Modern zk-SNARKs

Mary Maller
Ethereum Foundation, London

Talk Outline

- Motivate why NIST should be interested in zk-SNARKs
- Outline key challenges with moving SNARKs from theory to practice.
- Discuss each key challenge individually.
- Plug one of my recent papers.

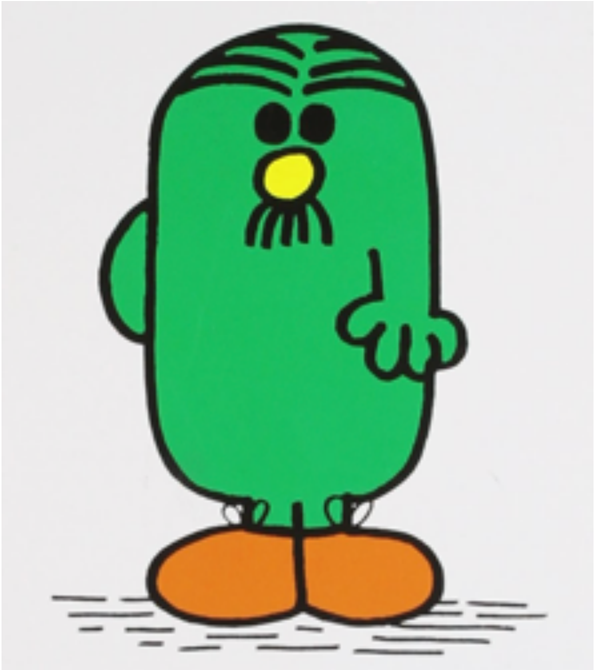
Why should NIST be interested in zk-SNARKs

Why do we care about SNARKs?

- A SNARK allows a user to prove that they have run a computation correctly.
- The verifier can check the output very quickly

$$f(x) = y$$
$$\pi = \text{SNARK}(x, y)$$

$$\xrightarrow{(x, y, \pi)}$$



y is correct

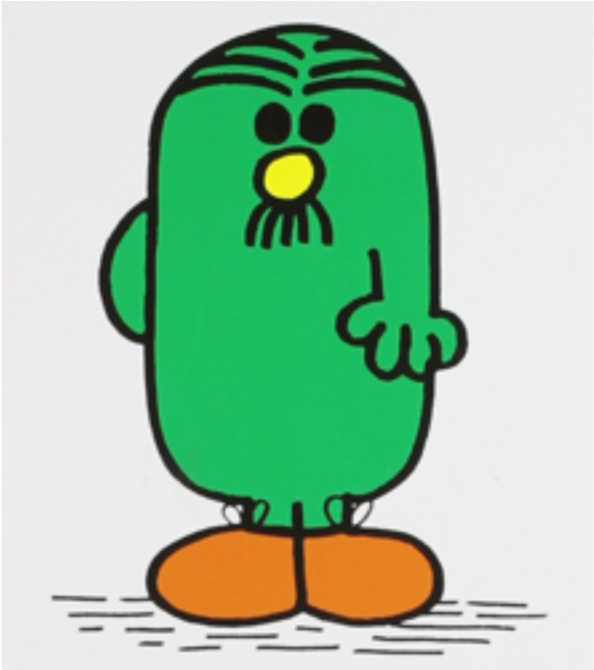
Outsourcing computations

Why do we care about SNARKs?

- A SNARK allows a user to prove that they have run a computation correctly.
- The verifier can check the output very quickly

$$f(x) = y$$
$$\pi = \text{SNARK}(x, y)$$

$$\xrightarrow{(x, y, \pi)}$$

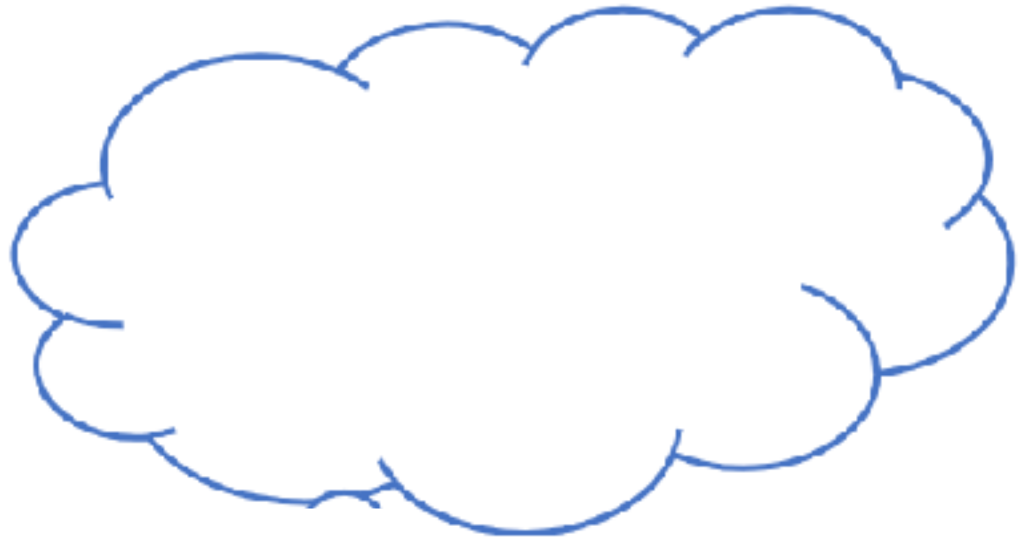


y is correct

Outsourcing computations

Why do we care about SNARKs?

- Can prove that a cloud computation has been carried out correctly.



Microsoft | Research Our research Programs & events Blogs & podcasts About

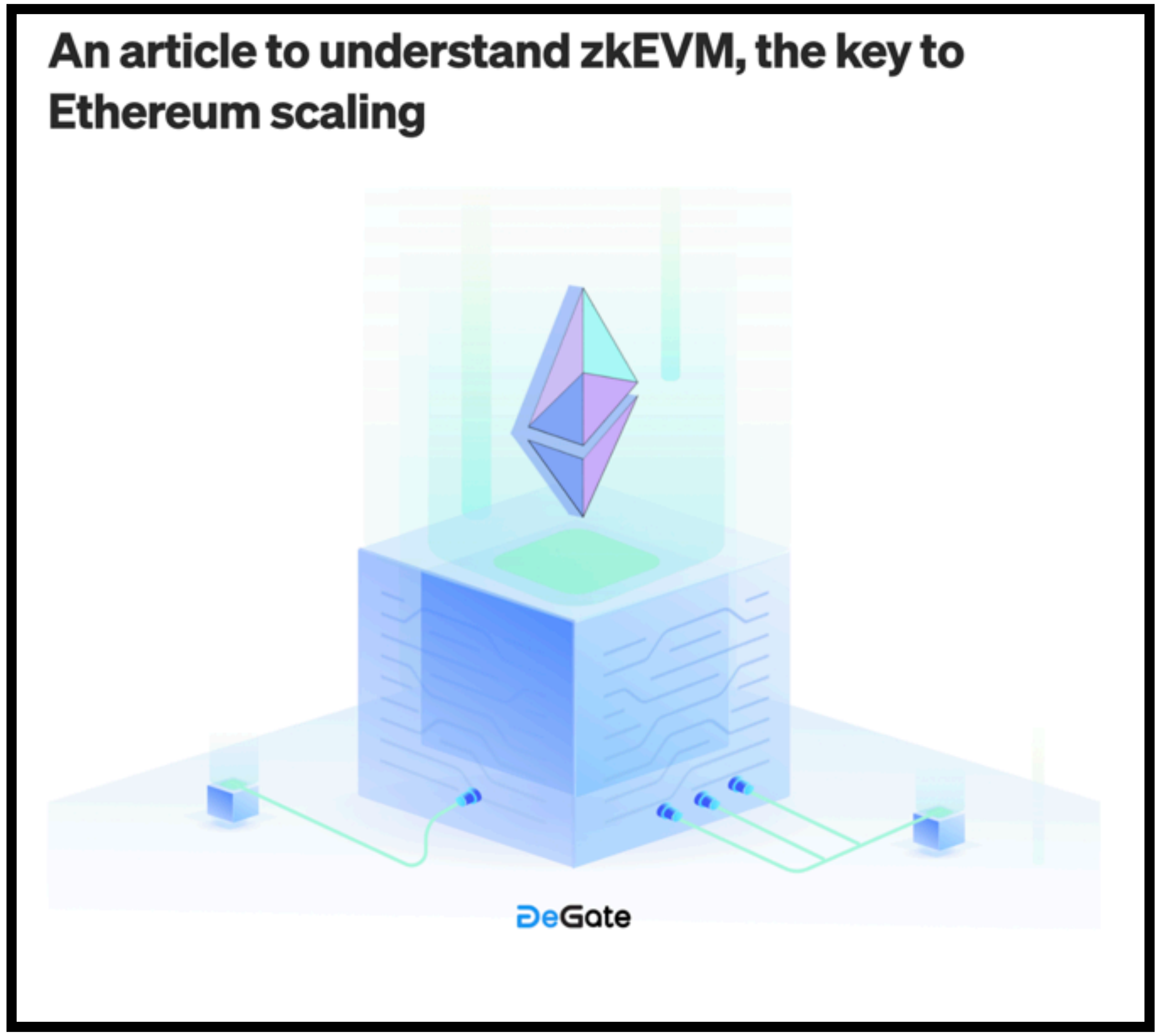
Pinocchio: Nearly Practical Verifiable Computation

Bryan Parno, Jon Howell, Craig Gentry, Mariana Raykova
Proceedings of the IEEE Symposium on Security and Privacy | May 2013
Published by IEEE
Best Paper Award

Outsourcing computations

Why do we care about SNARKs?

- Can reduce the size (and thus improve scalability) of blockchains.



