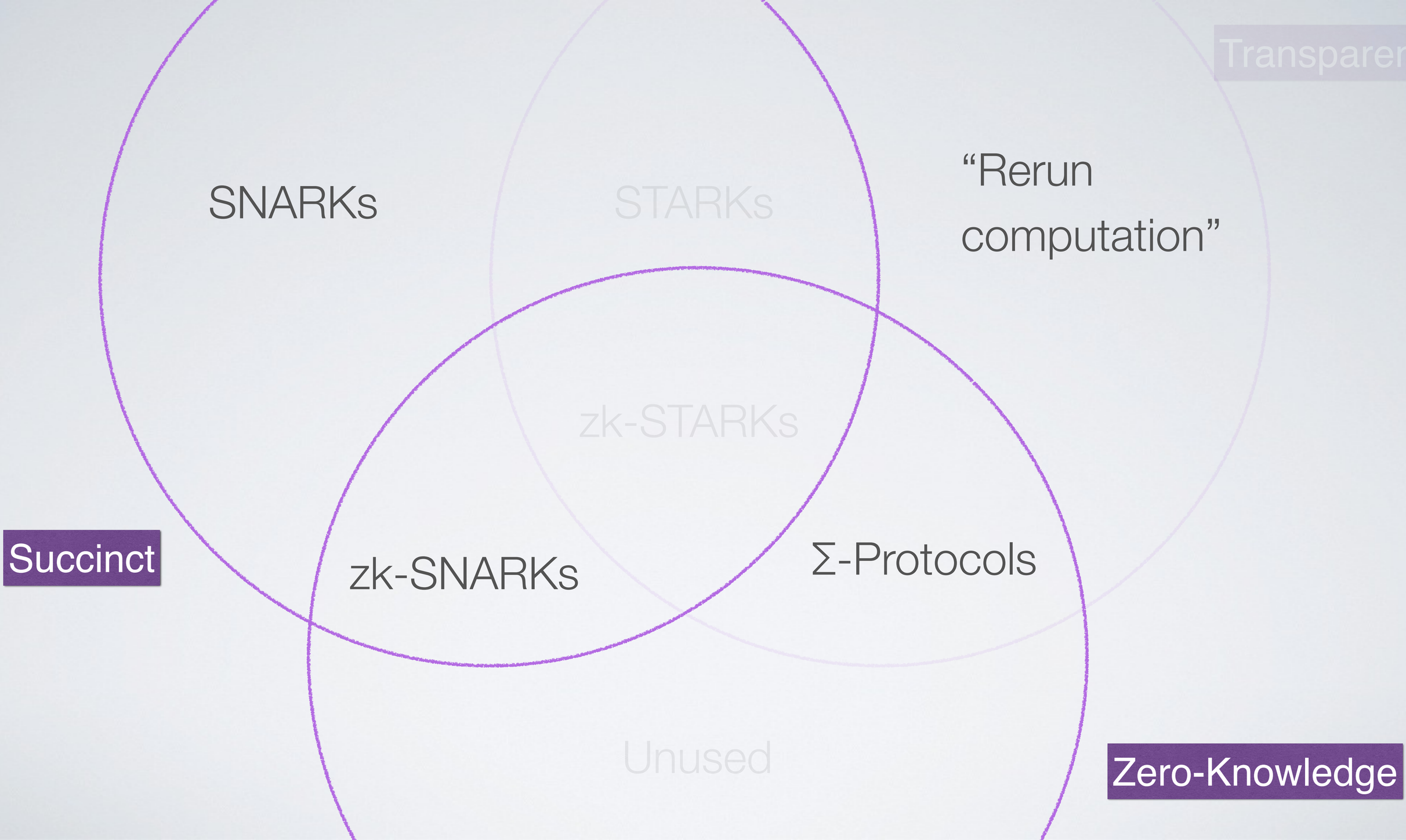
Total: 1 BTC

Input:
Hidden

Computation:
Disclose

Output:
Disclose

Check:
(1) ~~Rerun the computation~~
(2) I give you a “proof” and you check the proof


Transparent
SNARKs
STARKs
“Rerun computation”
zk-STARKs
Succinct
zk-SNARKs
Σ-Protocols
Unused
Zero-Knowledge

Some SNARKs require a trusted setup

- Generate a secret number
- Generate public values using secret
- Delete the secret
- If secret leaks, proofs can be faked

Some SNARKs require a trusted setup

- Generate a secret number
- Generate public values using secret
- Delete the secret
- If secret leaks, proofs can be faked

- Multiple people can collaborate to generate
- Secure if only 1 deletes their secret
- How many setups?

SNARKs require a trusted setup

BTC balance

ETH balance

XRP balance

SNARKs require a trusted setup

BTC balance

10 people do this

ETH balance

30 people do this

XRP balance

3 people do this

SNARKs require a trusted setup

BTC balance

10 people do this

ETH balance

30 people do this

XRP balance

3 people do this

43 Setups	3 Setups	1 Setup	0 Setups

SNARKs require a trusted setup

BTC balance

10 people do this

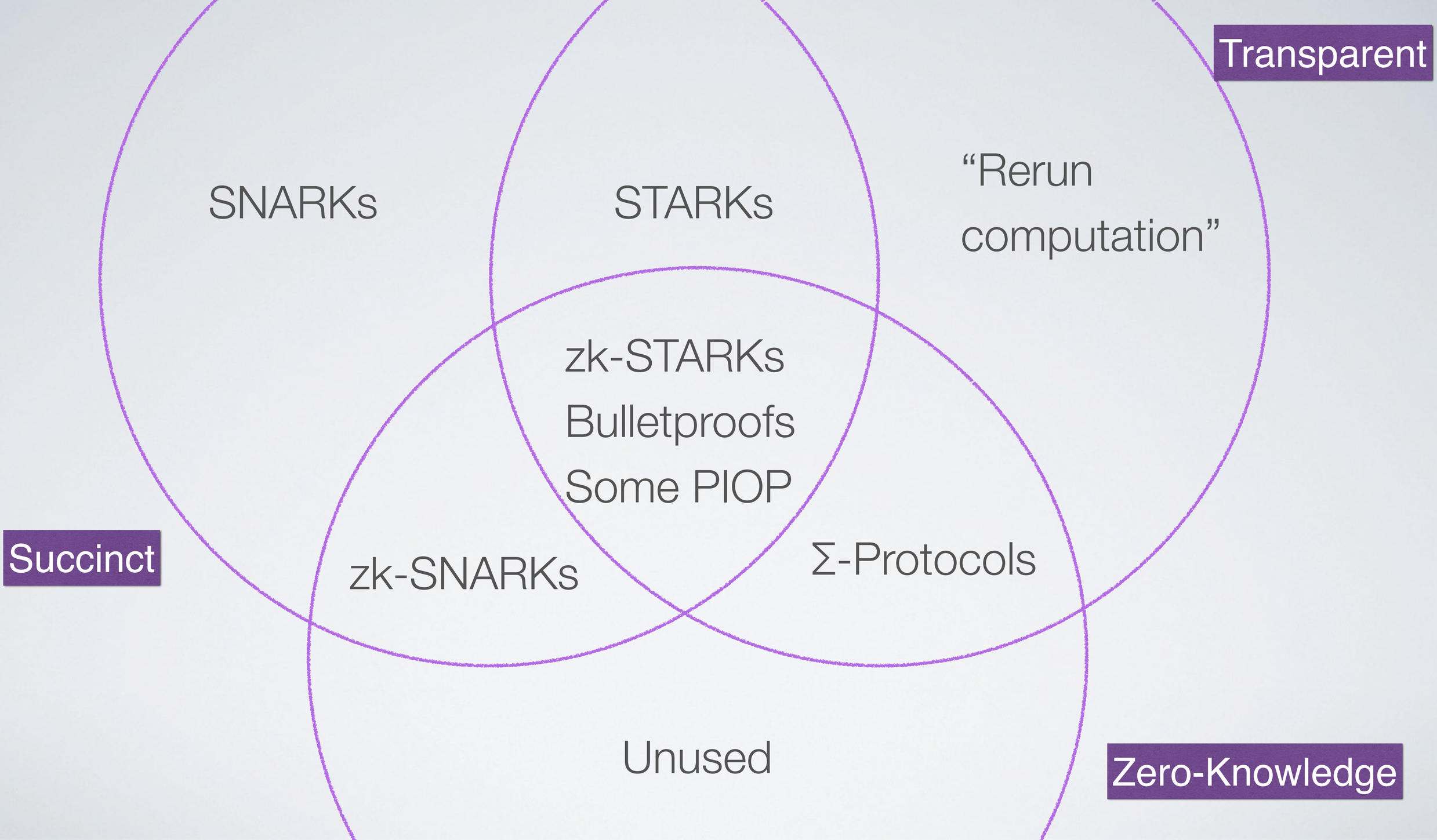
ETH balance

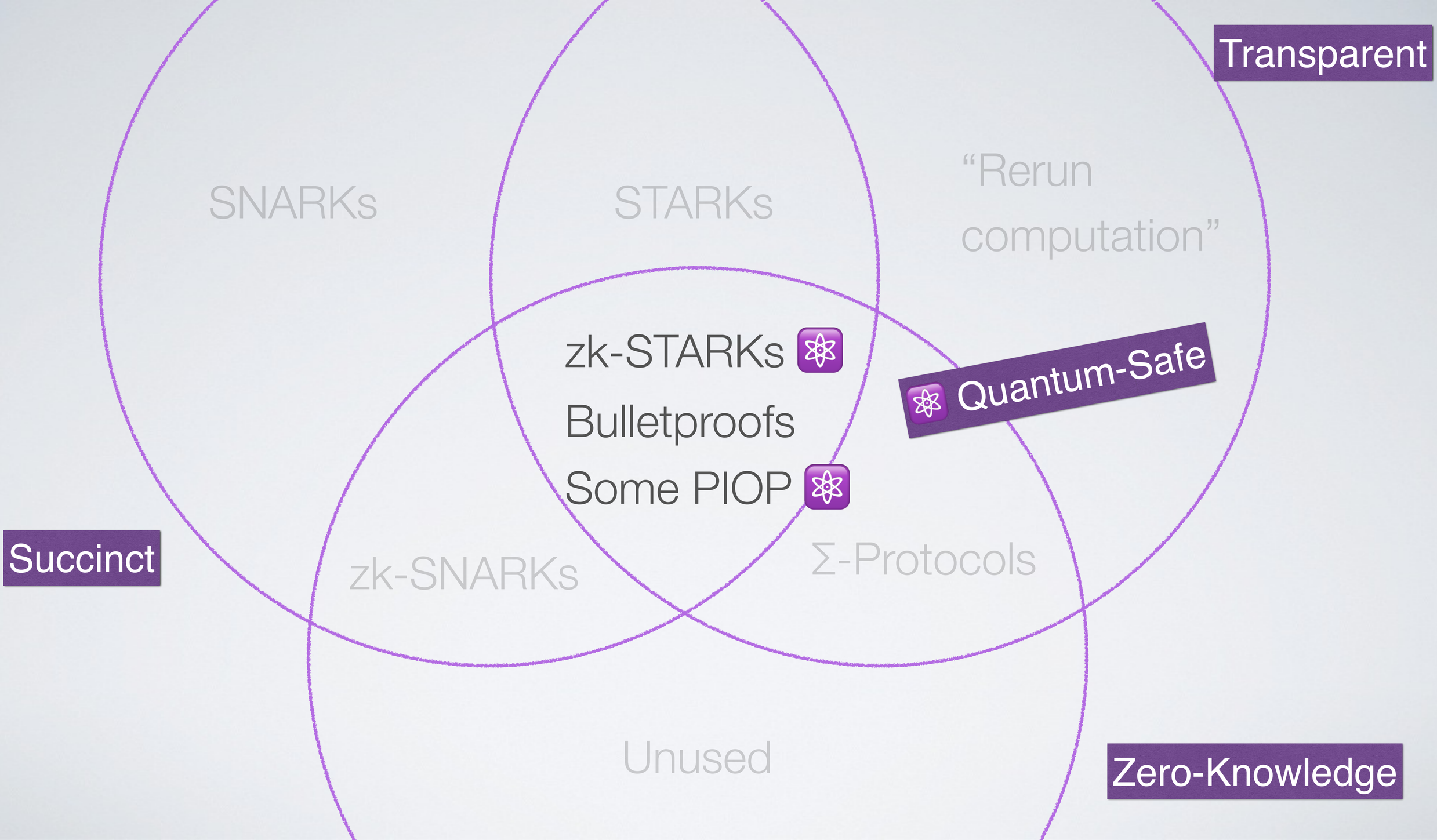
30 people do this

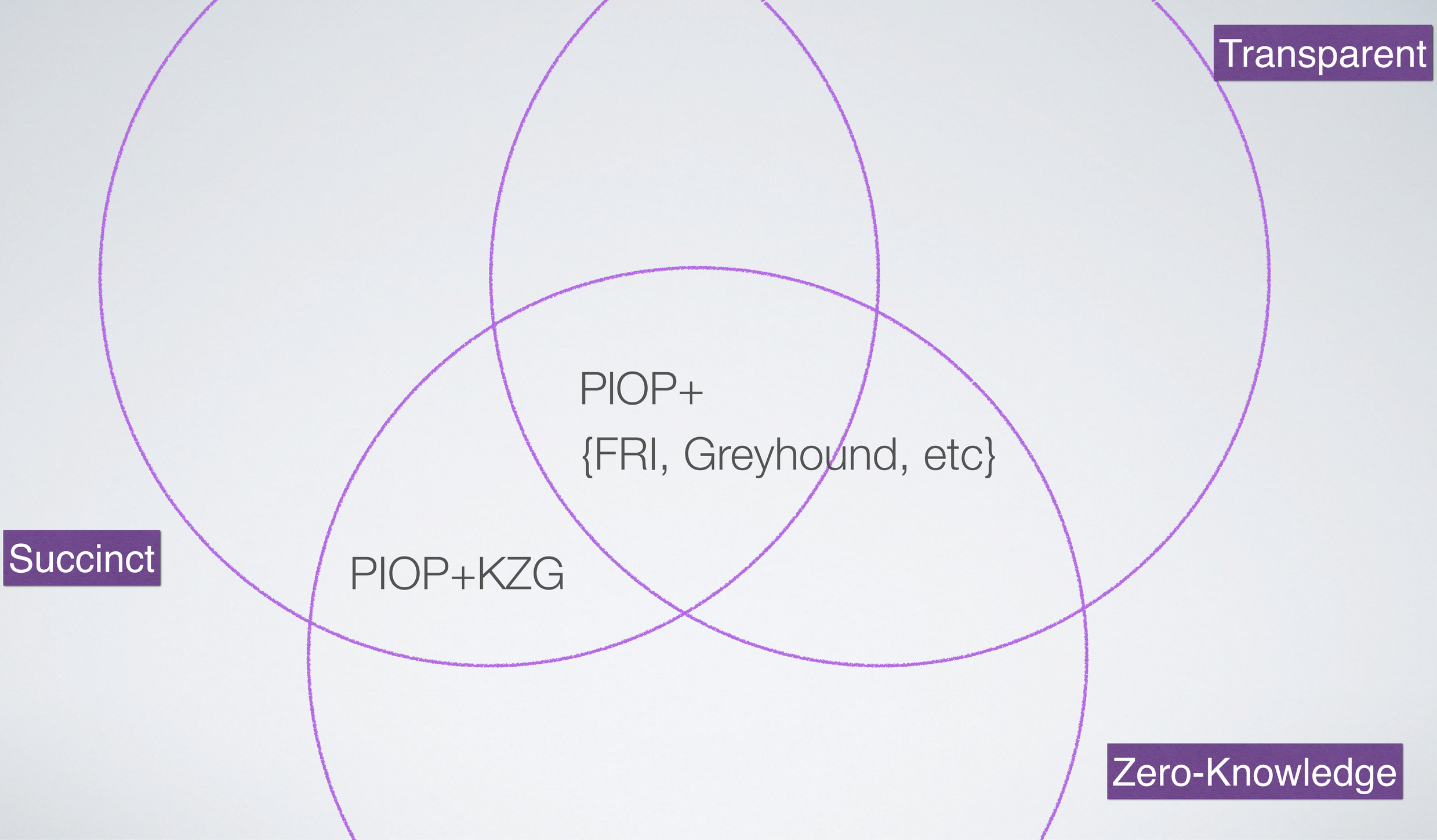
XRP balance

3 people do this

43 Setups	3 Setups	1 Setup	0 Setups
Groth16	Plonk/Halo	STARK	


Transparent
SNARKs
STARKs
“Rerun computation”
zk-STARKs
Bulletproofs
Some PIOP
Succinct
zk-SNARKs
Σ-Protocols
Unused
Zero-Knowledge