Why do we care about zk-SNARKs

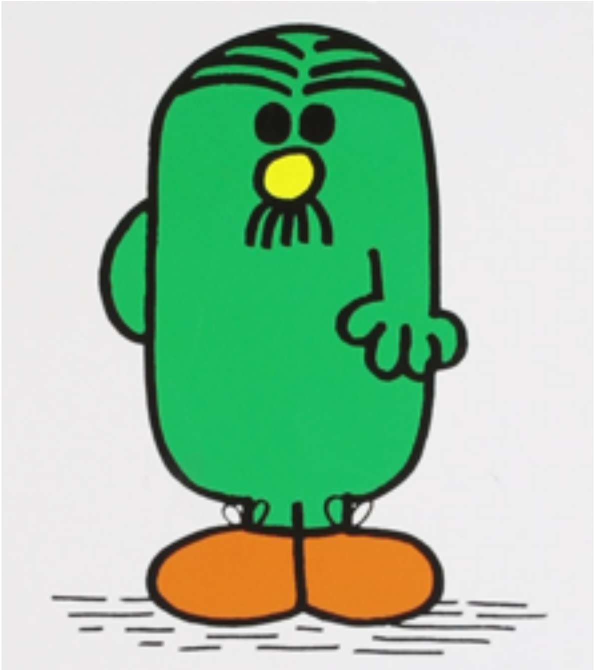
- A SNARK allows a user to prove that they have run a computation correctly.
- The verifier can check the output very quickly
- A zk-SNARK additionally reveals no information about the input of the computation.

exists x
such that y is
correct

$\pi = \mathsf{SNARK}(x, y)$

(y, π)

$\longrightarrow$



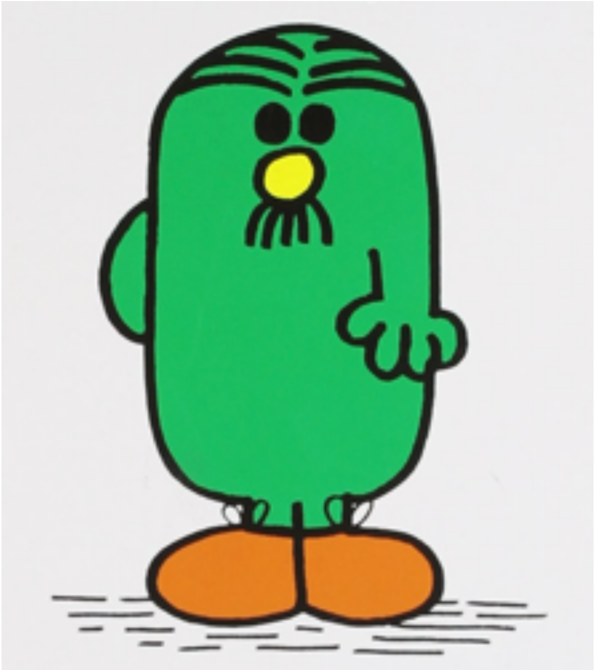
Privacy applications

Why do we care about zk-SNARKs

- A SNARK allows a user to prove that they have run a computation correctly.
- The verifier can check the output very quickly
- A zk-SNARK additionally reveals no information about the input of the computation.

$\pi = \text{SNARK}(x, y)$

$$\xrightarrow{\quad (y, \pi) \quad}$$



exists x such that y is correct

Privacy applications

Why do we care about zk-SNARKs

- Prove existence of code vulnerabilities without putting users at risk.
- Run an actively secure MPC.
- Build anonymous credentials.
- Demonstrate membership in a group.
- Many more

Researchers Demonstrate Potential for Zero-Knowledge Proofs in Vulnerability Disclosure

Research teams led by Galois, Trail of Bits develop capability to mathematically prove exploitability of vulnerable software without revealing critical information

OUTREACH@DARPA.MIL
4/22/2021

Introducing Multi-Party ECDSA library

Omer Shlomovits
November 4, 2019

Tweet Facebook

Since we started KZen, we invested in the development of a cryptographic stack that would enable us to build a new generation of keyless crypto wallets with simpler and stronger security eliminating that way typical single point of failures and tedious setup and recovery schemes.

Meta says its Anonymous Credentials Service (ACS) will help reduce data collection activities

Image Credit: Sean Gladwell // Getty Images

What are the key challenges for moving SNARKs from theory to practice?

The Good

- Proof sizes are small.
- Verification time is fast.
- There are no trusted third parties.
- Storage requirements are reasonable.
- SNARKs are applicable to any computation.

The Challenges

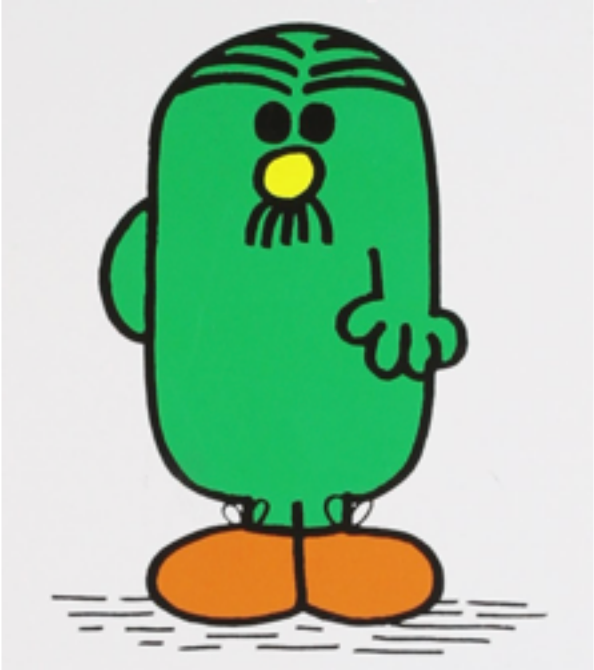
- Implementing SNARKs securely is really *really* hard.
- Prover time is slow.

The Good

- Proof sizes are small.
- Verification time is fast.

Proof sizes starting from 200 bytes

Verifier time starting from10s of microseconds



The Good

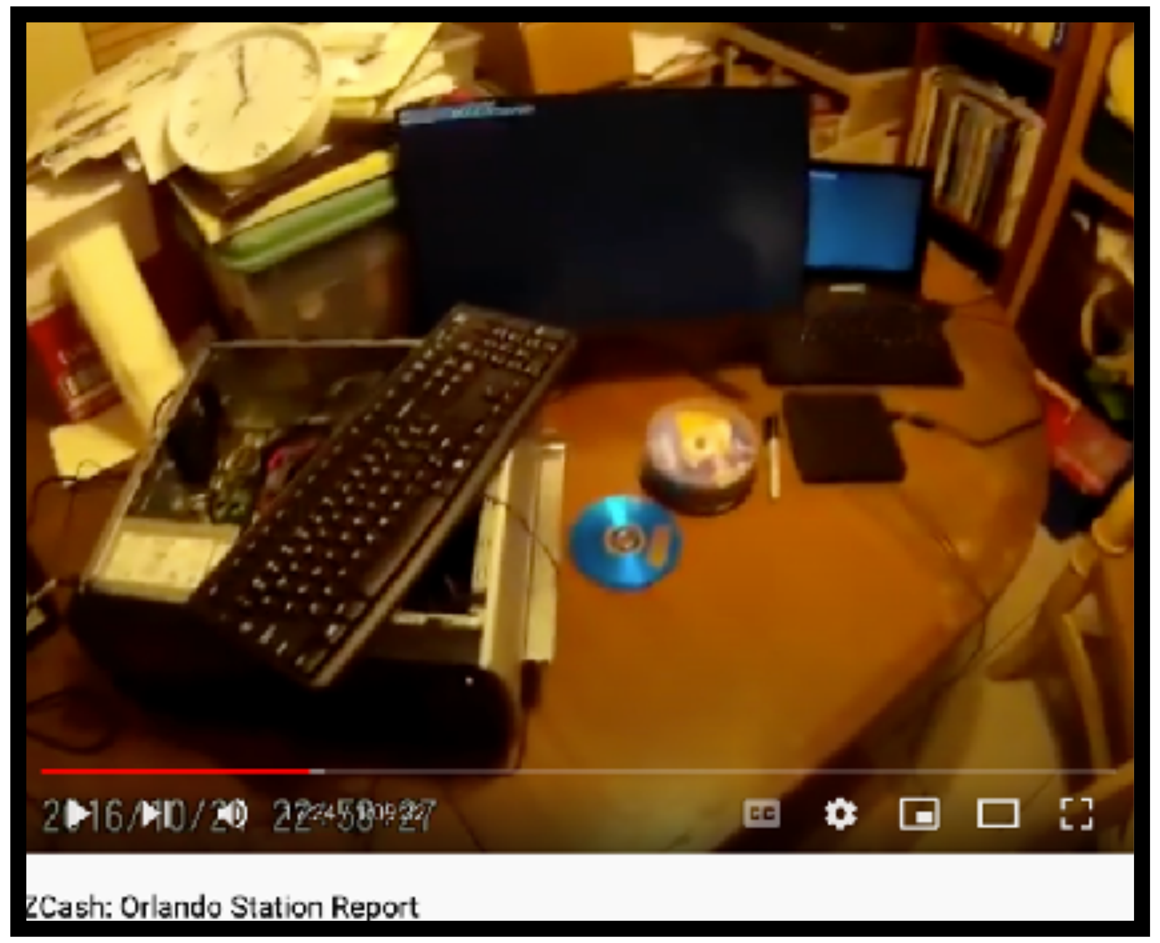
- Proof sizes are small.
- Verification time is fast.
- There are no trusted third parties

- A trusted third party is someone trusted to not cheat.
- Some SNARKs do require a one time “trusted setup” and others do not.
- Some trusted setups are better than others.

The Good

- Proof sizes are small.
- Verification time is fast.
- There are no trusted third parties

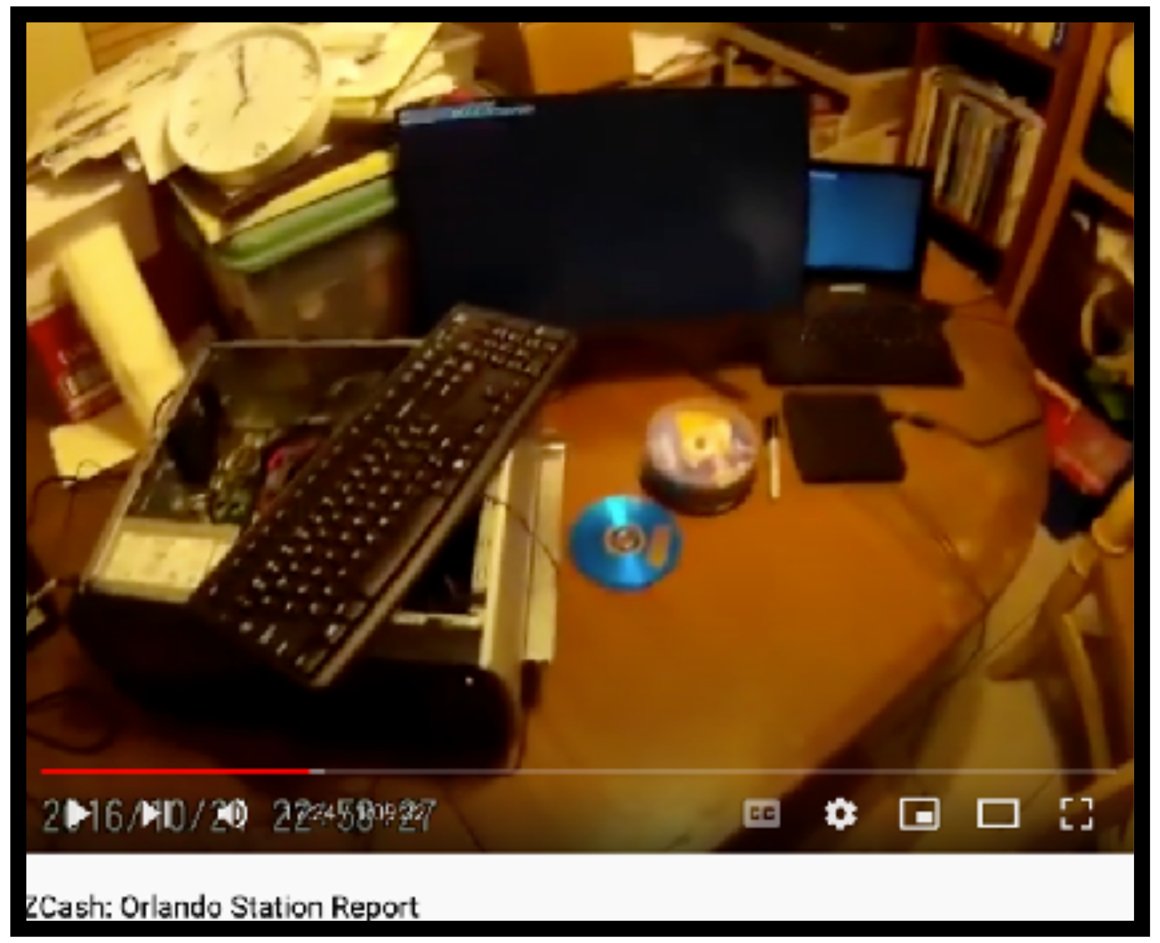
- A trusted third party is someone trusted to not cheat.
- Some SNARKs do require a one time "trusted setup" and others do not.
- Some trusted setups are better than others.