Transparent
SNARKs
STARKs
“Rerun computation”
zk-STARKs
Bulletproofs
Some PIOP
Quantum-Safe
Succinct
zk-SNARKs
Σ-Protocols
Unused
Zero-Knowledge


Transparent
PIOP+
{FRI, Greyhound, etc}
Succinct
PIOP+KZG
Zero-Knowledge

PIOP+KZG

Smallest proofs

Faster verifier

Trusted setup

q-SDH (not PQ)

PIOP+Greyhound

Middle ground

Newer assumptions (MSIS)

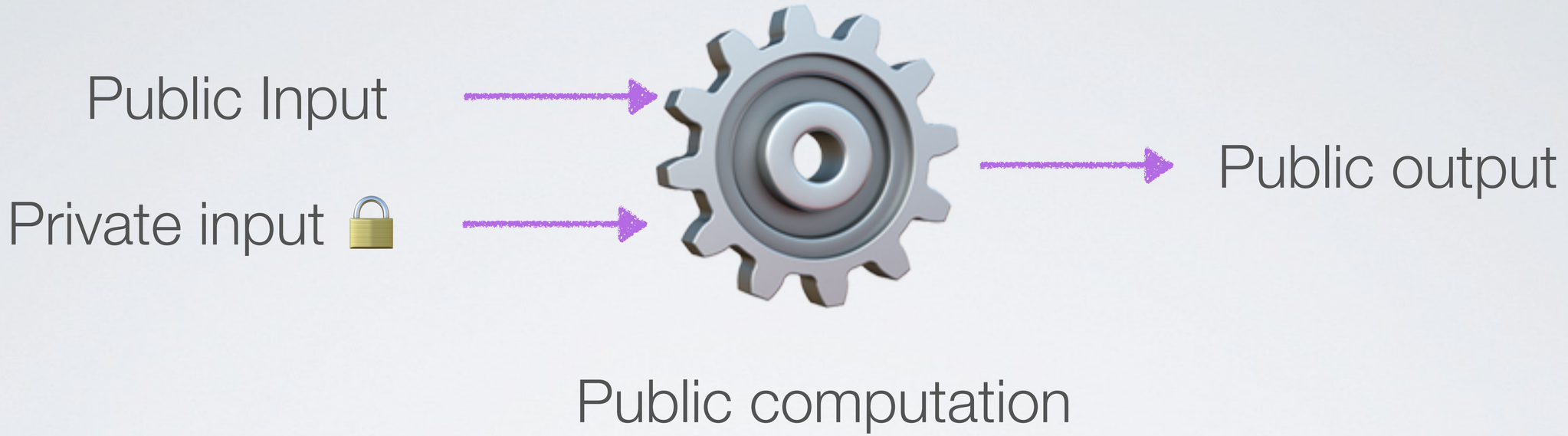
PIOP+FRI

Larger proofs

Slower verifier

Only hashes

General Purpose ZKP



Computation is one-shot, anything computable

General purpose ZKPs provide a programming language, complier, execution trace (circuits or VMs)

ZKP vs. MPC

ZKPs

- Someone knows the secrets
- Don’t trust them to do the computation

Multiparty Computation

- No individual knows the secrets (shared amongst parties)
- Computation can be done without learning the secrets

ZKP vs. MPC

ZKPs

- Someone knows the secrets
- Don’t trust them to do the computation

FHE

- Someone trusted does all the computation
- Do not learn the secrets

Multiparty Computation

- No individual knows the secrets (shared amongst parties)
- Computation can be done without learning the secrets

ZKP vs. MPC

ZKPs

- Someone knows the secrets
- Don’t trust them to do the computation

FHE

- Someone trusted does all the computation
- Do not learn the secrets

Multiparty Computation

- No individual knows the secrets (shared amongst parties)
- Computation can be done without learning the secrets

Hybrids

- MPC-in-the-head, Co-SNARKs, etc.

Three Use-Cases

- **Xiezhi: Toward Succinct Proofs of Solvency**
  Youwei Deng, Jeremy Clark
  WTSC 2026 (@ Financial Cryptography 2026)
  - **Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges**
    Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh
    CCS 2015 (+ Real World Crypto)
- **Zeeperio: Verifying Governmental Elections with Ethereum**
  Aikamdeep Malhotra, Aleksander Essex, Jeremy Clark
  WTSC 2026 (@ Financial Cryptography 2026)
  - **Eperio: Mitigating Technical Complexity in Cryptographic Election Verification**
    Aleks Essex, Jeremy Clark, Urs Hengartner, and Carlisle Adams
    EVT/WOTE 2010 (@ USENIX Security 2010)
- **Zeequent Batch Auctions**
  Kimia Esmaili, Elizabeth van Oorschot, Jeremy Clark
  To be submitted

Proofs of Solvency

- **Xiezhi: Toward Succinct Proofs of Solvency**
  Youwei Deng, Jeremy Clark
  WTSC 2026 (@ Financial Cryptography 2026)
  - **Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges**
    Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh
    CCS 2015 (+ Real World Crypto)
- Zeeperio: Verifying Governmental Elections with Ethereum
  Aikamdeep Malhotra, Aleksander Essex, Jeremy Clark
  WTSC 2026 (@ Financial Cryptography 2026)
  - Eperio: Mitigating Technical Complexity in Cryptographic Election Verification
    Aleks Essex, Jeremy Clark, Urs Hengartner, and Carlisle Adams
    EVT/WOTE 2010 (@ USENIX Security 2010)
- Zeequent Batch Auctions
  Kimia Esmaili, Elizabeth van Oorschot, Jeremy Clark
  To be submitted

Proofs of Solvency

1. Say you want to buy Bitcoin with Canadian dollars, you might use an exchange service
2. To use, you must first deposit your CAD/crypto into the exchange (why?)
3. Exchange might gamble with user deposits or get hacked and not disclose this to users (Mt Gox $450M, QuadrigaCX $130M, FTX $8B, etc.)
4. We can solve this with zero knowledge!

Proofs of Solvency

1. Some caveats apply:
    1. Relies on human behaviour (checking the proof)
    2. It *detects* insolvency, it does not *prevent* insolvency
    3. Collusion between multiple insolvent exchanges might be possible
2. However covering up insolvency when a proof is needed...
    1. Proves it is fraud and not just incompetence
    2. Adds friction for fraudsters
    3. Adds accomplices and paper trails

Proof of Solvency

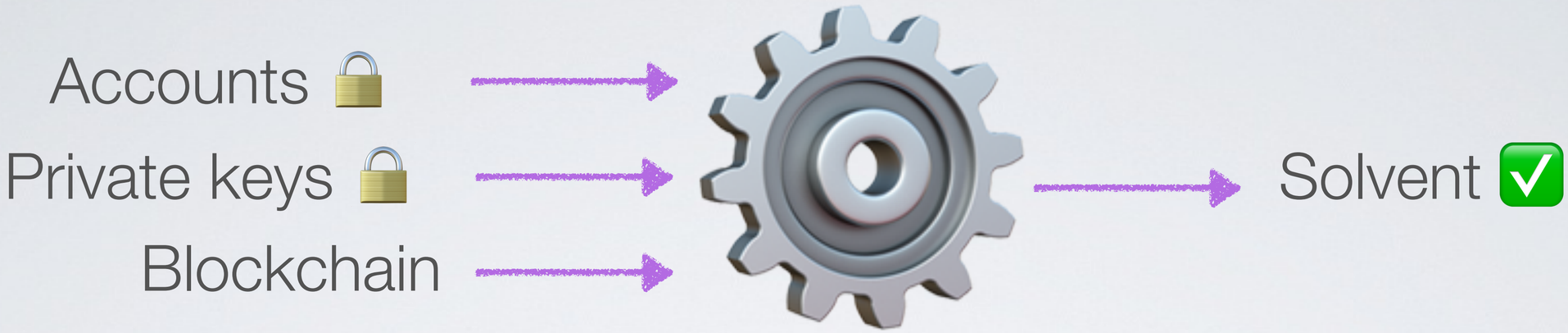
Accounts 🔒

Private keys 🔒

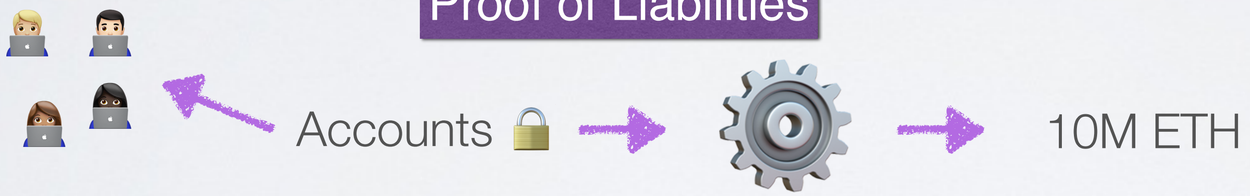
Blockchain

Solvent ✅

Proof of Solvency



Proof of Liabilities



Proof of Assets/Reserves

Solvent

Governmental Elections

- Xiezhi: Toward Succinct Proofs of Solvency
  Youwei Deng, Jeremy Clark
  WTSC 2026 (@ Financial Cryptography 2026)
  - Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges
    Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh
    CCS 2015 (+ Real World Crypto)
- **Zeeperio: Verifying Governmental Elections with Ethereum**
  Aikamdeep Malhotra, Aleksander Essex, Jeremy Clark
  WTSC 2026 (@ Financial Cryptography 2026)
  - **Eperio: Mitigating Technical Complexity in Cryptographic Election Verification**
    Aleks Essex, Jeremy Clark, Urs Hengartner, and Carlisle Adams
    EVT/WOTE 2010 (@ USENIX Security 2010)
- Zeequent Batch Auctions
  Kimia Esmaili, Elizabeth van Oorschot, Jeremy Clark
  To be submitted

Governmental Elections

1. You cast a ballot in an election, how do you know that your ballot was counted correctly? It wasn't modified or changed? The final tally reflects each individual ballot?
2. We can solve this with zero knowledge!

Governmental Elections

1. Some caveats apply:
    1. Relies on human behaviour (checking privacy-preserving ballot receipts are recorded correctly)
    2. It requires the election authority to know the content of every ballot (this can be solved with MPC instead of ZKP: we have done this in real world elections but we continue to face resistances in up-take of these systems. So this is a compromise)

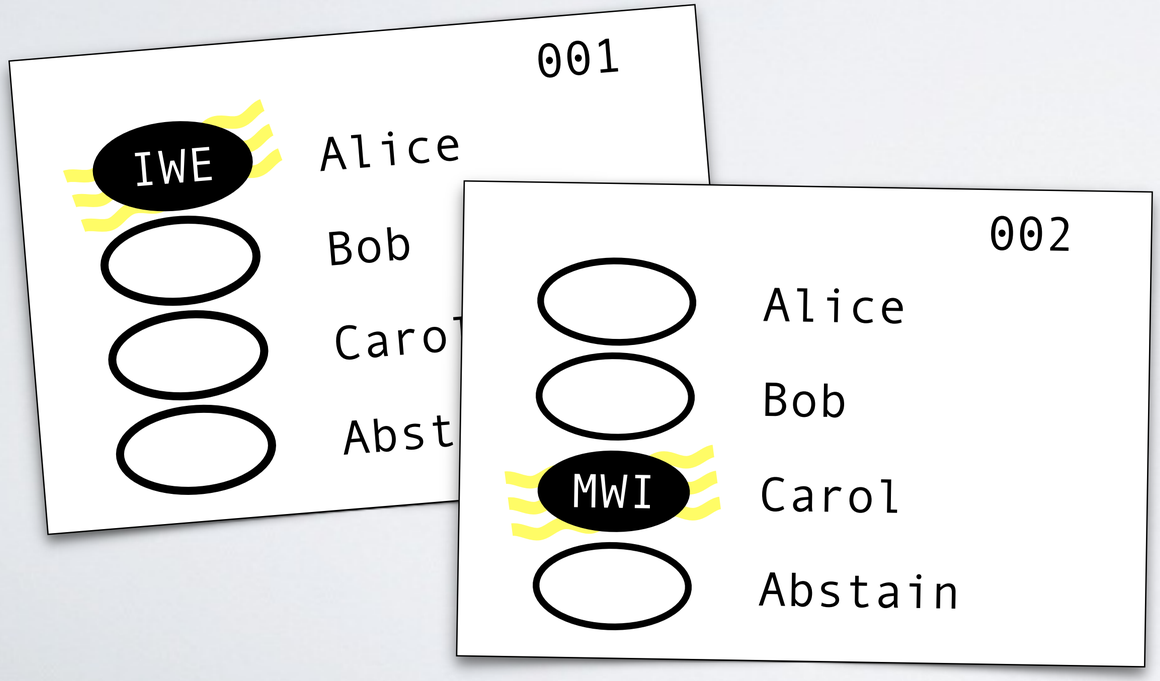
Governmental Elections

Ballots 🔒

Confirmation Code

Bulletin Board

Tally ✅



Ballot	Position	Code	Marked	Selection
001	1	IWE	1	Alice
001	2	XJS	0	Bob
001	3	NIP	0	Carol
001	4	QEM	0	Abstain
002	1	DCE	0	Alice
002	2	OWT	0	Bob
002	3	MWI	1	Carol
002	4	CPW	0	Abstain

Governmental Elections

Ballot Data 🔒
Confirmation Code
Bulletin Board

Tally ✅

Receipt Check

Confirmation Code
Bulletin Board

Code Recorded Correctly

Tally Proof

Ballot Data 🔒
Bulletin Board

Candidates for each Code Tallied Correctly

Tally ✅

Stock Markets

- Xiezhi: Toward Succinct Proofs of Solvency
  Youwei Deng, Jeremy Clark
  WTSC 2026 (@ Financial Cryptography 2026)
  - Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges
    Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh
    CCS 2015 (+ Real World Crypto)
- Zeeperio: Verifying Governmental Elections with Ethereum
  Aikamdeep Malhotra, Aleksander Essex, Jeremy Clark
  WTSC 2026 (@ Financial Cryptography 2026)
  - Eperio: Mitigating Technical Complexity in Cryptographic Election Verification
    Aleks Essex, Jeremy Clark, Urs Hengartner, and Carlisle Adams
    EVT/WOTE 2010 (@ USENIX Security 2010)
- **Zeequent Batch Auctions**
  Kimia Esmaili, Elizabeth van Oorschot, Jeremy Clark
  To be submitted

Stock Markets

1. Say you want to buy NVDA stocks
2. You wait until the price looks good and then you hit “trade” but fast bots change the market conditions before your trade can be executed
3. We can solve this by running short (e.g., 1 second) auctions for stocks (the same way the opening/closing price of NVDA is set)
4. You submit your trade to the auction, and it either fills or not, at some price
5. How do you know you got a fair price? If your trade was only partially filled, how do you know you got a fair amount?
6. We can solve this with zero knowledge!