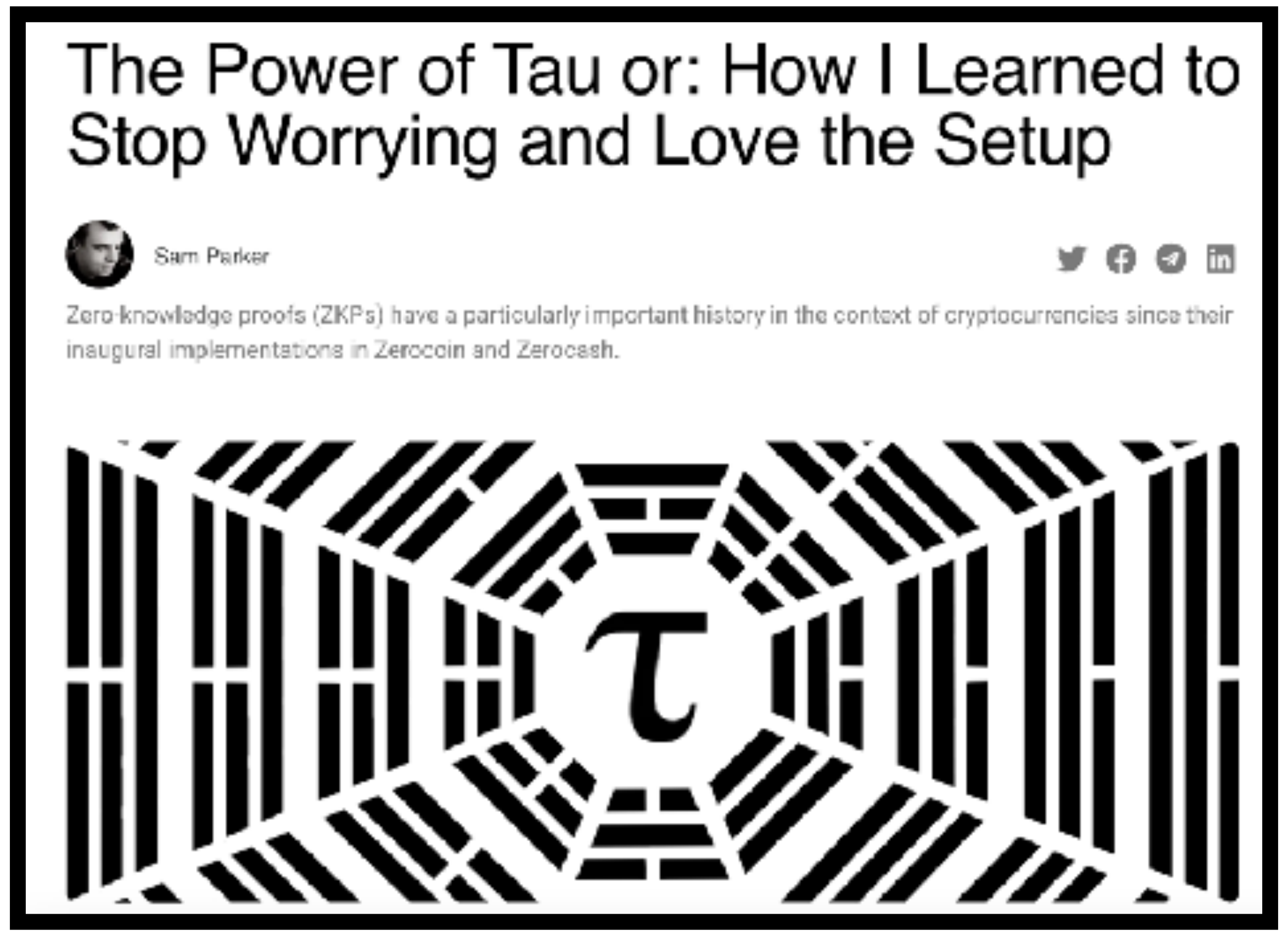


The Good

- Proof sizes are small.
- Verification time is fast.
- There are no trusted third parties

- A trusted third party is someone trusted to not cheat.
- Some SNARKs do require a one time “trusted setup” and others do not.
- Some trusted setups are better than others.