Governmental Elections

Orderbook Data 🔒

Bulletin Board

Market Clearing Price ✅

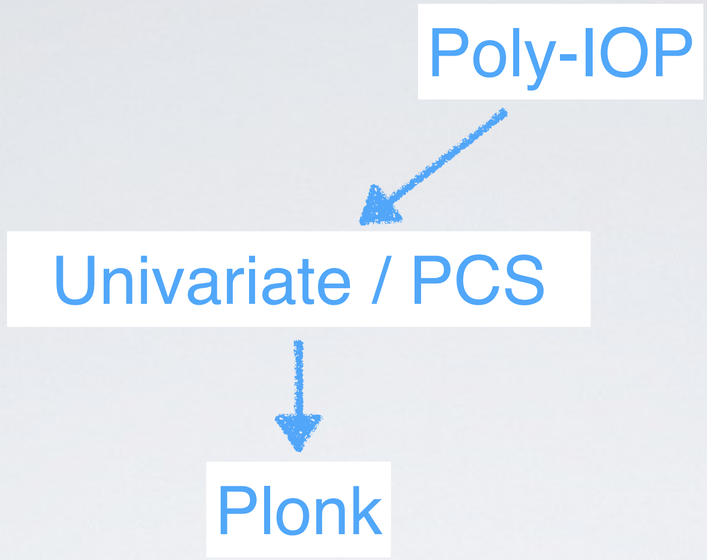
Order Check

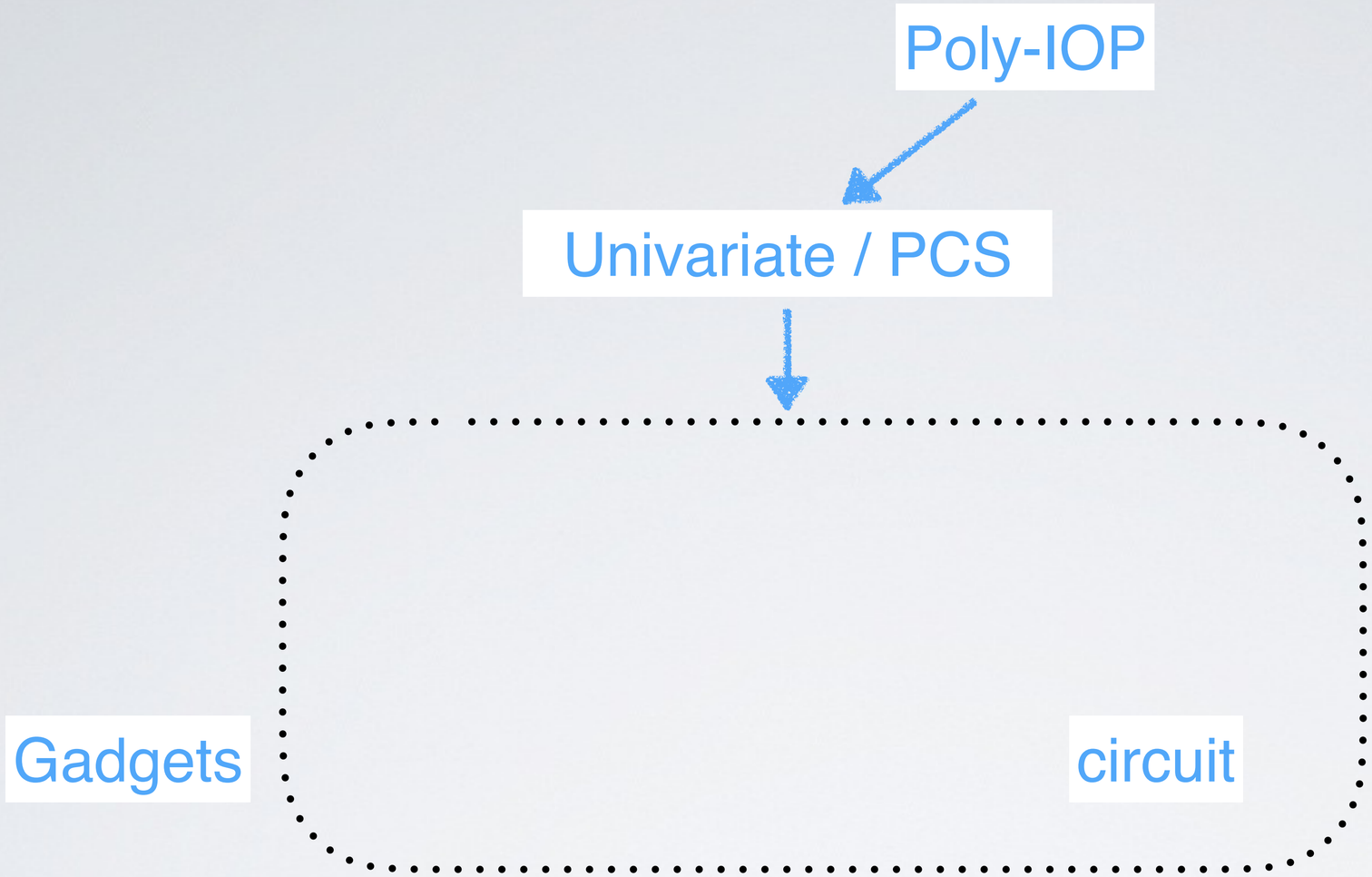
Orderbook Data 🔒 →  → Order Recorded Correctly

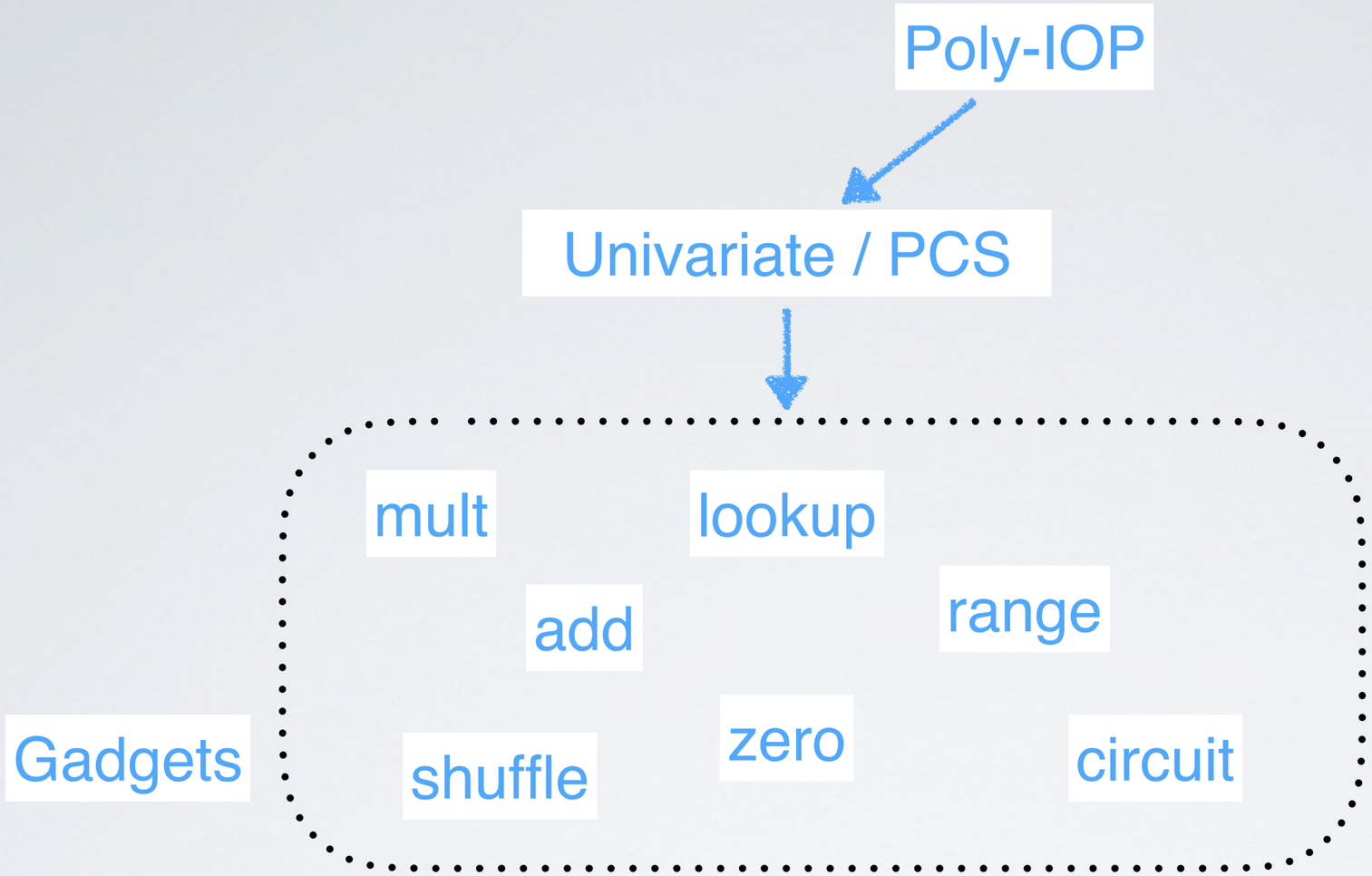
Trade Execution

Orderbook Data 🔒 →  → All Orders Executed Correctly

Order Recorded Correctly, All Orders Executed Correctly → Market Clearing Price ✅


Poly-IOP
Univariate / PCS
Plonk


Poly-IOP
Univariate / PCS
Gadgets
circuit


Poly-IOP
Univariate / PCS
mult
lookup
add
range
Gadgets
zero
shuffle
circuit

Plonkbook: A Handbook for Poly-IOP Gadgets

A handbook for gadgets in the polynomial interactive oracle proof (Poly-IOP) model used by Plonk.

Contributors: Elizabeth van Oorschot (McGill), Youwei Deng (Concordia), Jeremy Clark (Concordia)


```
@misc{vODC24,
      author = {Elizabeth van Oorschot and Youwei Deng and Jeremy Clark},
      title = {Plonkbook: A Handbook for Poly-IOP Gadgets},
      howpublished = {GitHub Pages},
      year = {2024},
      url = {https://madibagroup.github.io/2024-Gadgets}
}
```

Motivation

The zoo of zk-SNARKs is complex to navigate and always changing. Groth16 (by Jens Groth in 2016) was an early protocol that was elegant, explainable, and very performant. A lot of educational resources concentrated on explaining it over its other contemporaries. However a main drawback of Groth16 is a trusted setup that cannot be shared between other people doing Groth16 proofs (unless if they are proving different inputs to the exact same function). Shareable (universal)

Plonkbook: A Hand

A handbook for gadgets in the polyn

Contributors: Elizabeth van Oorsch


```
@misc{vODC24,
      author = {Elizabeth
      title = {Plonkbook:
      howpublished = {GitH
      year = {2024},
      url = {https://madi
}
```

Motivation

The zoo of zk-SNARKs is com
2016) was an early protocol t
educational resources conce
drawback of Groth16 is a tru
proofs (unless if they are pr

Gadget	Short Description	Recap
zero1	$\mathsf{Arr}[i] \leftarrow 0$	Zero-out elements of an array Arr such as: all, first, last, all-but-first, all-but-last.
zero2	$\mathsf{Arr}[i] \leftarrow 0$ iff $\mathsf{Sel}[i] = 0$	Zero-out elements of an array Arr according to a binary array Sel.
rotate	$\mathsf{Arr}[i] \leftarrow \mathsf{Arr}[i + \alpha]$	Rotate array Arr by α positions.
add1	$\mathsf{Arr}_3 = \mathsf{Arr}_1 + \mathsf{Arr}_2$	Array Arr_3 is the element-wise addition of Arr_1 and Arr_2.
add2	$\mathsf{Sum}_{\mathsf{Arr}} = \sum_{i=0}^{n-1} \mathsf{Arr}[i]$	$\mathsf{Sum}_{\mathsf{Arr}}$ is the disclosed sum of all the elements in array Arr.
add3	$\sum_{i=0}^{n-1} \mathsf{Arr}_1[i] = \sum_{i=0}^{n-1} \mathsf{Arr}_2[i]$	Arrays Arr_1 and Arr_2 have the same undisclosed sum.
mult1	$\mathsf{Arr}_3 = \mathsf{Arr}_1 \cdot \mathsf{Arr}_2$	Array Arr_3 is the element-wise multiplication of Arr_1 and Arr_2.
mult2	$\mathsf{Prod}_{\mathsf{Arr}} = \prod_{i=0}^{n-1} \mathsf{Arr}[i]$	$\mathsf{Prod}_{\mathsf{Arr}}$ is the disclosed product of all the elements in array Arr.
mult3	$\prod_{i=0}^{n-1} \mathsf{Arr}_1[i] = \prod_{i=0}^{n-1} \mathsf{Arr}_2[i]$	Arrays Arr_1 and Arr_2 have the same undisclosed product.