The Power of Tau or: How I Learned to Stop Worrying and Love the Setup

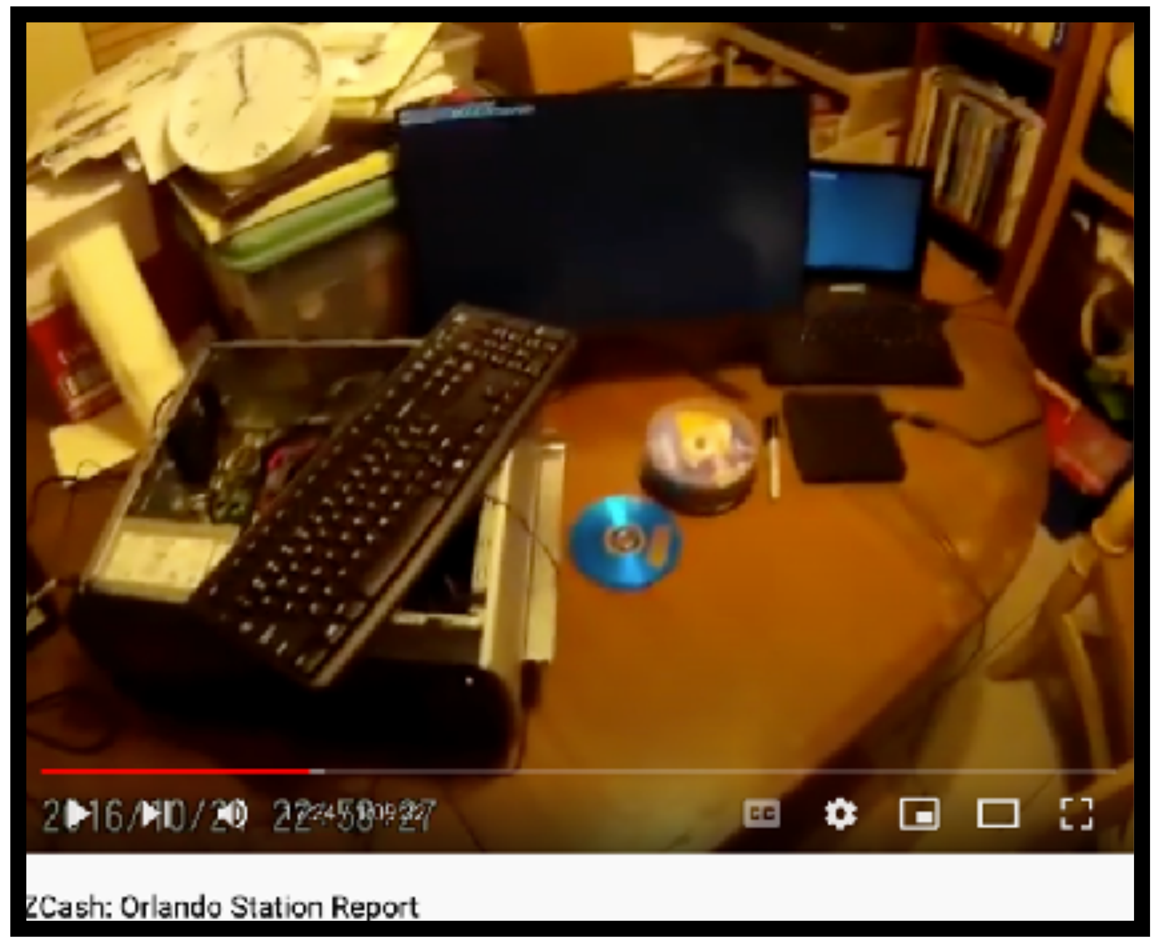
Sam Parker

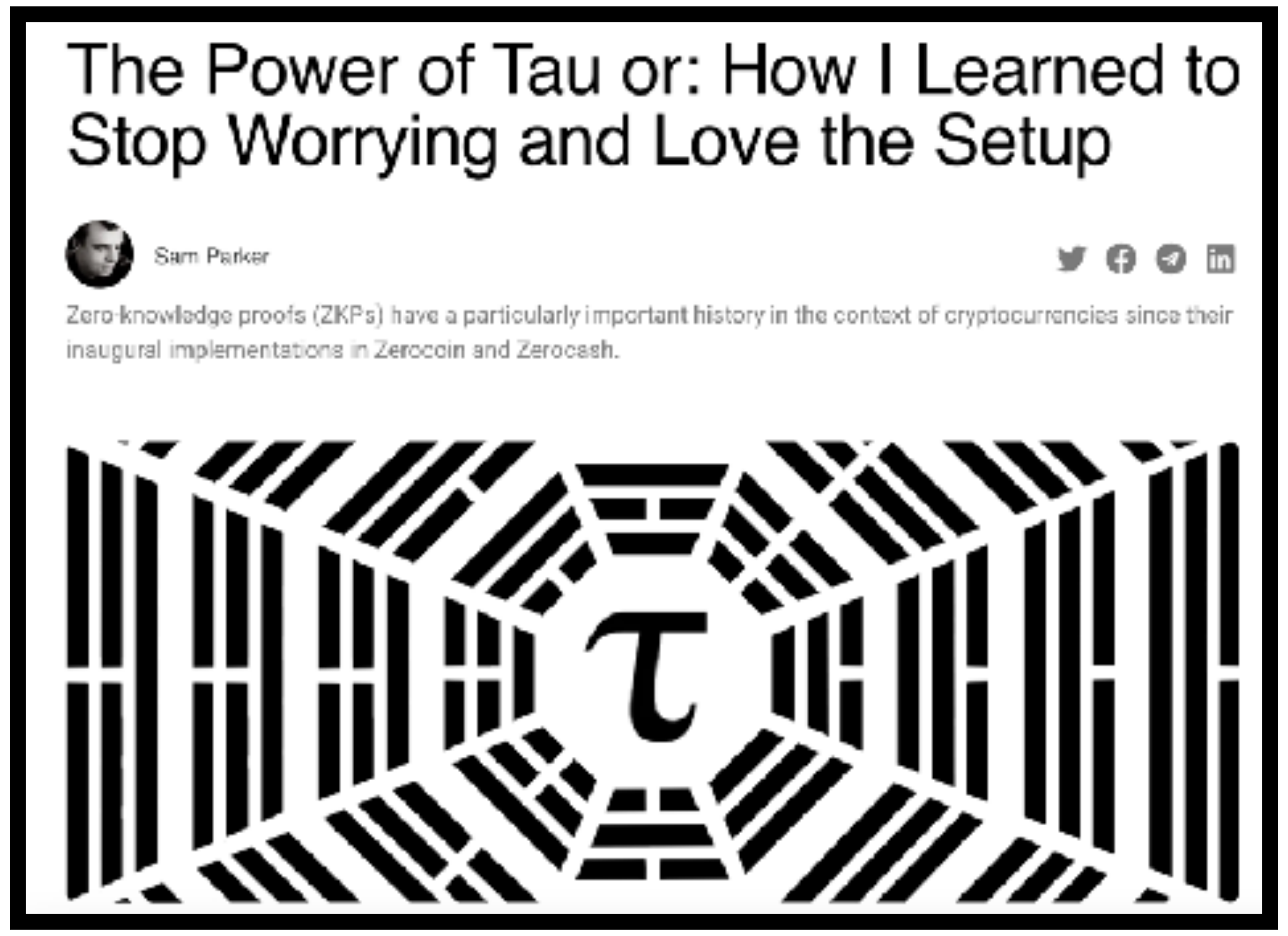
Zero-knowledge proofs (ZKPs) have a particularly important history in the context of cryptocurrencies since their inaugural implementations in Zerocoin and Zerocash.

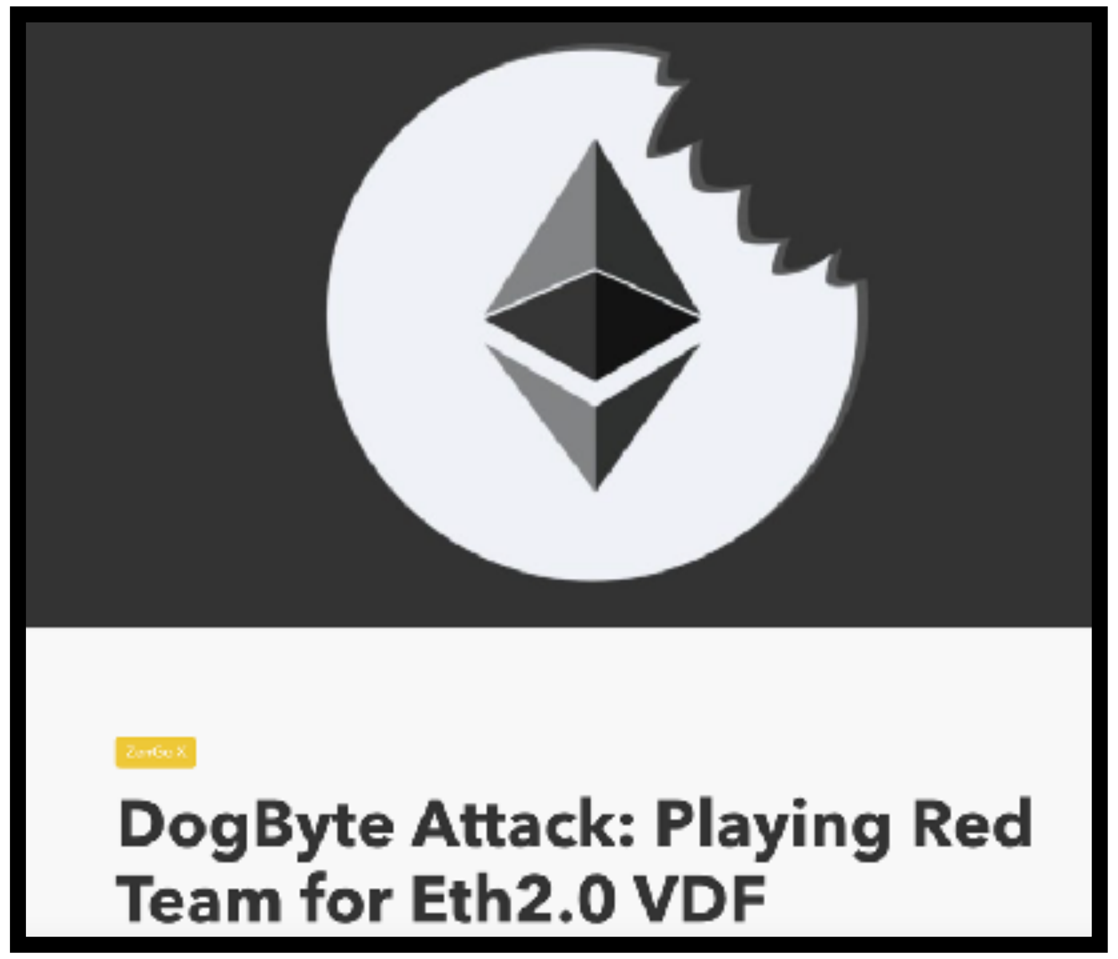
The Good

- Proof sizes are small.
- Verification time is fast.
- There are no trusted third parties

- A trusted third party is someone trusted to not cheat.
- Some SNARKs do require a one time "trusted setup" and others do not.
- Some trusted setups are better than others.







The Good

- Proof sizes are small.
- Verification time is fast.
- There are no trusted third parties
- Storage requirements are reasonable.

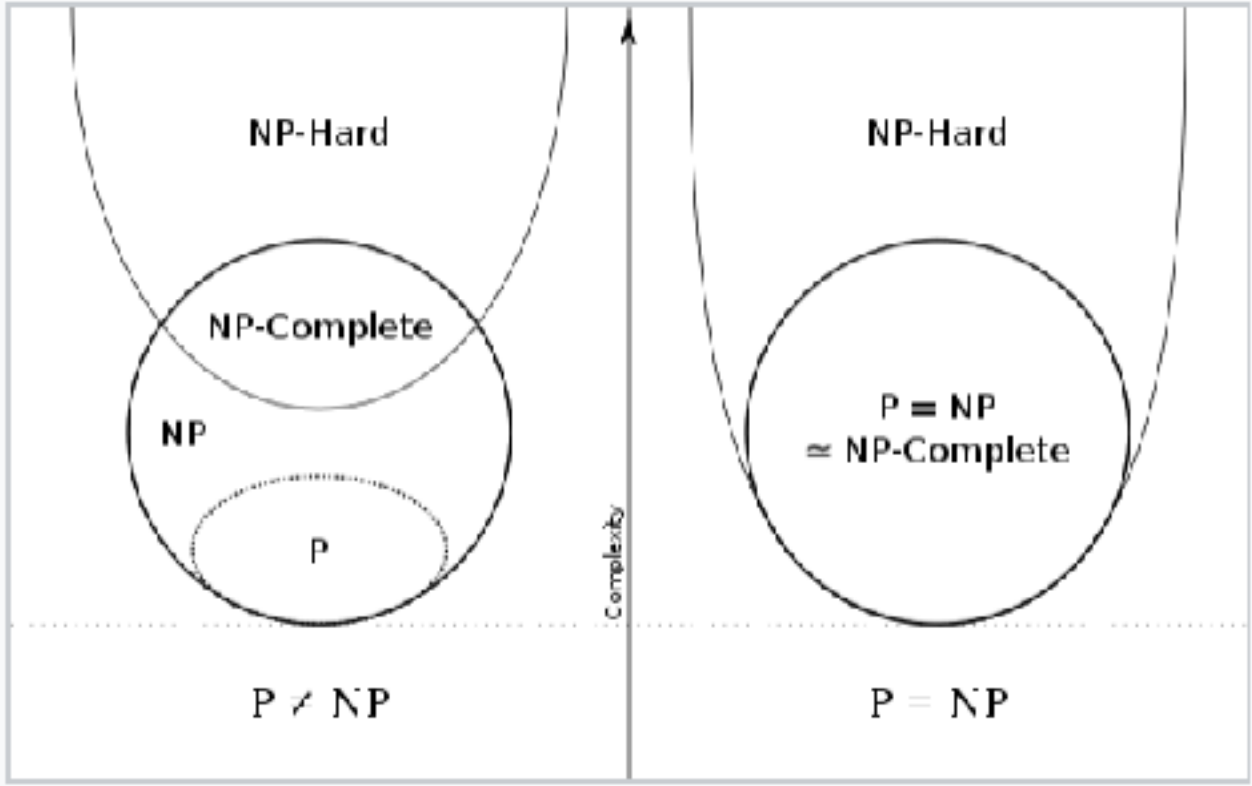
Prover storage can depend on the computation.

Verifier storage starting from 200 bytes.

The Good

- Proof sizes are small.
- Verification time is fast.
- There are no trusted third parties
- Storage requirements are reasonable.
- SNARKs are applicable to any computation.

Theoretically can cover any computation in NP



Euler diagram for P, NP, NP-complete, and NP-hard set of problems (excluding the empty language and its complement, which belong to P but are not NP-complete)

https://en.wikipedia.org/wiki/P_versus_NP_problem

The Challenges

- Implementing SNARKs securely is really *really* hard.

Background

On March 1, 2018, Ariel Gabizon, a cryptographer employed by the Zcash Company at the time, discovered a subtle cryptographic flaw in the [BCTV14] paper that describes the zk-SNARK construction used in the original launch of Zcash. The flaw allows an attacker to create counterfeit shielded value in any system that depends on parameters which are generated as described by the paper.

This vulnerability is so subtle that it evaded years of analysis by expert cryptographers focused on zero-knowledge proving systems and zk-SNARKs. In an analysis [Parno15] in 2015, Bryan Parno from Microsoft Research discovered a different mistake

The Challenges

- Implementing SNARKs securely is really *really* hard.
- Prover time is slow.