Plonkbook: A Hand

A handbook for gadgets in the polyn

Contributors: Elizabeth van Oorsch

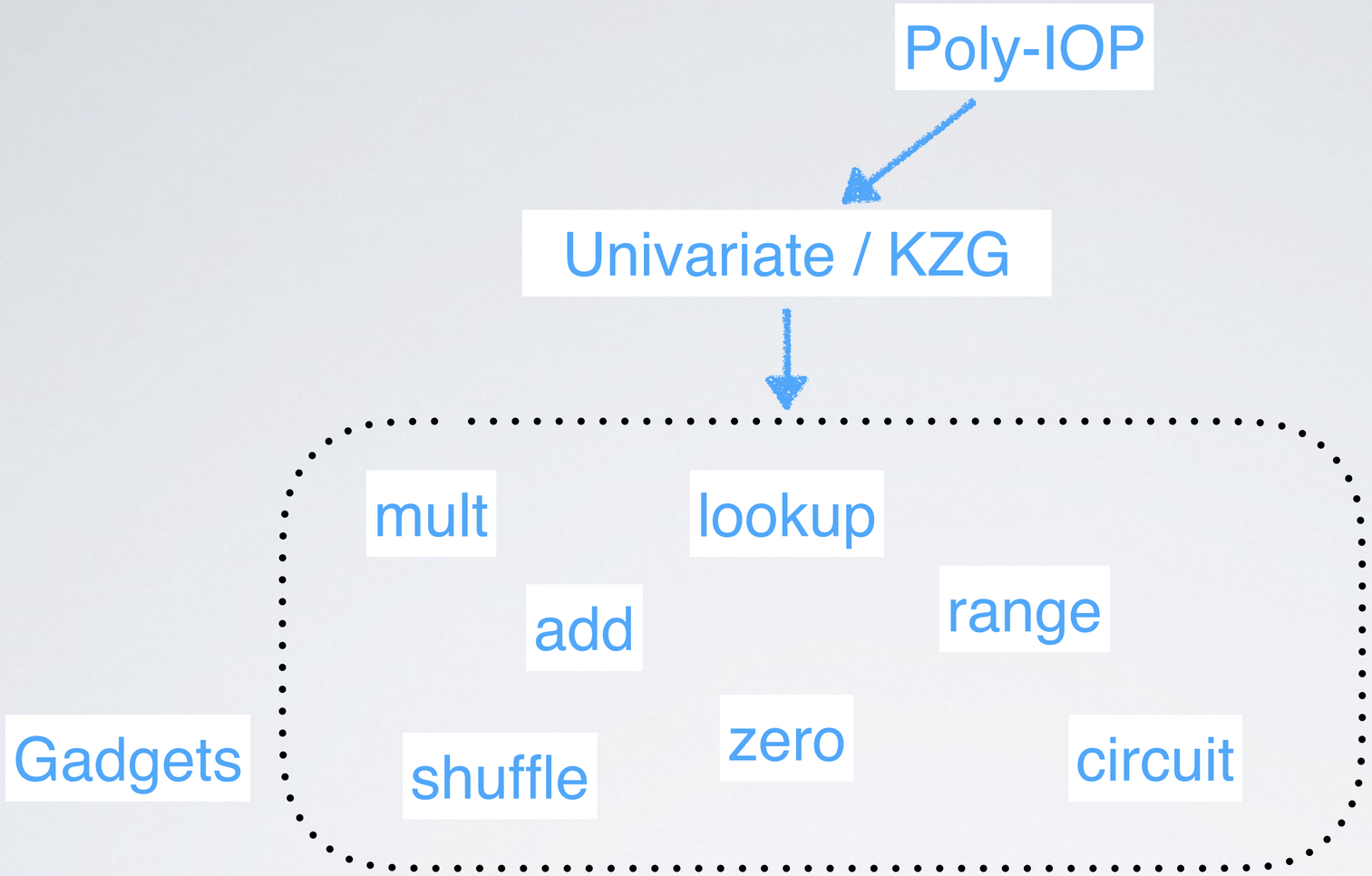

```
@misc{vODC24,
      author = {Elizabeth
      title = {Plonkbook:
      howpublished = {GitH
      year = {2024},
      url = {https://madi
}
```

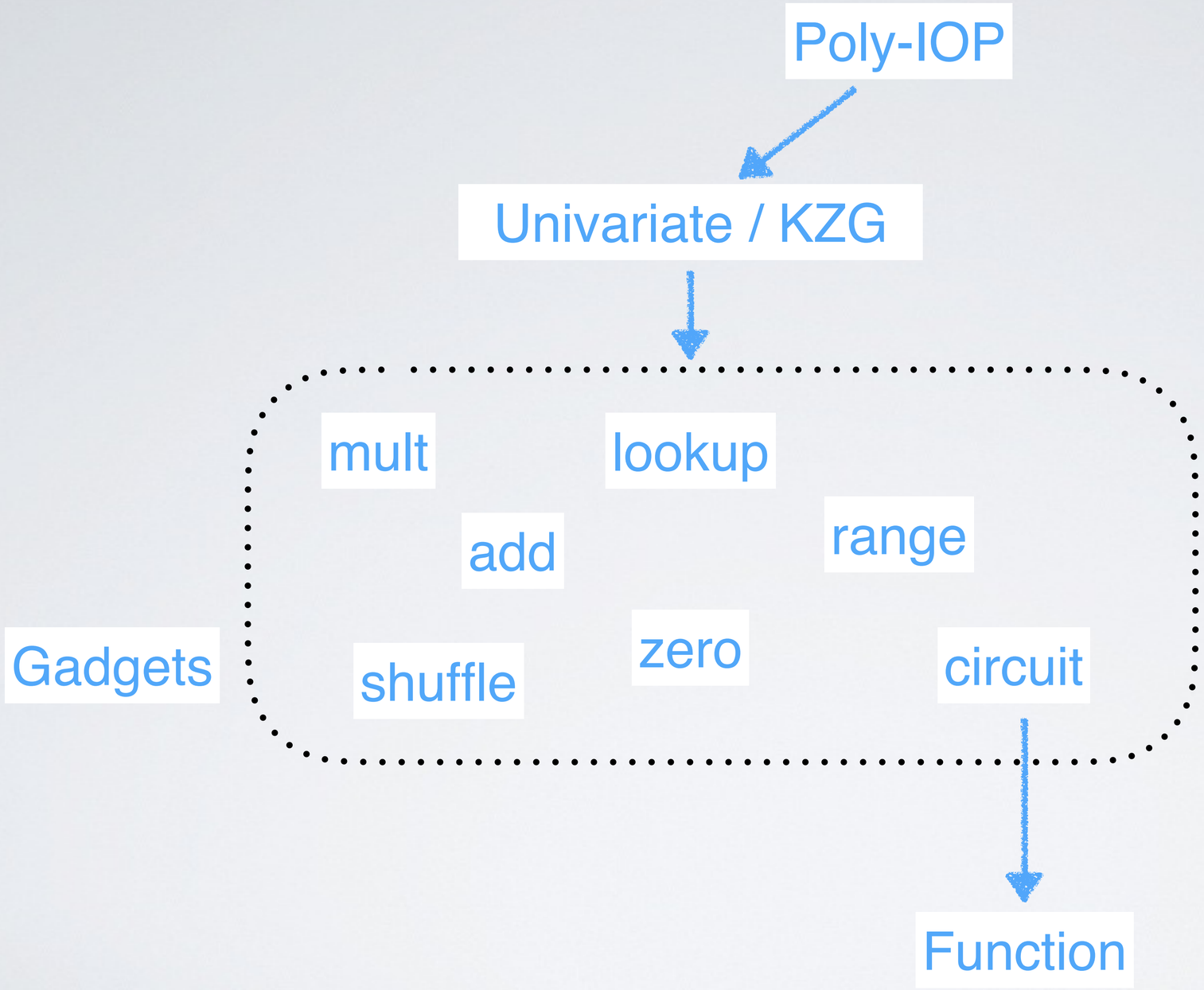
Motivation

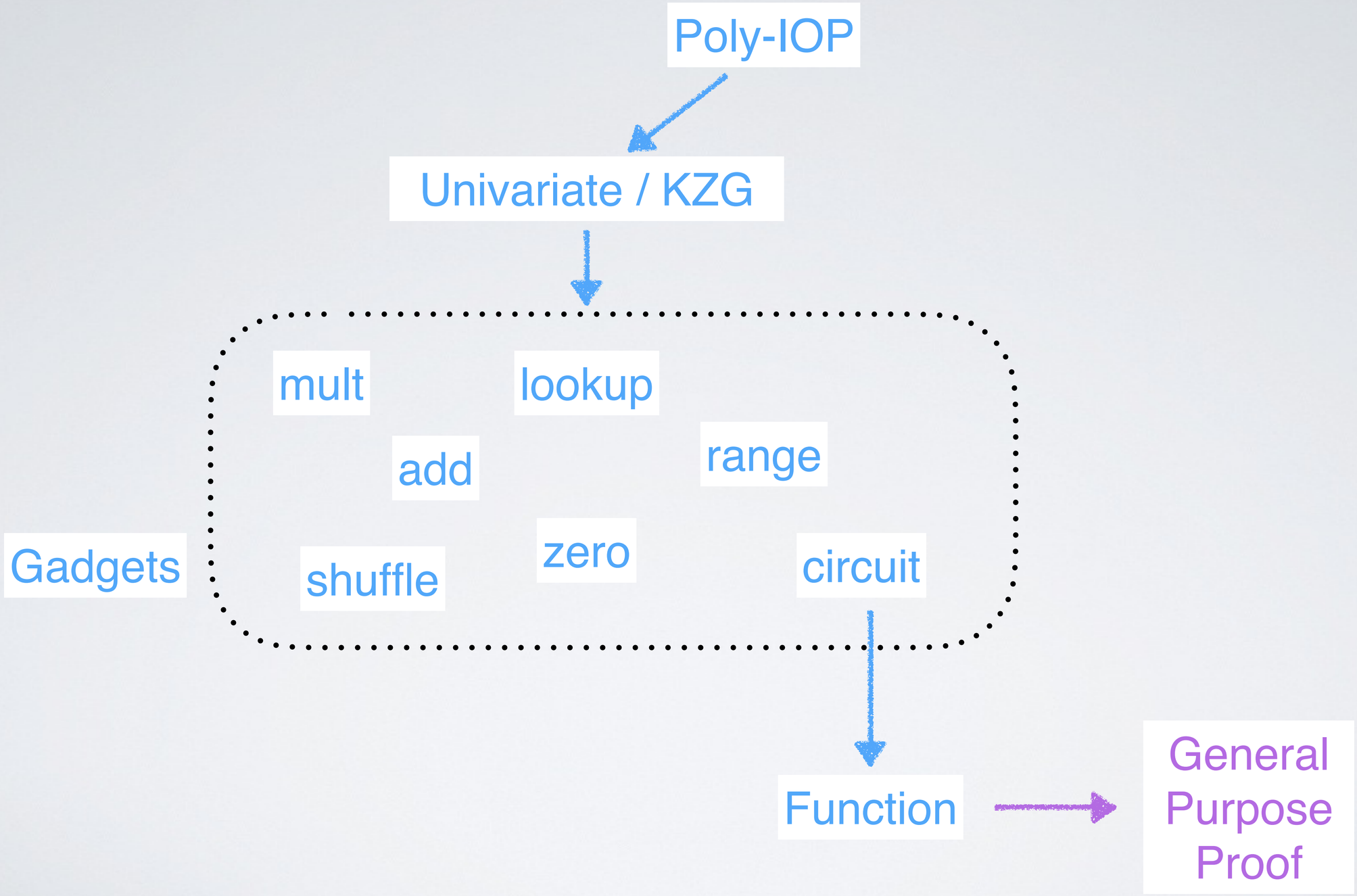
The zoo of zk-SNARKs is com
2016) was an early protocol t
educational resources conce
drawback of Groth16 is a tru
proofs (unless if they are pr

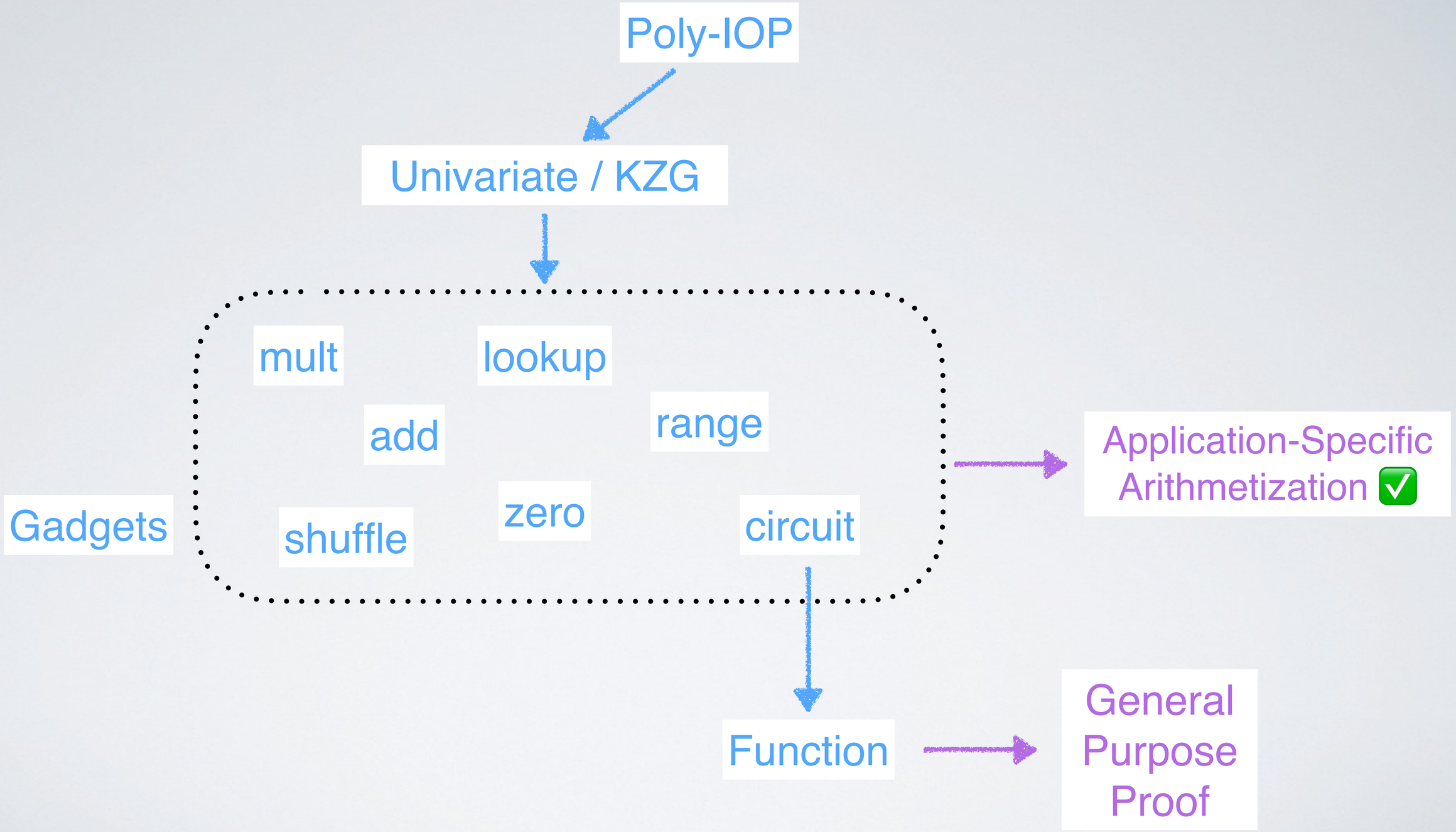
Gadget
zero1
zero2
rotate
add1
add2
add3
mult
mul
mult3

encode	$\mathsf{Arr}_3 \leftarrow \mathsf{Encode}(\mathsf{Arr}_1, \mathsf{Arr}_2)$	Map a set of elements, such as the pair $\{\mathsf{Arr}_1[i], \mathsf{Arr}_2[i]\}$, into a single element $\mathsf{Arr}_3[i]$ without collisions.
shuffle1	$\mathsf{Arr}_2 = \mathsf{Permute}(\mathsf{Arr}_1)$	Array Arr_2 is a shuffle of Arr_1 for some undisclosed permutation π.
shuffle2	$\mathsf{Arr}_2 = \mathsf{Permute}(\mathsf{Arr}_1, \pi)$	Array Arr_2 is a shuffle of Arr_1 under a disclosed permutation π.
lookup1	$\mathsf{Arr}[i] \in \{0, 1\}$	Each element of array Arr is in $\{0, 1\}$ (or another small set).
lookup2	$\mathsf{Arr}[i] \in \mathsf{Table}$	Each element of array Arr is in a disclosed table of values Table.
range	$\mathsf{Arr}[i] \in [0, r]$	Each element of array Arr is in the range $[0, r]$ for some upper-cap r.
circuit	$z = \mathsf{Circ}(x, y)$	z is the output of disclosed arithmetic circuit Circ with disclosed (and/or undisclosed) inputs x and y.
mult3	$\prod_{i=0}^{n-1} \mathsf{Arr}_1[i] = \prod_{i=0}^{} \mathsf{Arr}_2[i]$	undisclosed produ


Poly-IOP
Univariate / KZG
mult
lookup
add
range
zero
Gadgets
shuffle
circuit


Poly-IOP
Univariate / KZG
mult
lookup
add
range
Gadgets
zero
shuffle
circuit
Function


Poly-IOP
Univariate / KZG
mult
lookup
add
range
Gadgets
zero
shuffle
circuit
Function
General Purpose Proof


Poly-IOP
Univariate / KZG
mult
lookup
add
range
zero
shuffle
circuit
Gadgets
Application-Specific Arithmetization ✅
Function
General Purpose Proof