Prover time comparison of GKR+Groth16 vs. Groth16 for proving

zk-s[nt]arks zk-roll-up

We implemented the following circuit 21 to measure Gnark's performance.

Op	Runtime (sec)
Groth16 Prover - 2^15 hashes	20.2
Groth16 Prover - 2^17 hashes	78.2
Extrapolation to 2^22 hashes	2587.0
Extrapolation to 2^23 hashes	5006.3

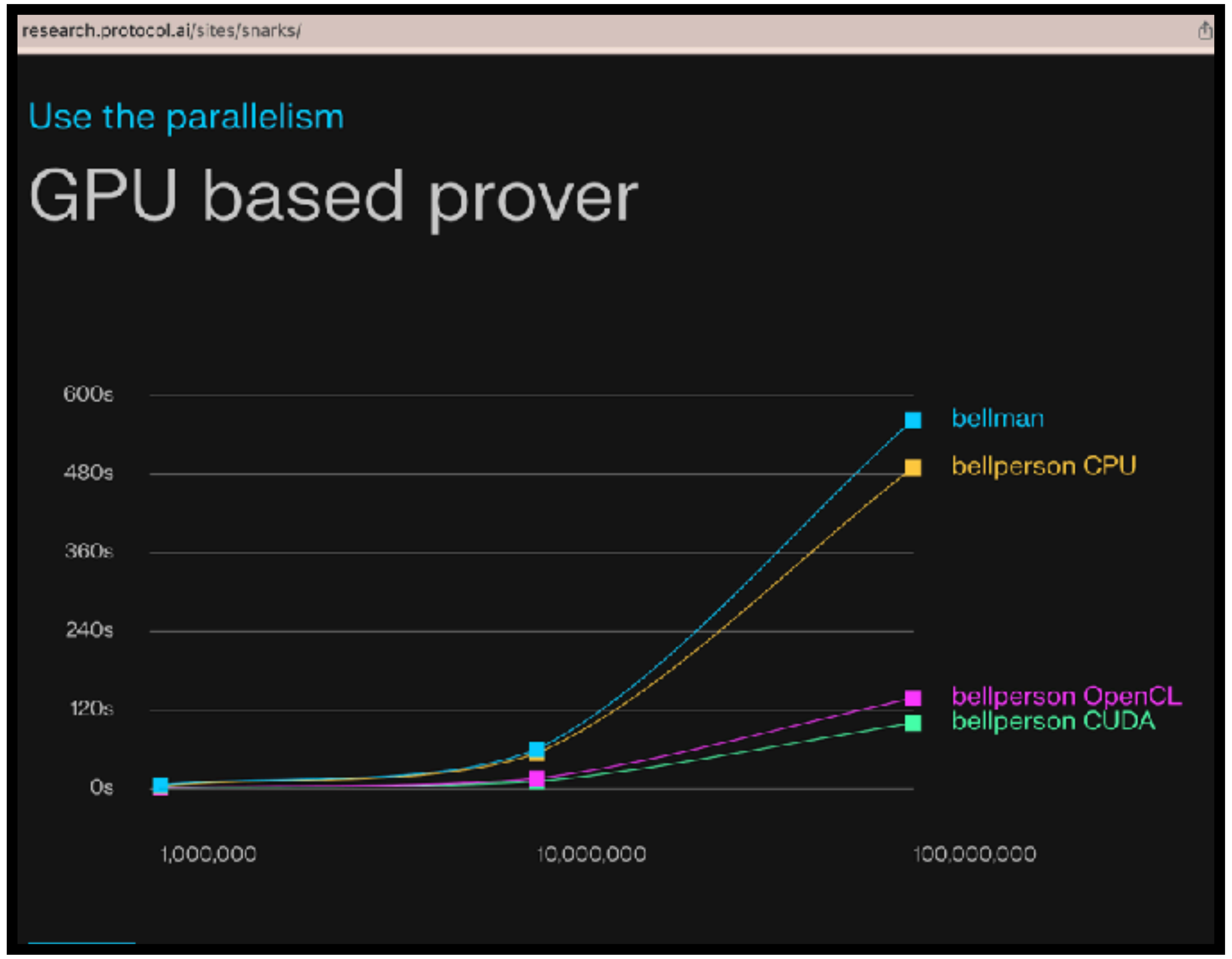
We implemented 11 the GKR prover and the Groth16 circuit of proof of the proof verification.

With 2^22 (~4M) hashes:

Op	Runtime (sec) for 2^22 hashes	Runtime (sec) for 2^23 hashes
GKR Prover assignment	6.0	8.4
GKR Prover	50.0	95.7
SNARK Assignment	0.3	0.66
SNARK Prover	48.2	76.5
Total	104.6	181.2
Baseline	2587.0	5006.3

Which is a 27-fold improvement compared to the baseline for 8M hashes.

Orders of magnitude slower than proving the computation



Key Challenge: Implementing SNARKs is hard.

Implementing SNARKs securely is hard

- Moving complicated zero-knowledge protocols from theory to practice is hard.
- Suddenly it really *really* matters that the security proof is correct.
- As a community we are still learning the best practices for how to ensure this.

EasyCrypt

Audits

Standards Efforts

Peer Review

Waiting a While

Independent Proofs

Case Study: The Groth16 SNARK

- We now have four different proofs for Groth16.
- Each of these analyses were conducted by hand.

Peer Review	Waiting a While	Independent Proofs	Audits

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

Jens Groth**

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

The Algebraic Group Model and its Applications

Georg Fuchsbauer[1] Eike Kiltz[2] Julian Loss[2]

April 15, 2019

[1] Inria, ENS, CNRS, PSL, France
georg.fuchsbauer@ens.fr
[2] Ruhr University Bochum, Germany
{eike.kiltz,julian.loss}@rub.de

Snarky Ceremonies

Markulf Kohlweiss[1,2], Mary Maller[3], Janno Siim[4], Mikhail Volkhov[2]

[1] IOHK
[2] The University of Edinburgh, UK
{mkohlwei, mikhail.volkhov}@ed.ac.uk
[3] Ethereum Foundation
mary.maller@ethereum.org
[4] University of Tartu, Estonia
janno.siim@ut.ee

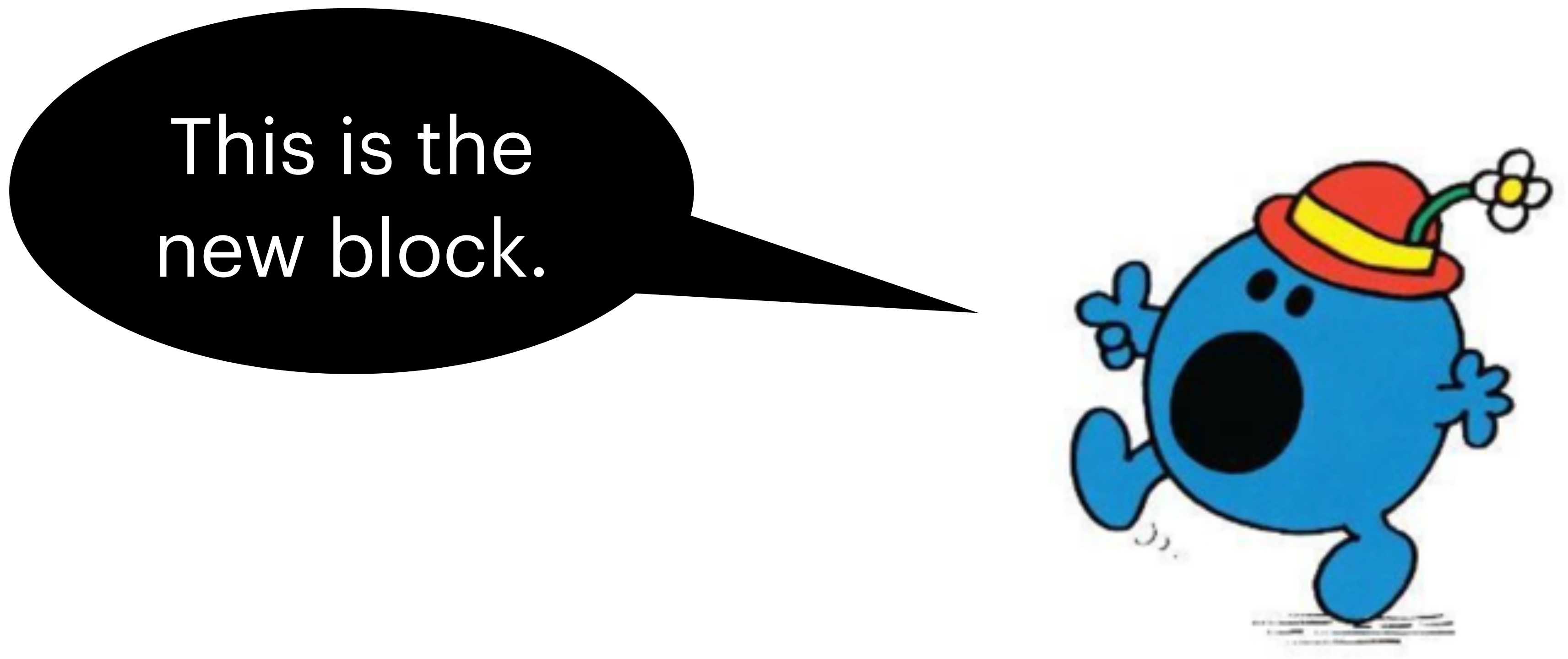
Another Look at Extraction and Randomization of Groth's zk-SNARK

Karim Baghery[1], Markulf Kohlweiss[2,3], Janno Siim[3,4], and Mikhail Volkhov[3]

[1] imec-COSIC, KU Leuven, Belgium
karim.baghery@kuleuven.be
[2] IOHK
[3] The University of Edinburgh, UK
[4] University of Tartu, Estonia
{mkohlwei, jsiim, mikhail.volkhov}@ed.ac.uk

WIP: A Shuffle Argument for ETH 2.0

- In the Ethereum Proof of Stake algorithm, new blocks are proposed by leaders.
- Each time slot has a unique leader who is determined in advance.
- DDOSing a single leader could grind the whole network to a halt.



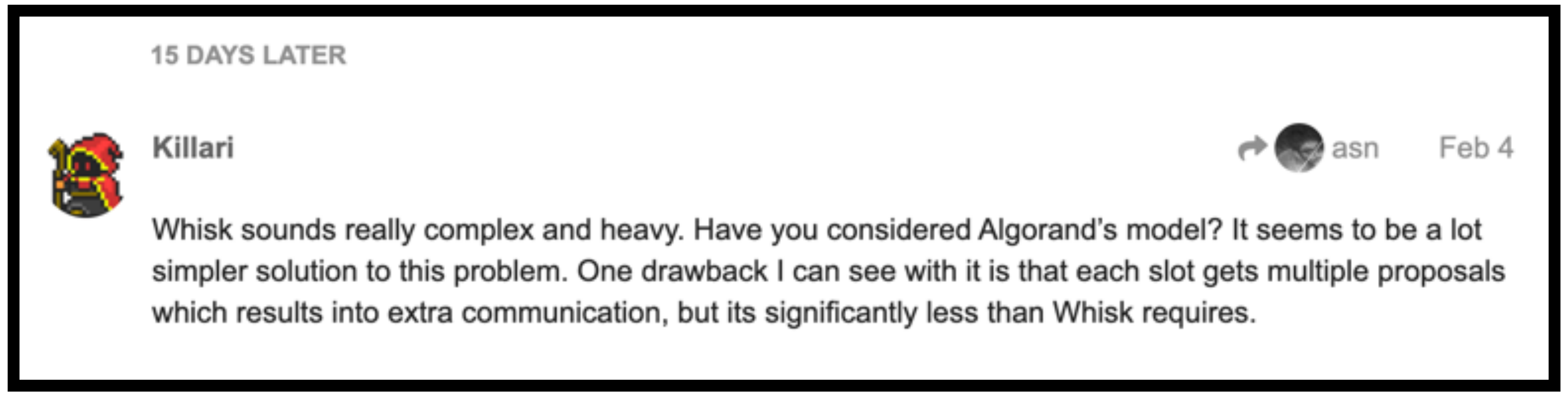
WIP: A Shuffle Argument for ETH 2.0

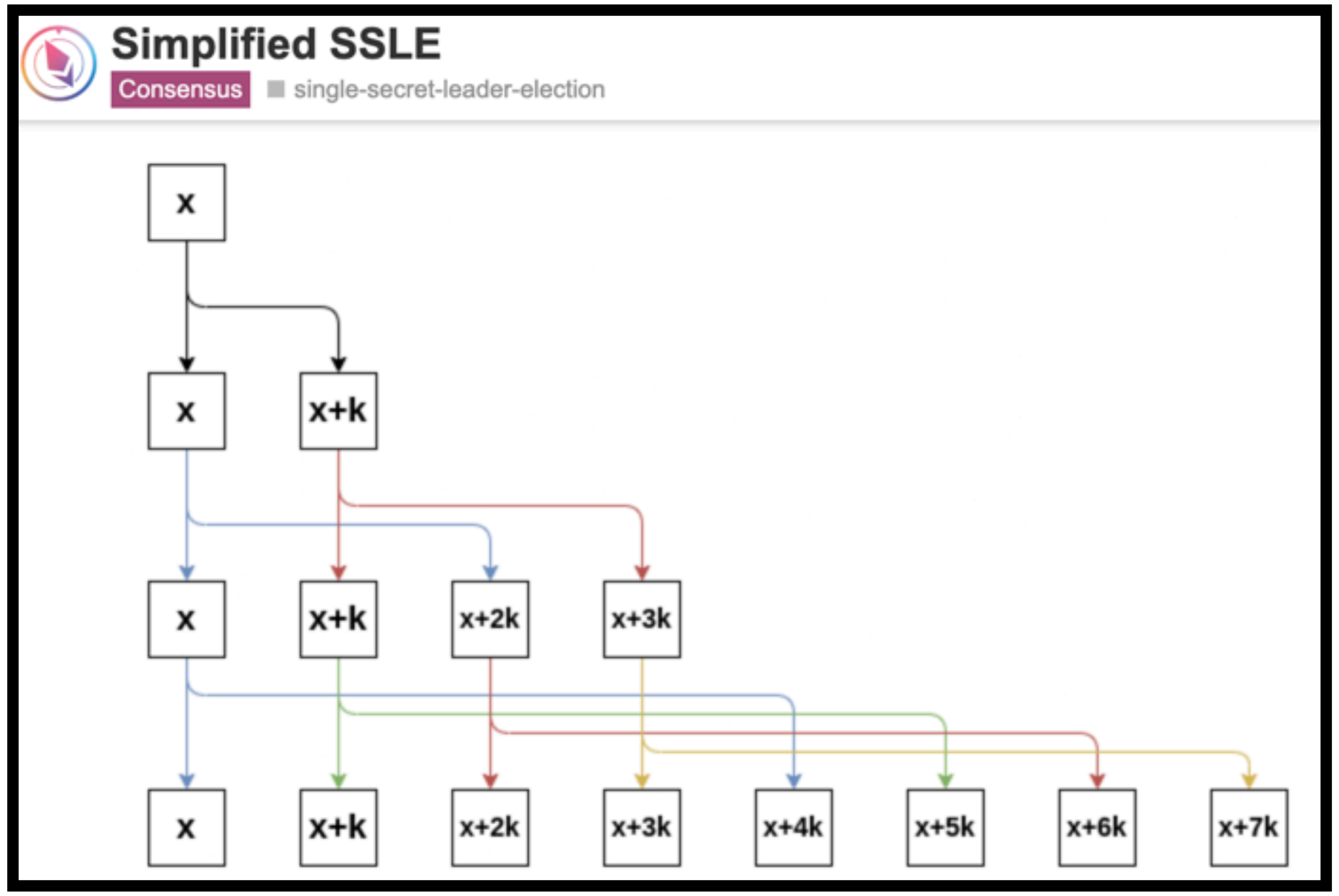
- In the Ethereum Proof of Stake algorithm, new blocks are proposed by leaders.
- Each time slot has a unique leader who is determined in advance.
- DDOSing a single leader could grind the whole network to a halt.
- Solution: hide the leader so that nobody knows who they are in advance (except the leader themselves).
- We plan to implement an adaptation of the Bayer-Groth Shuffle argument.

The Challenges

WIP: A Shuffle Argument for ETH 2.0

- We plan to implement an adaptation of the Bayer-Groth Shuffle argument.
- Currently it scares some implementers due to the complexity.
- BG is one of the simplest ZKPs…





The Challenges

WIP: A Shuffle Argument for ETH 2.0

- We plan to implement an adaptation of the Bayer-Groth Shuffle argument.
- Currently it scares some implementers due to the complexity.
- BG is one of the simplest ZKPs, and seems more complex than it is due to poor documentation.

Short Term Plan

Documentation

Formal “research styled” PDF

Technical Specification

Reference Implementation

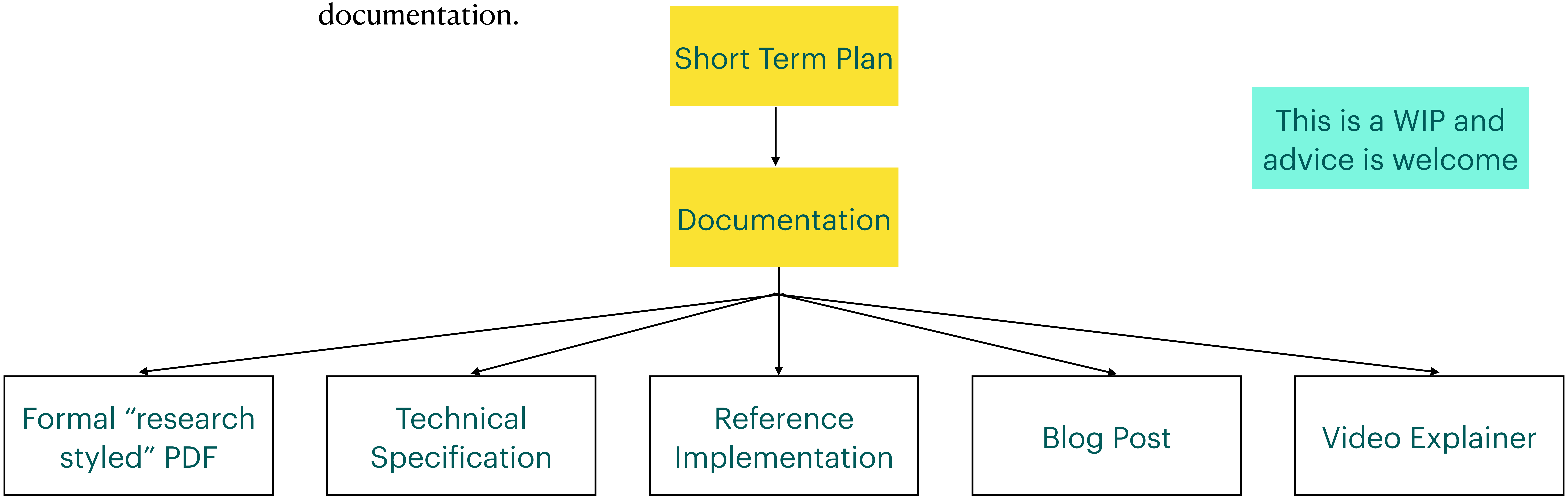
Blog Post

Video Explainer

The Challenges

WIP: A Shuffle Argument for ETH 2.0

- We plan to implement an adaptation of the Bayer-Groth Shuffle argument.
- Currently it scares some implementers due to the complexity.
- BG is one of the simplest ZKPs, and seems more complex than it is due to poor documentation.



This is a WIP and advice is welcome

Standards Efforts

Case Study: Zero-Knowledge Standardisation Effort

ZKPROOF

Active WGs

#	WG Name (Telegram Group)	Goal	Repo	Documents
1	Commit-and-Prove ZKP Systems & Extensions (TG)	Standardize how to modularly create verifiable commitments and ciphertexts with any ZK system	repo	W2, W3, W4, Notes, Slides, Charter
2	Σ-protocols (TG)	Standardize ZK interactive sigma protocols	repo	W4
3	DAPOL (TG)	v	—	W3, CCS21, ACNS21, Notes
4	zkInterface (TG)	Standardize interoperability between frontends and backends in ZK systems	repo	W2, W3, Notes
5	Snark-Friendly Primitives (TG)	Standardize the use of specific cryptographic primitives inside of the circuit of a ZK system	repo	Charter, Notes

- ZKProof is an effort to produce standards for ZKPs to ease their adoption.
- We have run a total of 4 community workshops to gather ideas about what to standardise.
- We have 5 active working groups that are focussing on specific topics.
- We have a community reference document designed to be an entry level explainer for ZKPs.
- We have additional online resources.

zkproof.org

Key Challenge: Prover time is high.

Prover Time is High

- SNARK provers depend (quasi)-linearly on the computation being proven and the constants are large.

- Computation dominated by group multiplications and Fast Fourier Transforms.

- The faster the prover, the more we can prove.