Array → Polynomial → Commitment

Array → Polynomial

3	1	1	7

1. Coefficients

2. Roots

3. Evaluation Points

Array → Polynomial

3	1	1	7

1. Coefficients

$$3 + X + X^2 + 7X^3$$

2. Roots

3. Evaluation Points

Array → Polynomial

3	1	1	7

1. Coefficients

$$3 + X + X^2 + 7X^3$$

2. Roots

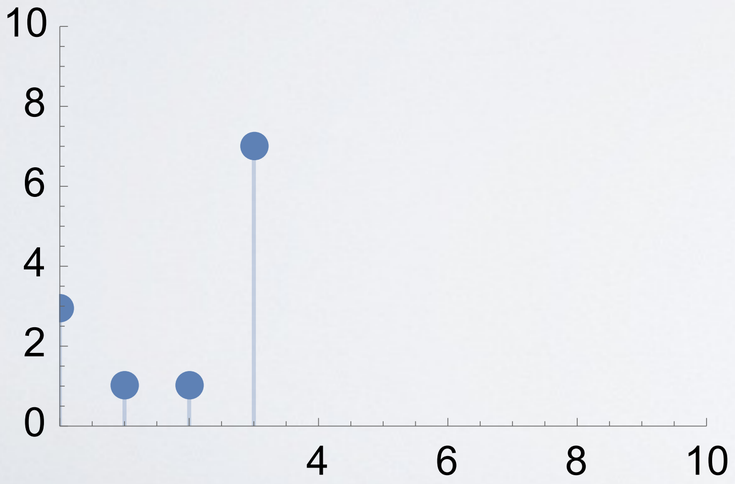
$$21 + 337X + 42X^2 + 377X^3 + X^4$$

3. Evaluation Points

Array → Polynomial

0	1	2	3
3	1	1	7

Domain



1. Coefficients

$$3 + X + X^2 + 7X^3$$

2. Roots

$$21 + 337X + 42X^2 + 377X^3 + X^4$$

3. Evaluation Points

$$122 + 47X + 203X^2 + 20X^3$$

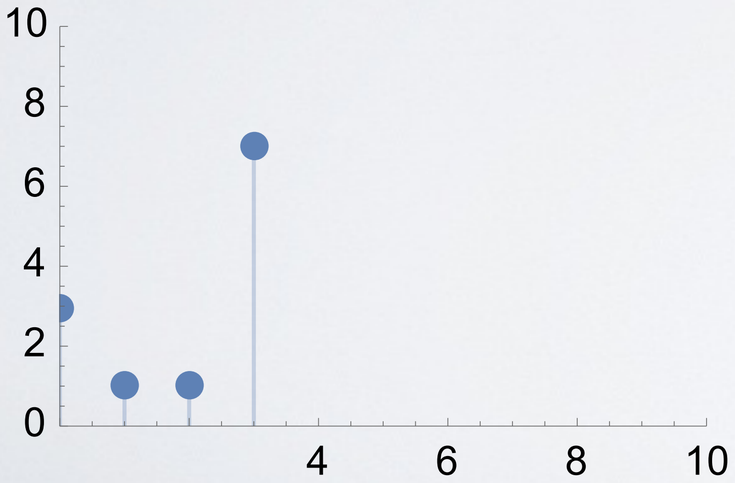
Array → Polynomial

0	1	2	3
3	1	1	7

1. Coefficients fast, can't reopen

$$3 + X + X^2 + 7X^3$$

2. Roots



$$21 + 337X + 42X^2 + 377X^3 + X^4$$

3. Evaluation Points

$$122 + 47X + 203X^2 + 20X^3$$

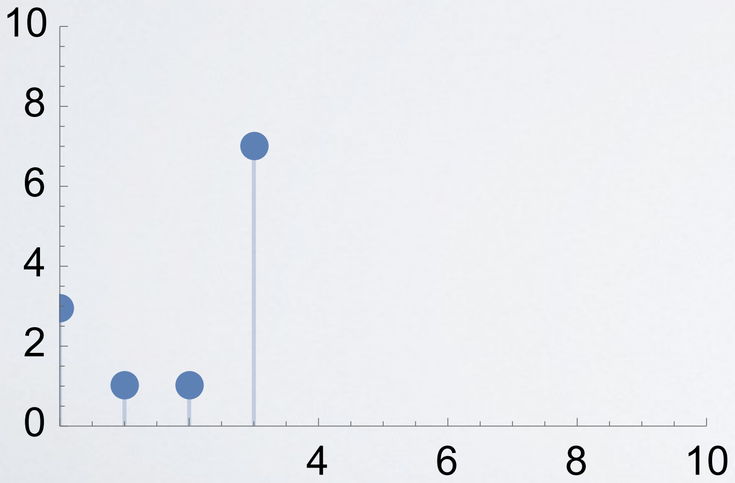
Array → Polynomial

0	1	2	3
3	1	1	7

1. Coefficients fast, can't reopen

$$3 + X + X^2 + 7X^3$$

2. Roots slow, order invariant



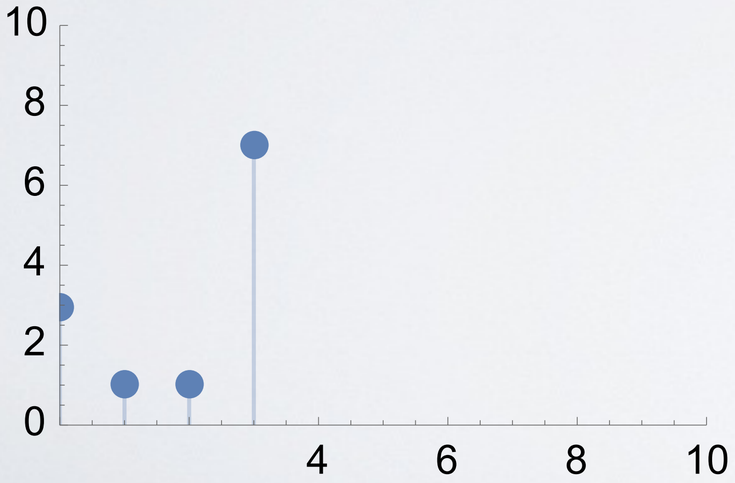
$$21 + 337X + 42X^2 + 377X^3 + X^4$$

3. Evaluation Points

$$122 + 47X + 203X^2 + 20X^3$$

Array → Polynomial

0	1	2	3
3	1	1	7



1. Coefficients fast, can't reopen

$$3 + X + X^2 + 7X^3$$

2. Roots slow, order invariant

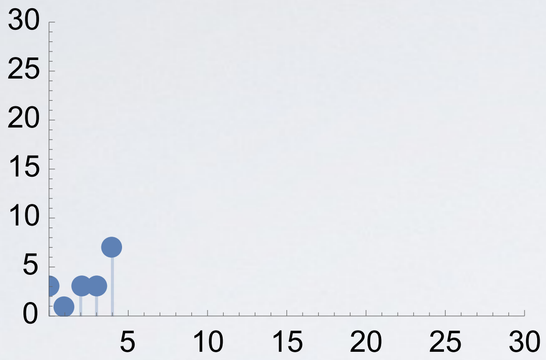
$$21 + 337X + 42X^2 + 377X^3 + X^4$$

3. Evaluation Points slow, flexible ✅

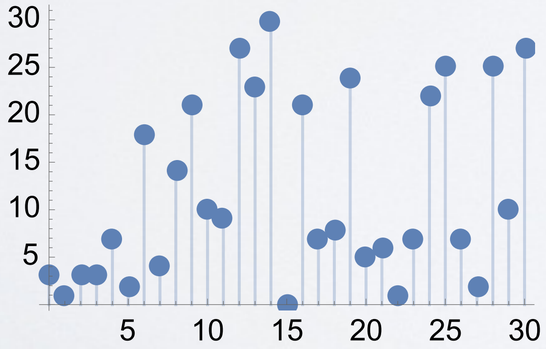
$$122 + 47X + 203X^2 + 20X^3$$

X	0	1	2	3	4
P(X)	3	1	3	3	7

Data



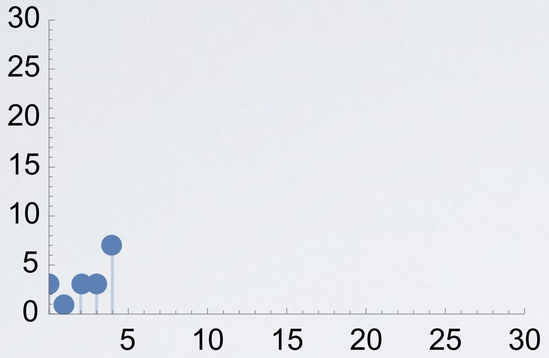
Lagrange interpolation $O(n^2)$



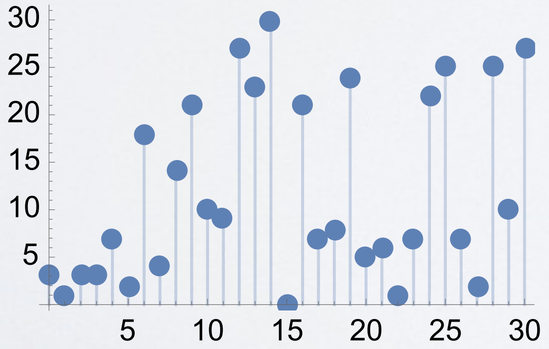
$$3 + 22X + 26X^2 + 27X^3 + 16X^4$$

X	0	1	2	3	4
P(X)	3	1	3	3	7

Data



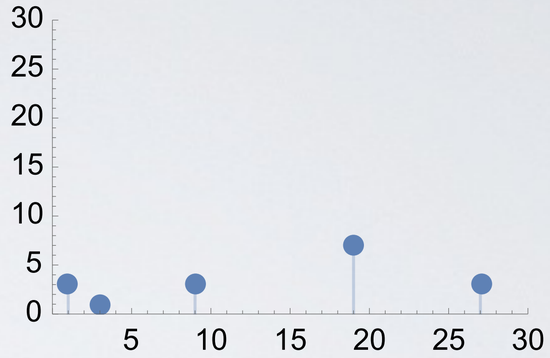
Lagrange interpolation O(n²)



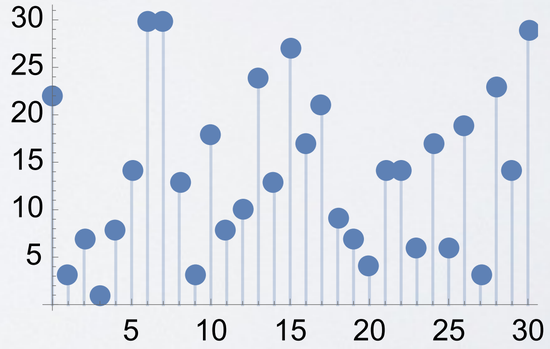
$$3 + 22X + 26X^2 + 27X^3 + 16X^4$$

	$\omega^0 \equiv \omega^5$	ω^1	ω^2	ω^3	ω^4
X	1	3	9	27	19
P(X)	3	1	3	3	7

Data



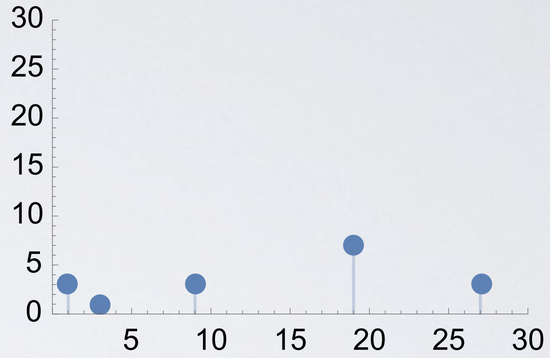
FFT O(*n* log *n*)



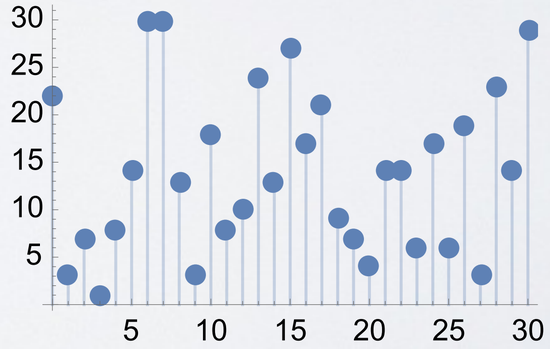
$$22 + 22X + 2X^2 + 27X^3 + 23X^4$$

	$\omega^0 \equiv \omega^5$	ω^1	ω^2	ω^3	ω^4
X	1	3	9	27	19
P(X)	3	1	3	3	7

Data



FFT O(*n* log *n*)



$22 + 22X + 2X^2 + 27X^3 + 23X^4$

ω has multiplicative order *κ* in *q*

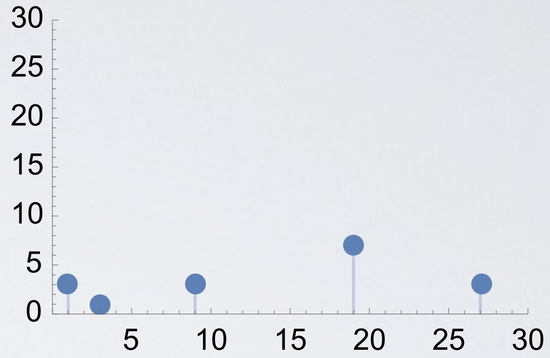
$\omega^\kappa = 1 \bmod q$

$\omega = \sqrt[\kappa]{1}$

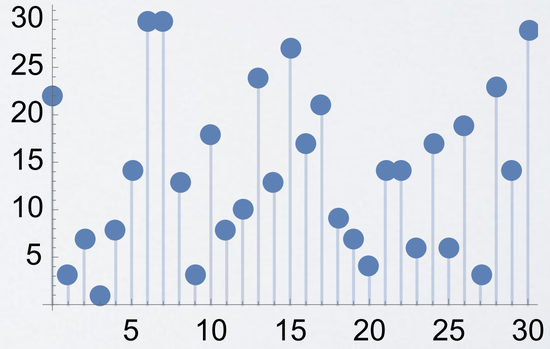
ω is a *κ*-th root of unity

	$\omega^0 \equiv \omega^5$	ω^1	ω^2	ω^3	ω^4
X	1	3	9	27	19
P(X)	3	1	3	3	7

Data



FFT O(*n* log *n*)



$22 + 22X + 2X^2 + 27X^3 + 23X^4$

ω has multiplicative order κ in q

$$\omega^\kappa = 1 \bmod q$$

$$\omega = \sqrt[\kappa]{1}$$

ω is a κ-th root of unity

bls12-381: 2-adicity

ω has multiplicative order κ in q

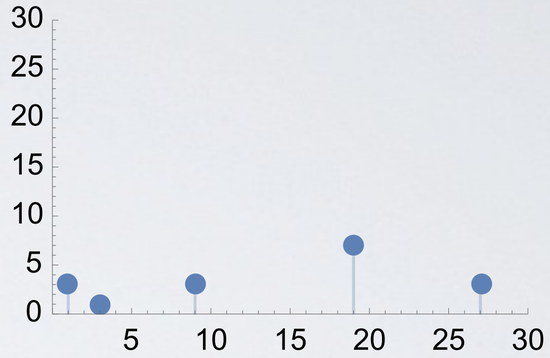
$$\omega^{\kappa} = 1 \bmod q$$

$$\omega = \sqrt[\kappa]{1}$$

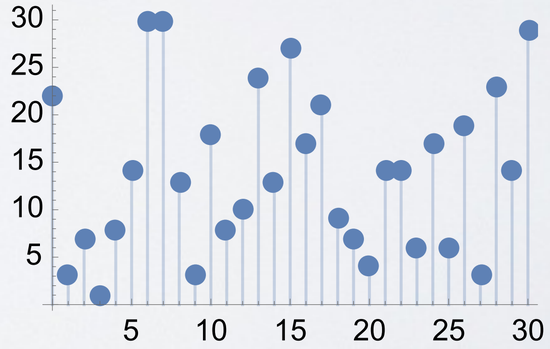
ω is a κ-th root of unity

	$\omega^0 \equiv \omega^5$	ω^1	ω^2	ω^3	ω^4
X	1	3	9	27	19
P(X)	3	1	3	3	7

Data



FFT O(*n* log *n*)



$$22 + 22X + 2X^2 + 27X^3 + 23X^4$$

Array Polynomial → Commitment

$$122 + 47X + 203X^2 + 20X^3$$

Constant-Size Commitments to Polynomials and Their Applications*

Aniket Kate[1], Gregory M. Zaverucha[2], and Ian Goldberg[3]