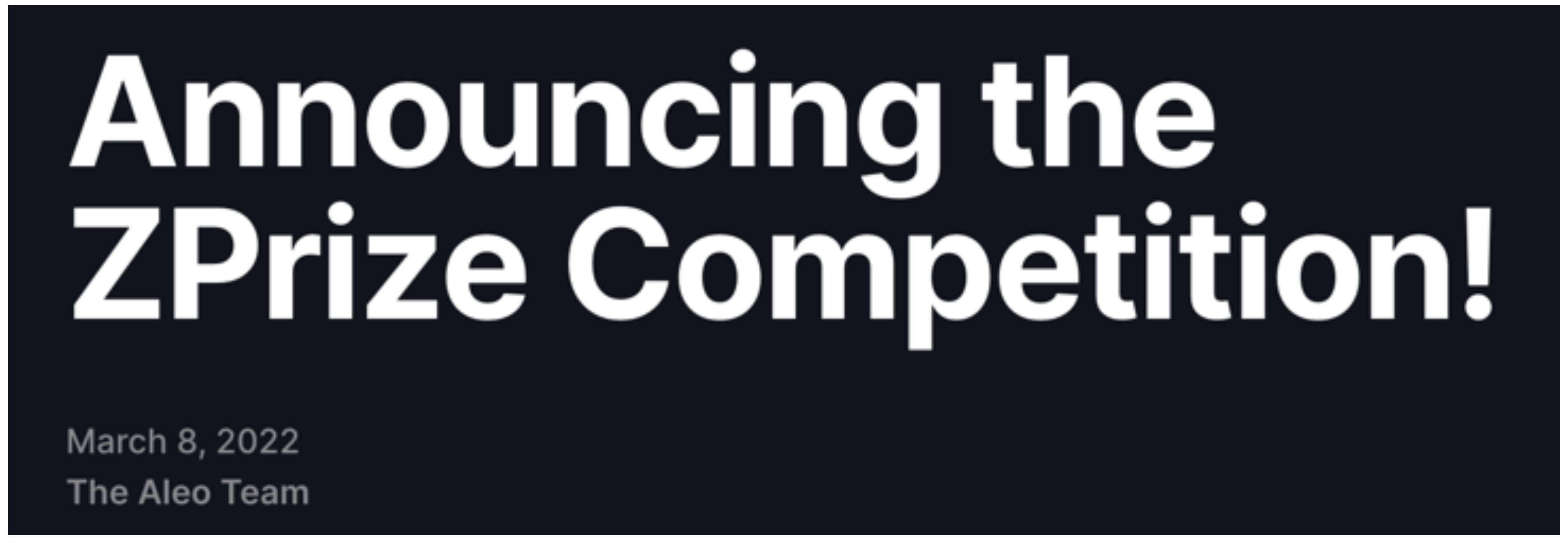
Specialised Hardware

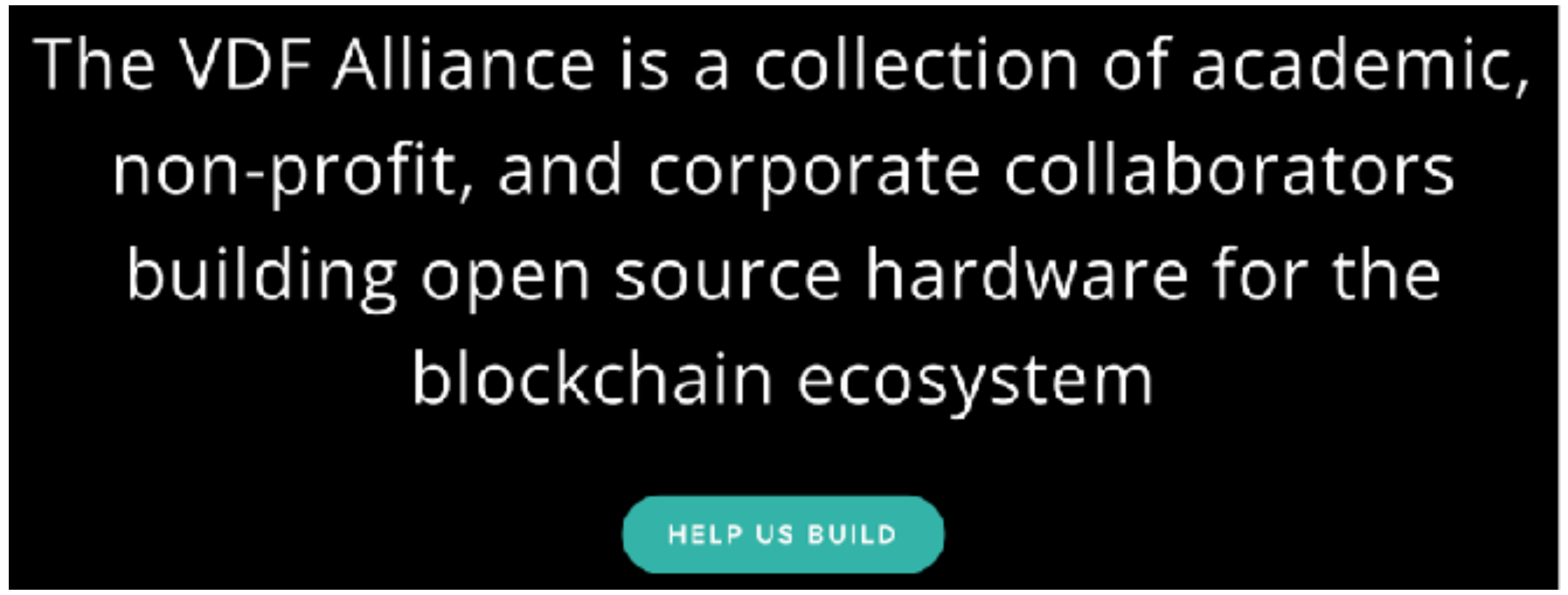
Smaller Computations

Recursion

Specialised Hardware

- ZKPrize an ongoing competition to produce better hardware for SNARKs.
- A related project is building specialised hardware for verifiable delay functions.
- ASICs can have a huge impact on SNARK proving time.





Recursion

- Recent work has looked into building SNARKs of SNARKs.

- This can improve prover time whenever smaller computations are repeated frequently: one key use case is blocklists.

Paper 2021/1577

SNARKBlock: Federated Anonymous Blocklisting from Hidden Common Input Aggregate Proofs

Michael Rosenberg, Mary Maller, and Ian Miers

Recursion

- Recent work has looked into building SNARKs of SNARKs.
- This can improve prover time whenever smaller computations are repeated frequently: one key use case is blocklists.
- We have known for a while that recursion is promising.

Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency

Paul Valiant
pvaliant@mit.edu, Massachusetts Institute of Technology

Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data

Nir Bitansky*
nirbitan@tau.ac.il
Tel Aviv University

Ran Canetti*
canetti@tau.ac.il
Boston University and Tel Aviv University

Alessandro Chiesa
alexch@csail.mit.edu
MIT

Eran Tromer†
tromer@tau.ac.il
Tel Aviv University

December 28, 2012

On cycles of pairing-friendly elliptic curves

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Lynn Chua
chualynn@berkeley.edu
UC Berkeley

Matthew Weidner
malw2@cam.ac.uk
Cambridge

November 5, 2018

Recursion

- Recent work has looked into building SNARKs of SNARKs.
- This can improve prover time whenever smaller computations are repeated frequently: one key use case is blocklists.
- There are lots of exciting developments in this area.

Paper 2020/499

Proof-Carrying Data from Accumulation Schemes

Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner

Paper 2020/1618

Proof-Carrying Data without Succinct Arguments

Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner

Proofs for Inner Pairing Products and Applications

Benedikt Bünz, benedikt@cs.stanford.edu, Stanford University
Mary Maller, mary.maller@ethereum.org, Ethereum Foundation
Pratyush Mishra, pratyush@berkeley.edu, UC Berkeley
Nirvan Tyagi, tyagi@cs.cornell.edu, Cornell University
Psi Vesely, psi@berkeley.edu, UC Berkeley

Paper 2021/529

SnarkPack: Practical SNARK Aggregation

Nicolas Gailly, Mary Maller, and Anca Nitulescu

Nova: Recursive Zero-Knowledge Arguments from Folding Schemes

Abhiram Kothapalli[*,†] Srinath Setty[*] Ioanna Tzialla[‡]

[*]Microsoft Research [†]Carnegie Mellon University [‡]New York University

The halo2 Book

Recursion

Alternative terms: Induction; Accumulation scheme; Proof-carrying data

However, the computation of G requires a length-2^k multiexponentiation $\langle \mathbf{G}, \mathbf{s} \rangle$, where $\mathbf{s}$ is composed of the round challenges $u_1, \cdots, u_k$ arranged in a binary counting structure. This is the linear-time computation that we want to amortise across a batch of proof instances. Instead of computing G, notice that we can express G as a commitment to a polynomial

Smaller Computations

- Usually the computation we are trying to prove has to be translated into a language the SNARK can read.

- i.e. we must arithmetise the computation.

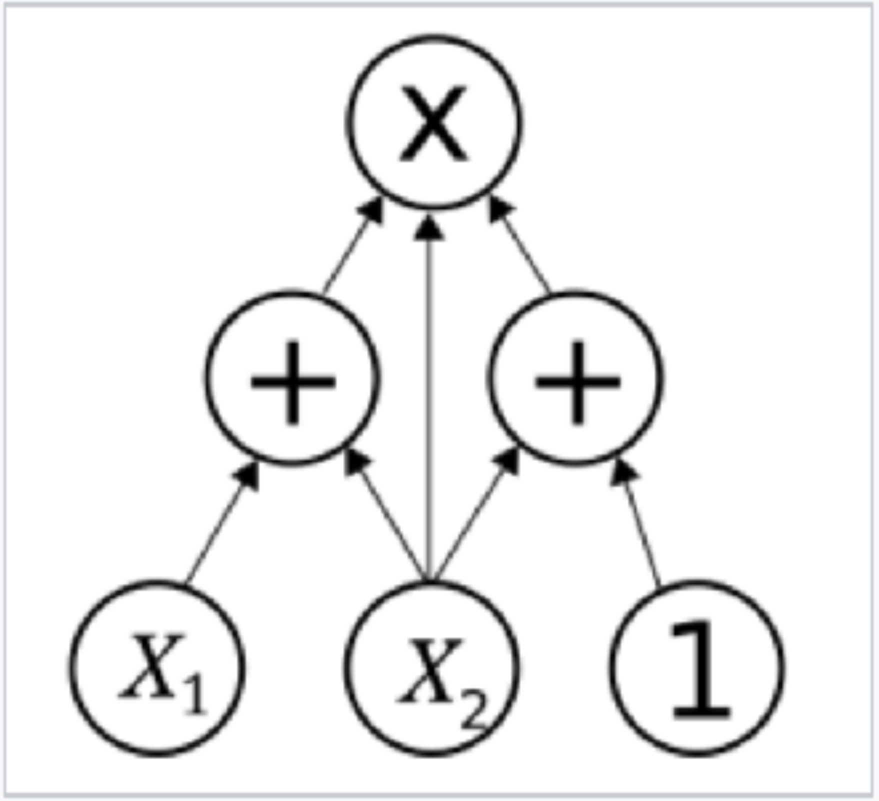
- The better our translation the faster the SNARK.