[1] Max Planck Institute for Software Systems (MPI-SWS)
[2] Certicom Research
[3] University of Waterloo

Abstract. We introduce and formally define polynomial commitment schemes, and provide two efficient constructions. A polynomial commitment scheme allows a committer to commit to a polynomial with a short string that can be used by a verifier to confirm claimed evaluations of the committed polynomial. Although the homomorphic commitment schemes in the literature can be used to achieve this goal, the sizes of their commitments are linear in the degree of the committed polynomial. On the other hand, polynomial commitments in our schemes are of constant size (single elements). The overhead of opening a commitment is also constant; even opening multiple evaluations requires only a constant amount of communication overhead. Therefore, our schemes are useful tools to reduce the communication cost in cryptographic protocols. On that front, we apply our polynomial commit
four problems in cryptography: verifiabl
sets, credentials and co

Array Polynomial → Commitment

$$122 + 47X + 203X^2 + 20X^3$$

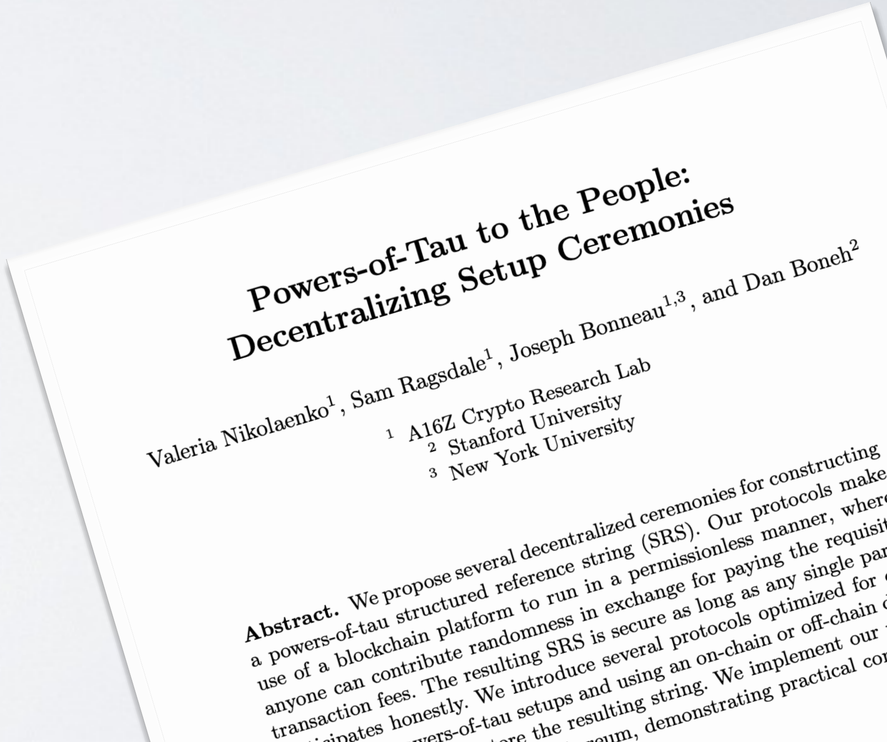
$$[g^{(1)}][g^{(\tau)}][g^{(\tau^2)}][g^{(\tau^3)}]$$

Constant-Size Commitments to Polynomials and Their Applications*

Aniket Kate[1], Gregory M. Zaverucha[2], and Ian Goldberg[3]

[1] Max Planck Institute for Software Systems (MPI-SWS)
[2] Certicom Research
[3] University of Waterloo

Abstract. We introduce and formally define polynomial commitment schemes, and provide two efficient constructions. A polynomial commitment scheme allows a committer to commit to a polynomial with a short string that can be used by a verifier to confirm claimed evaluations of the committed polynomial. Although the homomorphic commitment schemes in the literature can be used to achieve this goal, the sizes of their commitments are linear in the degree of the committed polynomial. On the other hand, polynomial commitments in our schemes are of constant size (single elements). The overhead of opening a commitment is also constant; even opening multiple evaluations requires only a constant amount of communication overhead. Therefore, our schemes are useful tools to reduce the communication cost in cryptographic protocols. On that front, we apply our polynomial commit
four problems in cryptography: verifiabl
sets, credentials and co



Powers-of-Tau to the People: Decentralizing Setup Ceremonies

Valeria Nikolaenko[1], Sam Ragsdale[1], Joseph Bonneau[1,3], and Dan Boneh[2]

[1] A16Z Crypto Research Lab
[2] Stanford University
[3] New York University

Abstract. We propose several decentralized ceremonies for constructing a powers-of-tau structured reference string (SRS). Our protocols make use of a blockchain platform to run in a permissionless manner, where anyone can contribute randomness in exchange for paying the requisite transaction fees. The resulting SRS is secure as long as any single participates honestly. We introduce several protocols optimized for powers-of-tau setups and using an on-chain or off-chain c
re the resulting string. We implement our
eum, demonstrating practical co

Array → Polynomial → Commitment

$$122 + 47X + 203X^2 + 20X^3$$

$$[g^{(1)}][g^{(\tau)}][g^{(\tau^2)}][g^{(\tau^3)}]$$

$$[g]^{122}[g^{\tau}]^{47}[g^{(\tau^2)}]^{203}[g^{(\tau^3)}]^{20}$$

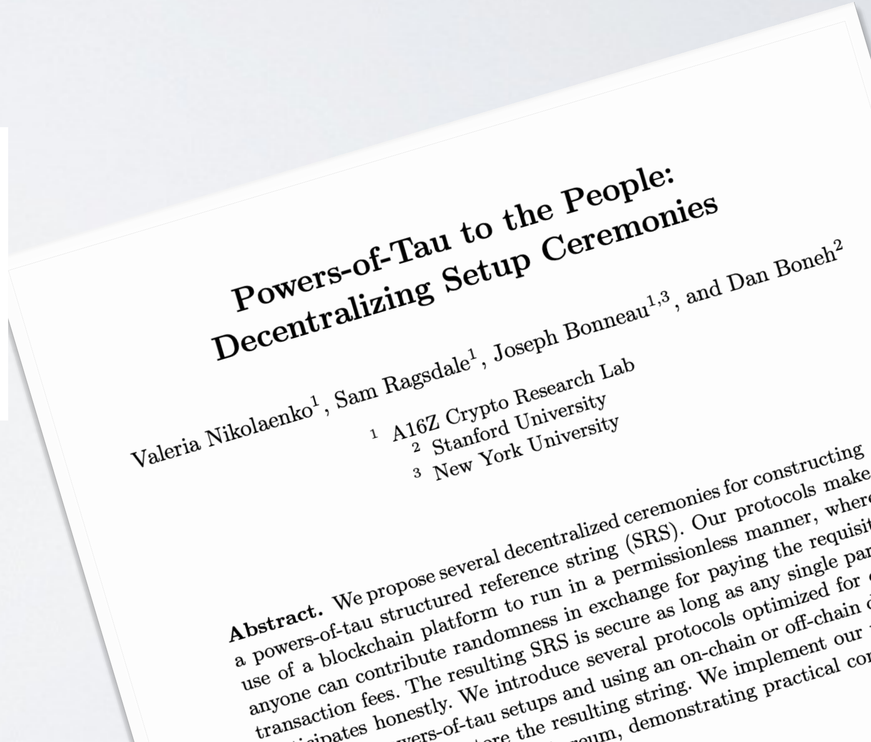
$$g^{122+47\tau+203\tau^2+20\tau^3}$$

Constant-Size Commitments to Polynomials and Their Applications*

Aniket Kate[1], Gregory M. Zaverucha[2], and Ian Goldberg[3]

[1] Max Planck Institute for Software Systems (MPI-SWS)
[2] Certicom Research
[3] University of Waterloo

Abstract. We introduce and formally define polynomial commitment schemes, and provide two efficient constructions. A polynomial commitment scheme allows a committer to commit to a polynomial with a short string that can be used by a verifier to confirm claimed evaluations of the committed polynomial. Although the homomorphic commitment schemes in the literature can be used to achieve this goal, the sizes of their commitments are linear in the degree of the committed polynomial. On the other hand, polynomial commitments in our schemes are of constant size (single elements). The overhead of opening a commitment is also constant; even opening multiple evaluations requires only a constant amount of communication overhead. Therefore, our schemes are useful tools to reduce the communication cost in cryptographic protocols. On that front, we apply our polynomial commit
four problems in cryptography: verifiab
sets, credentials and co



Powers-of-Tau to the People: Decentralizing Setup Ceremonies

Valeria Nikolaenko[1], Sam Ragsdale[1], Joseph Bonneau[1,3], and Dan Boneh[2]

[1] A16Z Crypto Research Lab
[2] Stanford University
[3] New York University

Abstract. We propose several decentralized ceremonies for constructing a powers-of-tau structured reference string (SRS). Our protocols make use of a blockchain platform to run in a permissionless manner, where anyone can contribute randomness in exchange for paying the requisite transaction fees. The resulting SRS is secure as long as any single participates honestly. We introduce several protocols optimized for powers-of-tau setups and using an on-chain or off-chain c
re the resulting string. We implement our
eum, demonstrating practical co

Array → Polynomial → Commitment

$$122 + 47X + 203X^2 + 20X^3$$

$$[g^{(1)}][g^{(\tau)}][g^{(\tau^2)}][g^{(\tau^3)}]$$

$$[g]^{122}[g^{\tau}]^{47}[g^{(\tau^2)}]^{203}[g^{(\tau^3)}]^{20}$$

$$g^{122+47\tau+203\tau^2+20\tau^3}h^r$$

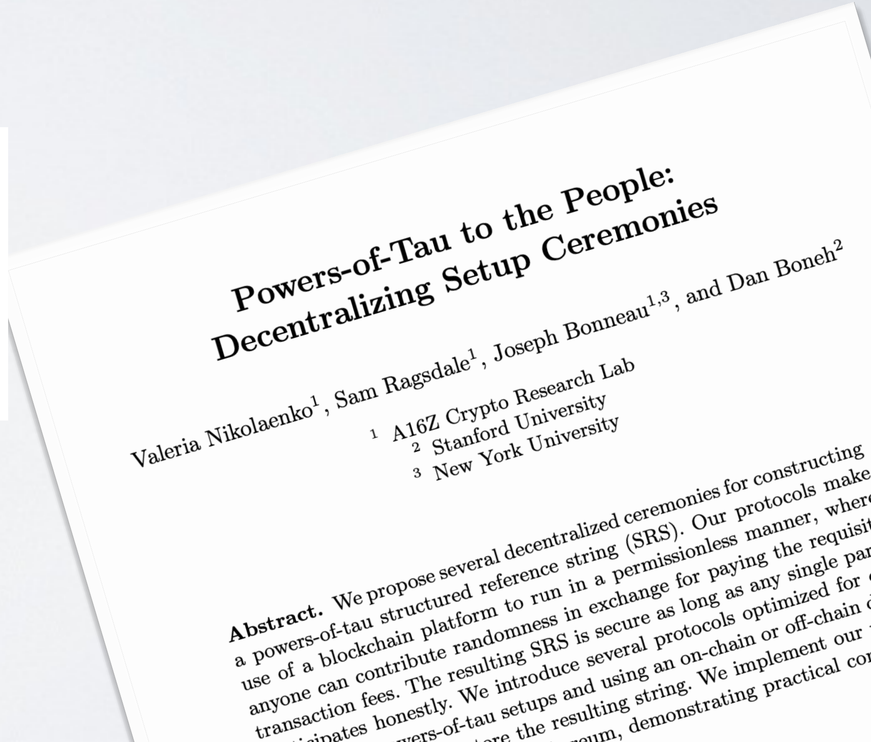
zk

Constant-Size Commitments to Polynomials and Their Applications*

Aniket Kate[1], Gregory M. Zaverucha[2], and Ian Goldberg[3]

[1] Max Planck Institute for Software Systems (MPI-SWS)
[2] Certicom Research
[3] University of Waterloo

Abstract. We introduce and formally define polynomial commitment schemes, and provide two efficient constructions. A polynomial commitment scheme allows a committer to commit to a polynomial with a short string that can be used by a verifier to confirm claimed evaluations of the committed polynomial. Although the homomorphic commitment schemes in the literature can be used to achieve this goal, the sizes of their commitments are linear in the degree of the committed polynomial. On the other hand, polynomial commitments in our schemes are of constant size (single elements). The overhead of opening a commitment is also constant; even opening multiple evaluations requires only a constant amount of communication overhead. Therefore, our schemes are useful tools to reduce the communication cost in cryptographic protocols. On that front, we apply our polynomial commit[ment] four problems in cryptography: verifiable sets, credentials and cre[dentials]



Powers-of-Tau to the People: Decentralizing Setup Ceremonies

Valeria Nikolaenko[1], Sam Ragsdale[1], Joseph Bonneau[1,3], and Dan Boneh[2]

[1] A16Z Crypto Research Lab
[2] Stanford University
[3] New York University

Abstract. We propose several decentralized ceremonies for constructing a powers-of-tau structured reference string (SRS). Our protocols make use of a blockchain platform to run in a permissionless manner, where anyone can contribute randomness in exchange for paying the requisite transaction fees. The resulting SRS is secure as long as any single participates honestly. We introduce several protocols optimized for [different] powers-of-tau setups and using an on-chain or off-chain [component] where the resulting string. We implement our [protocol] scheme, demonstrating practical [costs]

Array Polynomial → Commitment

Pairing

kzg . commit($P(X)$)
kzg . open($P(X)$,3) = 7

Constant-Size Commitments to Polynomials and Their Applications*

Aniket Kate[1], Gregory M. Zaverucha[2], and Ian Goldberg[3]

[1] Max Planck Institute for Software Systems (MPI-SWS)
[2] Certicom Research
[3] University of Waterloo

Abstract. We introduce and formally define polynomial commitment schemes, and provide two efficient constructions. A polynomial commitment scheme allows a committer to commit to a polynomial with a short string that can be used by a verifier to confirm claimed evaluations of the committed polynomial. Although the homomorphic commitment schemes in the literature can be used to achieve this goal, the sizes of their commitments are linear in the degree of the committed polynomial. On the other hand, polynomial commitments in our schemes are of constant size (single elements). The overhead of opening a commitment is also constant; even opening multiple evaluations requires only a constant amount of communication overhead. Therefore, our schemes are useful tools to reduce the communication cost in cryptographic protocols. On that front, we apply our polynomial commit-
four problems in cryptography: verifiabl-
sets, credentials and co-

	ω^0	ω^1	ω^2	ω^3
70	0	68	0	2

Gadget	Short Description	Recap
zero1	$\mathsf{Arr}[i] \leftarrow 0$	Zero-out elements of an array Arr such as: all, first, last, all-but-first, all-but-last.
zero2	$\mathsf{Arr}[i] \leftarrow 0$ iff $\mathsf{Sel}[i] = 0$	Zero-out elements of an array Arr according to a binary array Sel.
rotate	$\mathsf{Arr}[i] \leftarrow \mathsf{Arr}[i+\alpha]$	Rotate array Arr by α positions.
add1	$\mathsf{Arr}_3 = \mathsf{Arr}_1 + \mathsf{Arr}_2$	Array Arr_3 is the element-wise addition of Arr_1 and Arr_2.
add2	$\mathsf{Sum}_{\mathsf{Arr}} = \sum_{i=0}^{n-1} \mathsf{Arr}[i]$	$\mathsf{Sum}_{\mathsf{Arr}}$ is the disclosed sum of all the elements in array Arr.
add3	$\sum_{i=0}^{n-1} \mathsf{Arr}_1[i] = \sum_{i=0}^{n-1} \mathsf{Arr}_2[i]$	Arrays Arr_1 and Arr_2 have the same undisclosed sum.
mult1	$\mathsf{Arr}_3 = \mathsf{Arr}_1 \cdot \mathsf{Arr}_2$	Array Arr_3 is the element-wise multiplication of Arr_1 and Arr_2.
mult2	$\mathsf{Prod}_{\mathsf{Arr}} = \prod_{i=0}^{n-1} \mathsf{Arr}[i]$	$\mathsf{Prod}_{\mathsf{Arr}}$ is the disclosed product of all the elements in array Arr.
mult3	$\prod_{i=0}^{n-1} \mathsf{Arr}_1[i] = \prod_{i=0}^{n-1} \mathsf{Arr}_2[i]$	Arrays Arr_1 and Arr_2 have the same undisclosed product.

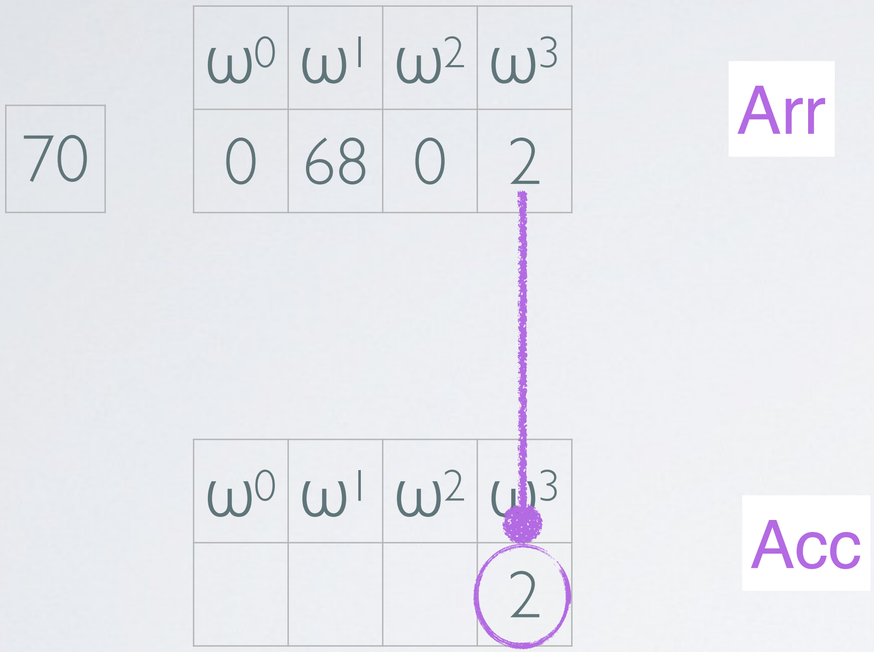
	ω^0	ω^1	ω^2	ω^3
70	0	68	0	2

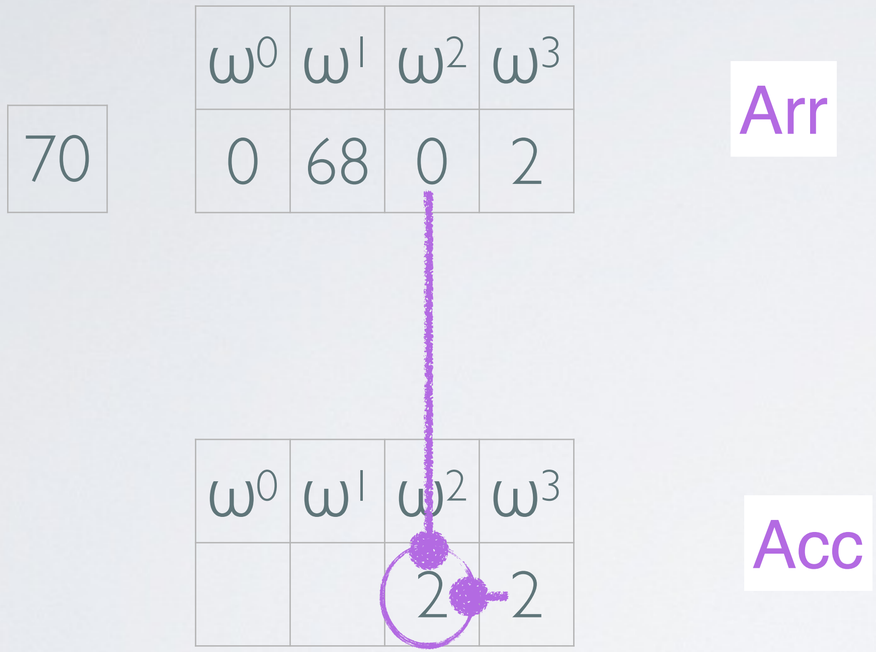
Arr

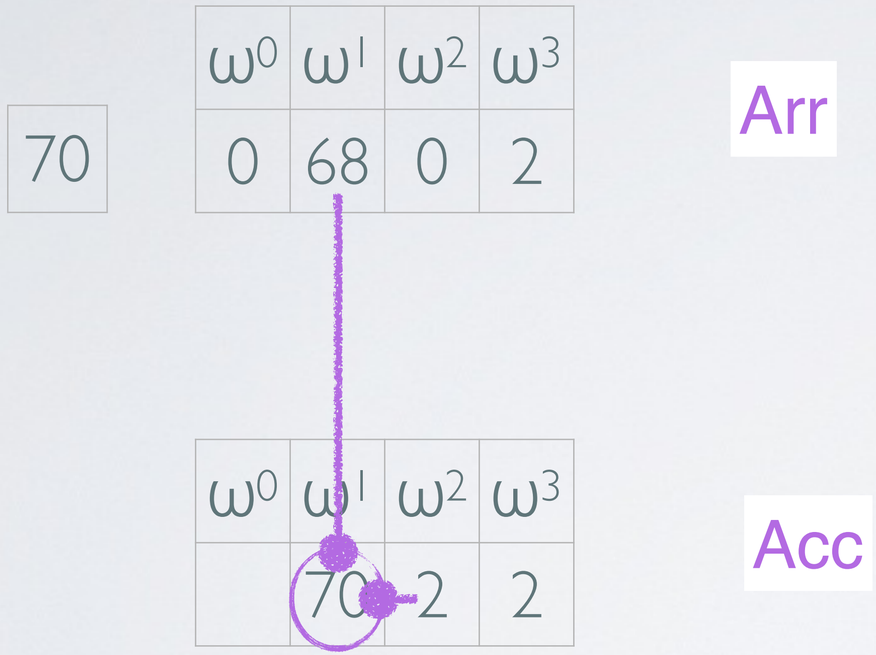
ω^0	ω^1	ω^2	ω^3

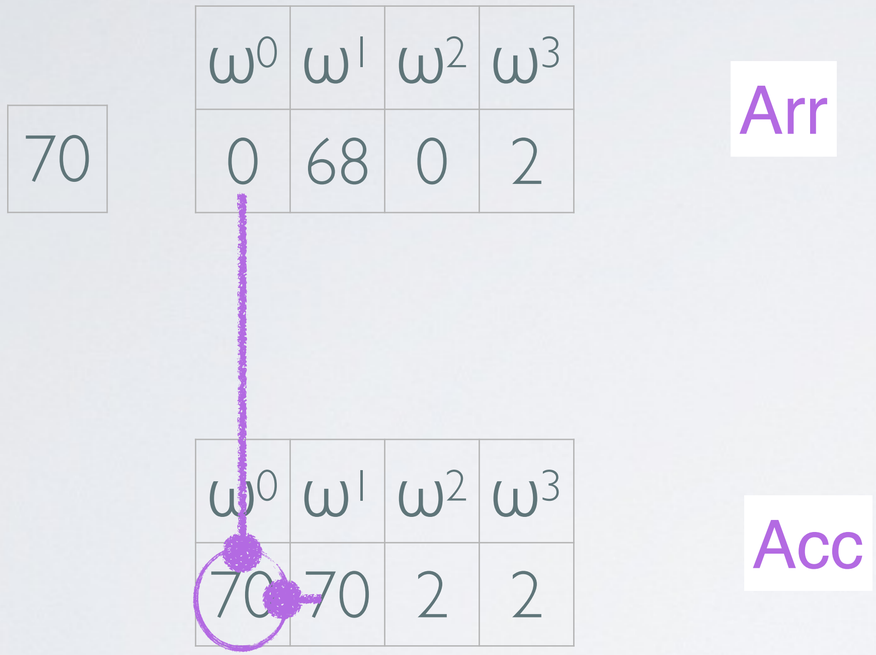
Acc

Helper polynomial


70
ω0 ω1 ω2 ω3
0 68 0 2
Arr
ω0 ω1 ω2 ω3
2
Acc


ω0 ω1 ω2 ω3
70
0 68 0 2
Arr
ω0 ω1 ω2 ω3
2 2
Acc


ω0 ω1 ω2 ω3
70
0 68 0 2
Arr
ω0 ω1 ω2 ω3
70 2 2
Acc


70
ω0 ω1 ω2 ω3
0 68 0 2
Arr
ω0 ω1 ω2 ω3
70 70 2 2
Acc

	ω^0	ω^1	ω^2	ω^3
70	0	68	0	2

Arr

Sum’s “correctness” depends on every other cell

ω^0	ω^1	ω^2	ω^3
70	70	2	2

Acc

Each cell’s “correctness” depends on only 2 other cells

1. The first value in Acc matches the first value in Arr,
2. The rest of the values in Acc are of the form $\mathsf{Acc}[i] = \mathsf{Arr}[i] + \mathsf{Acc}[i-1]$,
3. The last value in Acc matches $\mathsf{Sum}_{\mathsf{Arr}}$.

	ω^0	ω^1	ω^2	ω^3
70	0	68	0	2

Arr

ω^0	ω^1	ω^2	ω^3
70	70	2	2

Acc

1. The first value in Acc matches the first value in Arr,
2. The rest of the values in Acc are of the form $\mathsf{Acc}[i] = \mathsf{Arr}[i] + \mathsf{Acc}[i-1]$,
3. The last value in Acc matches $\mathsf{Sum}_{\mathsf{Arr}}$.

1. For $X = w^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X)$,
2. For all X except $X = \omega^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X)$
3. For $X = w^0$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Sum}_{\mathsf{Arr}}$

	ω^0	ω^1	ω^2	ω^3
70	0	68	0	2

Arr

ω^0	ω^1	ω^2	ω^3
70	70	2	2

Acc

	ω^0	ω^1	ω^2	ω^3
70	0	68	0	2

Arr

ω^0	ω^1	ω^2	ω^3
70	70	2	2

Acc

1. The first value in Acc matches the first value in Arr,
2. The rest of the values in Acc are of the form $\mathsf{Acc}[i] = \mathsf{Arr}[i] + \mathsf{Acc}[i-1]$,
3. The last value in Acc matches $\mathsf{Sum}_{\mathsf{Arr}}$.

1. For $X = w^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X)$,
2. For all X except $X = \omega^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X)$
3. For $X = w^0$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Sum}_{\mathsf{Arr}}$

1. For $X = w^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) = 0$,
2. For all X except $X = \omega^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X) = 0$
3. For $X = w^0$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Sum}_{\mathsf{Arr}} = 0$

1. The first value in Acc matches the first value in Arr,
2. The rest of the values in Acc are of th
3. The last

Operation	Input Array	Input Polynomial	Output Array	Output Polynomial
Zero all	$\langle 3,1,3,3,7\rangle$	$P(X)$	$\langle 0,0,0,0,0\rangle$	$P(X)\cdot(X^n-1)$
Zero first	$\langle 3,1,3,3,7\rangle$	$P(X)$	$\langle 0,\perp,\perp,\perp,\perp\rangle$	$P(X)\cdot(X-\omega^0)$
Zero last	$\langle 3,1,3,3,7\rangle$	$P(X)$	$\langle \perp,\perp,\perp,\perp,0\rangle$	$P(X)\cdot(X-\omega^{n-1})$
Zero all but first	$\langle 3,1,3,3,7\rangle$	$P(X)$	$\langle \perp,0,0,0,0\rangle$	$P(X)\cdot\frac{(X^n-1)}{(X-\omega^0)}$
Zero all but last	$\langle 3,1,3,3,7\rangle$	$P(X)$	$\langle 0,0,0,0,\perp\rangle$	$P(X)\cdot\frac{(X^n-1)}{(X-\omega^{n-1})}$

	ω^0	ω^1	ω^2	ω^3
70	0	68	0	2

Arr

ω^0	ω^1	ω^2	ω^3
70	70	2	2

Acc

1. The first value in Acc matches the first value in Arr,
2. The rest of the values in Acc are of the form $\mathsf{Acc}[i] = \mathsf{Arr}[i] + \mathsf{Acc}[i-1]$,
3. The last value in Acc matches $\mathsf{Sum}_{\mathsf{Arr}}$.

1. For $X = w^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X)$,
2. For all X except $X = \omega^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X)$
3. For $X = w^0$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Sum}_{\mathsf{Arr}}$

1. For $X = w^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) = 0$,
2. For all X except $X = \omega^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X) = 0$
3. For $X = w^0$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Sum}_{\mathsf{Arr}} = 0$

1. $\mathsf{Poly}_{\mathsf{Vanish1}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X)) \cdot \frac{(X^\kappa - 1)}{(X - \omega^{\kappa-1})} = 0$,
2. $\mathsf{Poly}_{\mathsf{Vanish2}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X)) \cdot (X - \omega^{\kappa-1}) = 0$
3. $\mathsf{Poly}_{\mathsf{Vanish3}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Sum}_{\mathsf{Arr}}) \cdot \frac{(X^\kappa - 1)}{(X - \omega^0)} = 0$

	ω^0	ω^1	ω^2	ω^3
70	0	68	0	2

Arr

1. The first value in Acc matches the first value in Arr,
2. The rest of the values in Acc are of the form $\text{Acc}[i] = \text{Arr}[i] + \text{Acc}[i-1]$,
3. The last value in Acc matches Sum_{Arr}.

1. For $X = w^{\kappa-1}$: $\text{Poly}_{\text{Acc}}(X) = \text{Poly}_{\text{Arr}}(X)$,
2. For all X except $X = \omega^{\kappa-1}$: $\text{Poly}_{\text{Acc}}(X) = \text{Poly}_{\text{Arr}}(X) + \text{Poly}_{\text{Acc}}(\omega \cdot X)$
3. For $X = w^0$: $\text{Poly}_{\text{Acc}}(X) = \text{Sum}_{\text{Arr}}$

1. For $X = w^{\kappa-1}$: $\text{Poly}_{\text{Acc}}(X) - \text{Poly}_{\text{Arr}}(X) = 0$,
2. For all X except $X = \omega^{\kappa-1}$: $\text{Poly}_{\text{Acc}}(X) - \text{Poly}_{\text{Arr}}(X) + \text{Poly}_{\text{Acc}}(\omega \cdot X) = 0$
3. For $X = w^0$: $\text{Poly}_{\text{Acc}}(X) - \text{Sum}_{\text{Arr}} = 0$

	ω^0	ω^1	ω^2	ω^3
	70	70	2	2

Acc

1. $\text{Poly}_{\text{Vanish1}}(X) = (\text{Poly}_{\text{Acc}}(X) - \text{Poly}_{\text{Arr}}(X)) \cdot \frac{(X^\kappa - 1)}{(X - \omega^{\kappa-1})} = 0$,
2. $\text{Poly}_{\text{Vanish2}}(X) = (\text{Poly}_{\text{Acc}}(X) - \text{Poly}_{\text{Arr}}(X) + \text{Poly}_{\text{Acc}}(\omega \cdot X)) \cdot (X - \omega^{\kappa-1}) = 0$
3. $\text{Poly}_{\text{Vanish3}}(X) = (\text{Poly}_{\text{Acc}}(X) - \text{Sum}_{\text{Arr}}) \cdot \frac{(X^\kappa - 1)}{(X - \omega^0)} = 0$

1. $Q_1(X) = \frac{\text{Poly}_{\text{Vanish1}}(X)}{X^\kappa - 1}$
2. $Q_2(X) = \frac{\text{Poly}_{\text{Vanish2}}(X)}{X^\kappa - 1}$
3. $Q_3(X) = \frac{\text{Poly}_{\text{Vanish3}}(X)}{X^\kappa - 1}$

1. $\text{Poly}_{\text{Zero}}(X) = \text{Poly}_{\text{Vanish1}}(X) - Q_1(X) \cdot (X^n - 1) = 0$
2. $\text{Poly}_{\text{Zero}}(X) = \text{Poly}_{\text{Vanish2}}(X) - Q_2(X) \cdot (X^n - 1) = 0$
3. $\text{Poly}_{\text{Zero}}(X) = \text{Poly}_{\text{Vanish3}}(X) - Q_3(X) \cdot (X^n - 1) = 0$

1. The first value in Acc matches the first value in Arr,
2. The rest of the values in Acc are of the form $\mathsf{Acc}[i] = \mathsf{Arr}[i] + \mathsf{Acc}[i-1]$,
3. The last value in Acc matches $\mathsf{Sum}_{\mathsf{Arr}}$.

1. For $X = w^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X)$,
2. For all X except $X = \omega^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X)$
3. For $X = w^{0}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Sum}_{\mathsf{Arr}}$

1. For $X = w^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) = 0$,
2. For all X except $X = \omega^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X) = 0$
3. For $X = w^{0}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Sum}_{\mathsf{Arr}} = 0$

1. $\mathsf{Poly}_{\mathsf{Vanish1}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X)) \cdot \frac{(X^{\kappa}-1)}{(X-\omega^{\kappa-1})} = 0$,
2. $\mathsf{Poly}_{\mathsf{Vanish2}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X)) \cdot (X - \omega^{\kappa-1}) = 0$
3. $\mathsf{Poly}_{\mathsf{Vanish3}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Sum}_{\mathsf{Arr}}) \cdot \frac{(X^{\kappa}-1)}{(X-\omega^{0})} = 0$

1. $Q_1(X) = \frac{\mathsf{Poly}_{\mathsf{Vanish1}}(X)}{X^{\kappa}-1}$
2. $Q_2(X) = \frac{\mathsf{Poly}_{\mathsf{Vanish2}}(X)}{X^{\kappa}-1}$
3. $Q_3(X) = \frac{\mathsf{Poly}_{\mathsf{Vanish3}}(X)}{X^{\kappa}-1}$

1. $\mathsf{Poly}_{\mathsf{Zero}}(X) = \mathsf{Poly}_{\mathsf{Vanish1}}(X) - Q_1(X) \cdot (X^n - 1) = 0$
2. $\mathsf{Poly}_{\mathsf{Zero}}(X) = \mathsf{Poly}_{\mathsf{Vanish2}}(X) - Q_2(X) \cdot (X^n - 1) = 0$
3. $\mathsf{Poly}_{\mathsf{Zero}}(X) = \mathsf{Poly}_{\mathsf{Vanish3}}(X) - Q_3(X) \cdot (X^n - 1) = 0$

1. The first value in Acc matches the first value in Arr,
2. The rest of the values in Acc are of the form $\mathsf{Acc}[i] = \mathsf{Arr}[i] + \mathsf{Acc}[i-1]$,
3. The last value in Acc matches $\mathsf{Sum}_{\mathsf{Arr}}$.

1. For $X = w^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X)$,
2. For all X except $X = \omega^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X)$
3. For $X = w^0$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Sum}_{\mathsf{Arr}}$

1. For $X = w^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) = 0$,
2. For all X except $X = \omega^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X) = 0$
3. For $X = w^0$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Sum}_{\mathsf{Arr}} = 0$

1. $\mathsf{Poly}_{\mathsf{Vanish1}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X)) \cdot \frac{(X^\kappa - 1)}{(X - \omega^{\kappa-1})} = 0$,
2. $\mathsf{Poly}_{\mathsf{Vanish2}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X)) \cdot (X - \omega^{\kappa-1}) = 0$
3. $\mathsf{Poly}_{\mathsf{Vanish3}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Sum}_{\mathsf{Arr}}) \cdot \frac{(X^\kappa - 1)}{(X - \omega^0)} = 0$

1. $Q_1(X) = \frac{\mathsf{Poly}_{\mathsf{Vanish1}}(X)}{X^\kappa - 1}$
2. $Q_2(X) = \frac{\mathsf{Poly}_{\mathsf{Vanish2}}(X)}{X^\kappa - 1}$
3. $Q_3(X) = \frac{\mathsf{Poly}_{\mathsf{Vanish3}}(X)}{X^\kappa - 1}$

1. $\mathsf{Poly}_{\mathsf{Zero}}(X) = \mathsf{Poly}_{\mathsf{Vanish1}}(X) - Q_1(X) \cdot (X^n - 1) = 0$
2. $\mathsf{Poly}_{\mathsf{Zero}}(X) = \mathsf{Poly}_{\mathsf{Vanish2}}(X) - Q_2(X) \cdot (X^n - 1) = 0$
3. $\mathsf{Poly}_{\mathsf{Zero}}(X) = \mathsf{Poly}_{\mathsf{Vanish3}}(X) - Q_3(X) \cdot (X^n - 1) = 0$

- ζ
- $\mathsf{Poly}_{\mathsf{Arr}}(\zeta) = \mathsf{KZG.Open}(K_{\mathsf{Arr}}, \zeta)$
- $\mathsf{Poly}_{\mathsf{Arr}}(\zeta\omega) = \mathsf{KZG.Open}(K_{\mathsf{Arr}}, \zeta\omega)$
- $\mathsf{Poly}_{\mathsf{Acc}}(\zeta) = \mathsf{KZG.Open}(K_{\mathsf{Acc}}, \zeta)$
- $Q_1(\zeta) = \mathsf{KZG.Open}(K_{Q_1}, \zeta)$
- $Q_2(\zeta) = \mathsf{KZG.Open}(K_{Q_2}, \zeta)$
- $Q_3(\zeta) = \mathsf{KZG.Open}(K_{Q_3}, \zeta)$

1. The first value in Acc matches the first value in Arr,
2. The rest of the values in Acc are of the form $\mathsf{Acc}[i] = \mathsf{Arr}[i] + \mathsf{Acc}[i-1]$,
3. The last value in Acc matches $\mathsf{Sum}_{\mathsf{Arr}}$.

1. For $X = w^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X)$,
2. For all X except $X = \omega^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X)$
3. For $X = w^0$: $\mathsf{Poly}_{\mathsf{Acc}}(X) = \mathsf{Sum}_{\mathsf{Arr}}$

1. For $X = w^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) = 0$,
2. For all X except $X = \omega^{\kappa-1}$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X) = 0$
3. For $X = w^0$: $\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Sum}_{\mathsf{Arr}} = 0$

1. $\mathsf{Poly}_{\mathsf{Vanish1}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X)) \cdot \frac{(X^\kappa - 1)}{(X - \omega^{\kappa-1})} = 0$,
2. $\mathsf{Poly}_{\mathsf{Vanish2}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Poly}_{\mathsf{Arr}}(X) + \mathsf{Poly}_{\mathsf{Acc}}(\omega \cdot X)) \cdot (X - \omega^{\kappa-1}) = 0$
3. $\mathsf{Poly}_{\mathsf{Vanish3}}(X) = (\mathsf{Poly}_{\mathsf{Acc}}(X) - \mathsf{Sum}_{\mathsf{Arr}}) \cdot \frac{(X^\kappa - 1)}{(X - \omega^0)} = 0$

1. $Q_1(X) = \frac{\mathsf{Poly}_{\mathsf{Vanish1}}(X)}{X^\kappa - 1}$
2. $Q_2(X) = \frac{\mathsf{Poly}_{\mathsf{Vanish2}}(X)}{X^\kappa - 1}$
3. $Q_3(X) = \frac{\mathsf{Poly}_{\mathsf{Vanish3}}(X)}{X^\kappa - 1}$

1. $\mathsf{Poly}_{\mathsf{Zero}}(X) = \mathsf{Poly}_{\mathsf{Vanish1}}(X) - Q_1(X) \cdot (X^n - 1) = 0$
2. $\mathsf{Poly}_{\mathsf{Zero}}(X) = \mathsf{Poly}_{\mathsf{Vanish2}}(X) - Q_2(X) \cdot (X^n - 1) = 0$
3. $\mathsf{Poly}_{\mathsf{Zero}}(X) = \mathsf{Poly}_{\mathsf{Vanish3}}(X) - Q_3(X) \cdot (X^n - 1) = 0$

- ζ
- $\mathsf{Poly}_{\mathsf{Arr}}(\zeta) = \mathsf{KZG.Open}(K_{\mathsf{Arr}}, \zeta)$
- $\mathsf{Poly}_{\mathsf{Arr}}(\zeta\omega) = \mathsf{KZG.Open}(K_{\mathsf{Arr}}, \zeta\omega)$
- $\mathsf{Poly}_{\mathsf{Acc}}(\zeta) = \mathsf{KZG.Open}(K_{\mathsf{Acc}}, \zeta)$
- $Q_1(\zeta) = \mathsf{KZG.Open}(K_{Q_1}, \zeta)$
- $Q_2(\zeta) = \mathsf{KZG.Open}(K_{Q_2}, \zeta)$
- $Q_3(\zeta) = \mathsf{KZG.Open}(K_{Q_3}, \zeta)$

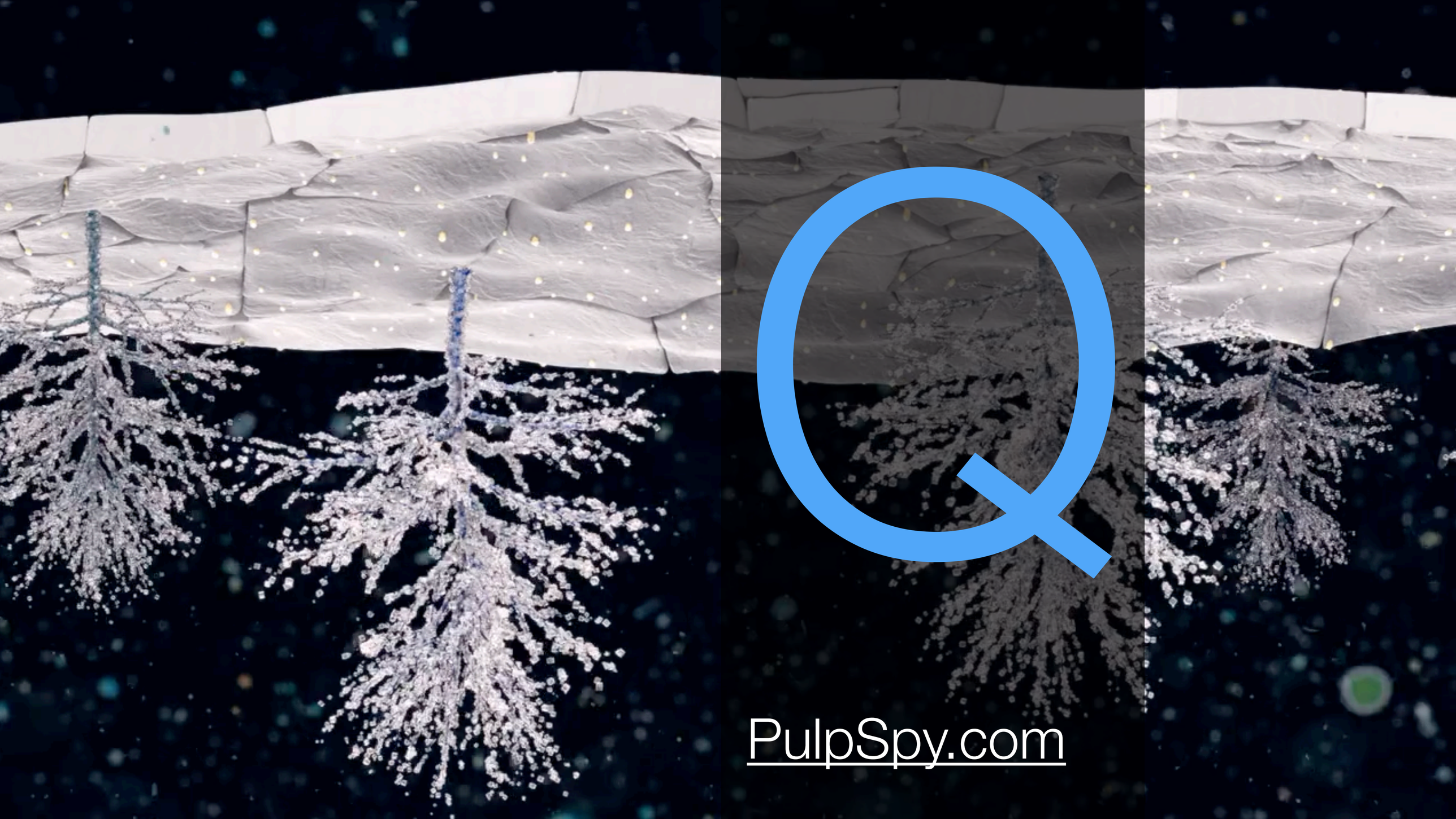
Overwhelmingly will be 0 at λ (Schwartz-Zippel lemma)

Methodology & Results

- Find use-cases based on column-style data (accounts, keys, ballots, orders) where (i) there is a “prover” that knows everything and (ii) the prover’s integrity is in question
- We see if the gadget book is sufficient to prove the computation we need to do (if not, we can use general circuits but we avoid them for efficiency)
- We develop new gadgets (e.g., min and max for orders) and we deal with technical issues (mapping between elliptic curves, packing things into domain sizes, etc.)
- We code (Arkworks built on Rust) the protocols and benchmark them
- Prover time is variable (~10s of hours) but proofs are small (KB) and verification is fast (ms) or can be outsourced to a smart contract on Ethereum for $30
- Papers have lots of details, charts, etc.

Future Work

- Benchmarks for application-specific arithmetization vs general purpose SNARKs
- Gadget book for multi-linear polynomials (HONK vs PLONK)
- More use-cases!


Q
PulpSpy.com