The halo2 Book

PLONKish Arithmetization

The arithmetization used by Halo 2 comes from PLONK, or more precisely its extension UltraPLONK that supports custom gates and lookup arguments. We'll call it ***PLONKish***.

https://zcash.github.io/halo2/concepts/arithmetization.html



A simple arithmetic circuit.

https://en.wikipedia.org/wiki/Arithmetic_circuit_complexity

R1CS

We now have the computation that we want to prove, nicely expressed as a **sequence of lines** that simply assign the result of the multiplication of sums of variables to a new variable: great!

That's not enough though: to build our "Pinocchio" zk-Snark that can prove that we did indeed run our computation, without revealing the values used when doing it, we need to take it one step further: make **another transformation** that describes the same computation in a different format.

https://www.zeroknowledgeblog.com/index.php/the-pinocchio-protocol/r1cs

Constraint System Proof

A constraint system is a collection of two kinds of constraints:

Multiplicative constraint (*secret-secret multiplication*):

$$x \cdot y = z$$

Linear constraint (*secret variables with cleartext weights*):

$$a \cdot x + b \cdot y + c \cdot z + \ldots = 0$$

https://cathieyun.medium.com/building-on-bulletproofs-2faa58afoba8

Smaller Computations

- Usually the computation we are trying to prove has to be translated into a language the SNARK can read.
- i.e. we must arithmetise the computation.
- The better our translation the faster the SNARK.
- Recent research directions look into minimising SNARK prover costs.

Doubly Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time

Jiaheng Zhang*, Tianyi Liu**, Weijie Wang***, Yinuo Zhang*, Dawn Song*, Xiang Xie†, Yupeng Zhang**.

Abstract. We propose a new doubly efficient interactive proof protocol for general arithmetic circuits. The protocol generalizes the interactive proof for layered circuits proposed by Goldwasser, Kalai and Rothblum to arbitrary circuits, while preserving the optimal prover complexity that is strictly linear to the size of the circuits. The proof size remains succinct for low depth circuits and the verifier time is sublinear for structured circuits. We then construct a new zero knowledge argument scheme for general arithmetic circuits using our new interactive proof protocol together with polynomial commitments.

Our key technique is a new sumcheck equation that reduces a claim about the output of one layer to claims about its input only, instead of claims about all the layers above which inevitably incurs an overhead proportional to the depth of the circuit. We developed efficient algorithms for the prover to run this sumcheck protocol and to combine multiple claims back into one in linear time in the size of the circuit.

Not only does our new protocol achieve optimal prover complexity asymptotically, but it is also efficient in practice. Our experiments show that it only takes 0.3 seconds to generate the proof for a circuit with more than 600,000 gates, which is 13 times faster than the original interactive proof protocol on the corresponding layered circuit. The proof size is 208 kilobytes and the verifier time is 66 milliseconds. Our implementation can take general arithmetic circuits directly, without transforming them to layered circuits with a high overhead on the size of the circuit.

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.orru@berkeley.edu
UC Berkeley

Abstract

We introduce and study *elastic SNARKs*, a class of succinct arguments where the prover has multiple configurations with different time and memory tradeoffs, which can be selected depending on the execution environment and the proved statement. The output proof is independent of the chosen configuration.

We construct an elastic SNARK for rank-1 constraint satisfiability (R1CS). In a time-efficient configuration, the prover uses a linear number of cryptographic operations and a linear amount of memory. In a space-efficient configuration, the prover uses a quasilinear number of cryptographic operations and a logarithmic amount of memory. A key component of our construction is an elastic probabilistic proof. Along the way, we also formulate a streaming framework for R1CS that we deem of independent interest.

We additionally contribute Gemini, a Rust implementation of our protocol. Our benchmarks show that Gemini, on a single machine, supports R1CS instances with tens of billions of constraints.

Keywords: interactive oracle proofs; SNARKs; streaming algorithms

Linear-Time Arguments with Sublinear Verification from Tensor Codes

Jonathan Bootle
jonathan.bootle@berkeley.edu
UC Berkeley

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Jens Groth
jens@dfinity.org
Dfinity

December 28, 2020

Abstract

Minimizing the computational cost of the prover is a central goal in the area of succinct arguments. In particular, it remains a challenging open problem to construct a succinct argument where the prover runs in linear time and the verifier runs in polylogarithmic time.

We make progress towards this goal by presenting a new linear-time probabilistic proof. For any fixed $\epsilon > 0$, we construct an interactive oracle proof (IOP) that, when used for the satisfiability of an N-gate arithmetic circuit, has a prover that uses $O(N)$ field operations and a verifier that uses $O(N^\epsilon)$ field operations. The sublinear verifier time is achieved in the holographic setting for every circuit (the verifier has oracle access to a linear-size encoding of the circuit that is computable in linear time).

When combined with a linear-time collision-resistant hash function, our IOP immediately leads to an argument system where the prover performs $O(N)$ field operations and hash computations, and the verifier performs $O(N^\epsilon)$ field operations and hash computations (given a short digest of the N-gate circuit).

Keywords: interactive oracle proofs; tensor codes; succinct arguments

Lookup Arguments

- Lookup arguments can be used to reduce the number of constraints required to represent a computation.

- They are an extension to arithmetic circuits/ r1cs/ cairo/ tiny ram etc.

plookup: A simplified polynomial protocol for lookup tables

Ariel Gabizon, Aztec

Zachary J. Williamson, Aztec

November 20, 2020

Abstract

We present a protocol for checking the values of a committed polynomial $f \in \mathbb{F}_{<n}[X]$ over a multiplicative subgroup $H \subset \mathbb{F}$ of size n, are contained in the values of a table $t \in \mathbb{F}^d$. Our protocol can be viewed as a simplification of one from Bootle et. al [BCG+] for a similar problem, with potential efficiency improvements when $d \leq n$. In particular, [BCG+]'s protocol requires comitting to several auxiliary polynomials of degree $d \cdot \log n$, whereas ours requires three commitments to auxiliary polynomials of degree n, which can be much smaller in the case $d \sim n$.

One common use case of this primitive in the zk-SNARK setting is a "batched range proof", where one wishes to check all of f's values on H are in a range $[0, \ldots, M]$. We present a slightly optimized protocol for this special case, and pose improving it as an open problem.

zcash.github.io/halo2/design/proving-system/lookup.html

The halo2 Book

Lookup argument

Halo 2 uses the following lookup technique, which allows for lookups in arbitrary sets, and is arguably simpler than Plookup.

Arya: Nearly Linear-Time Zero-Knowledge Proofs for Correct Program Execution *

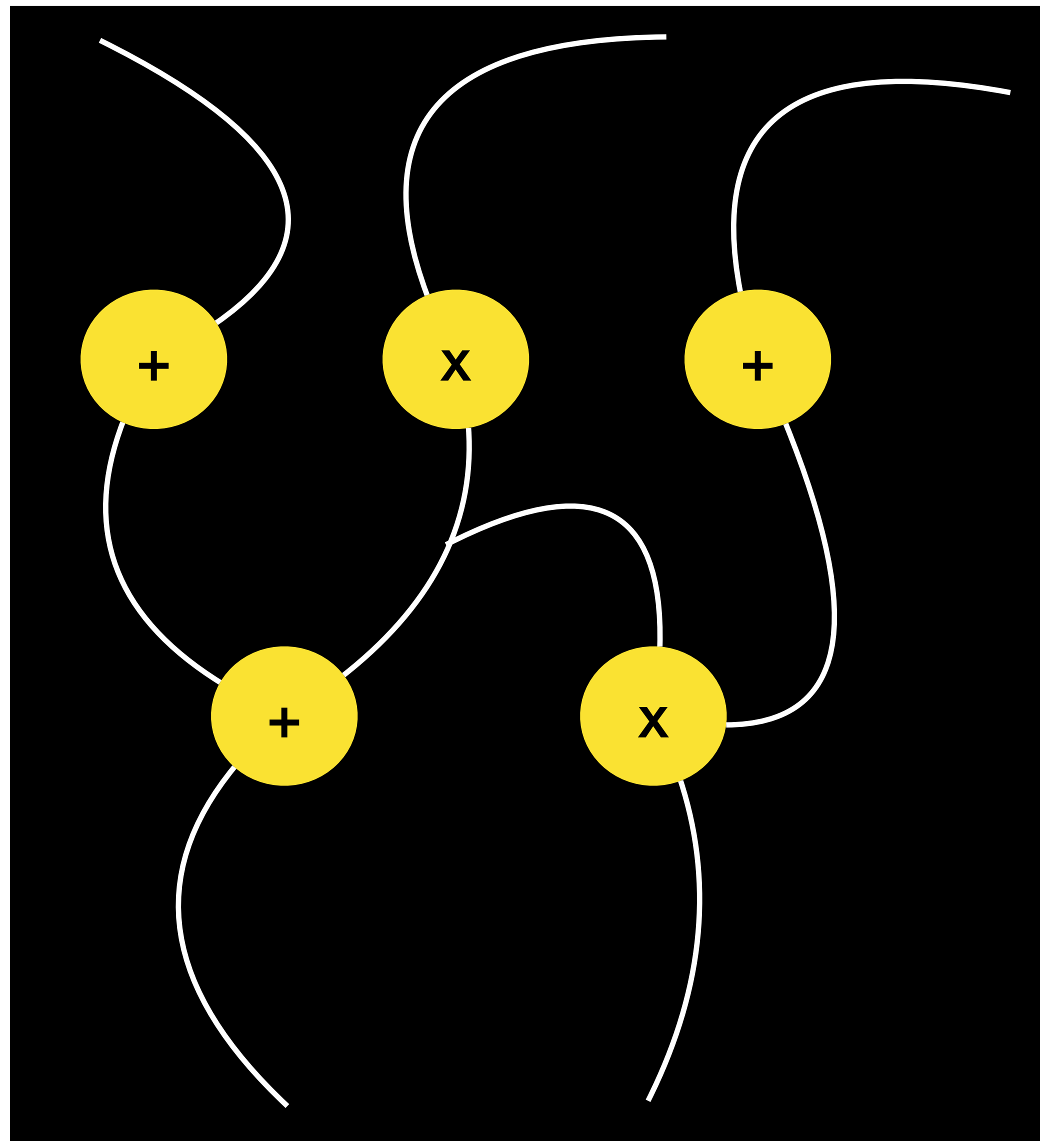
Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, Mary Maller **

University College London

The Power of Plookup

- Suppose Gadget A is used over and over and over again in the circuit.

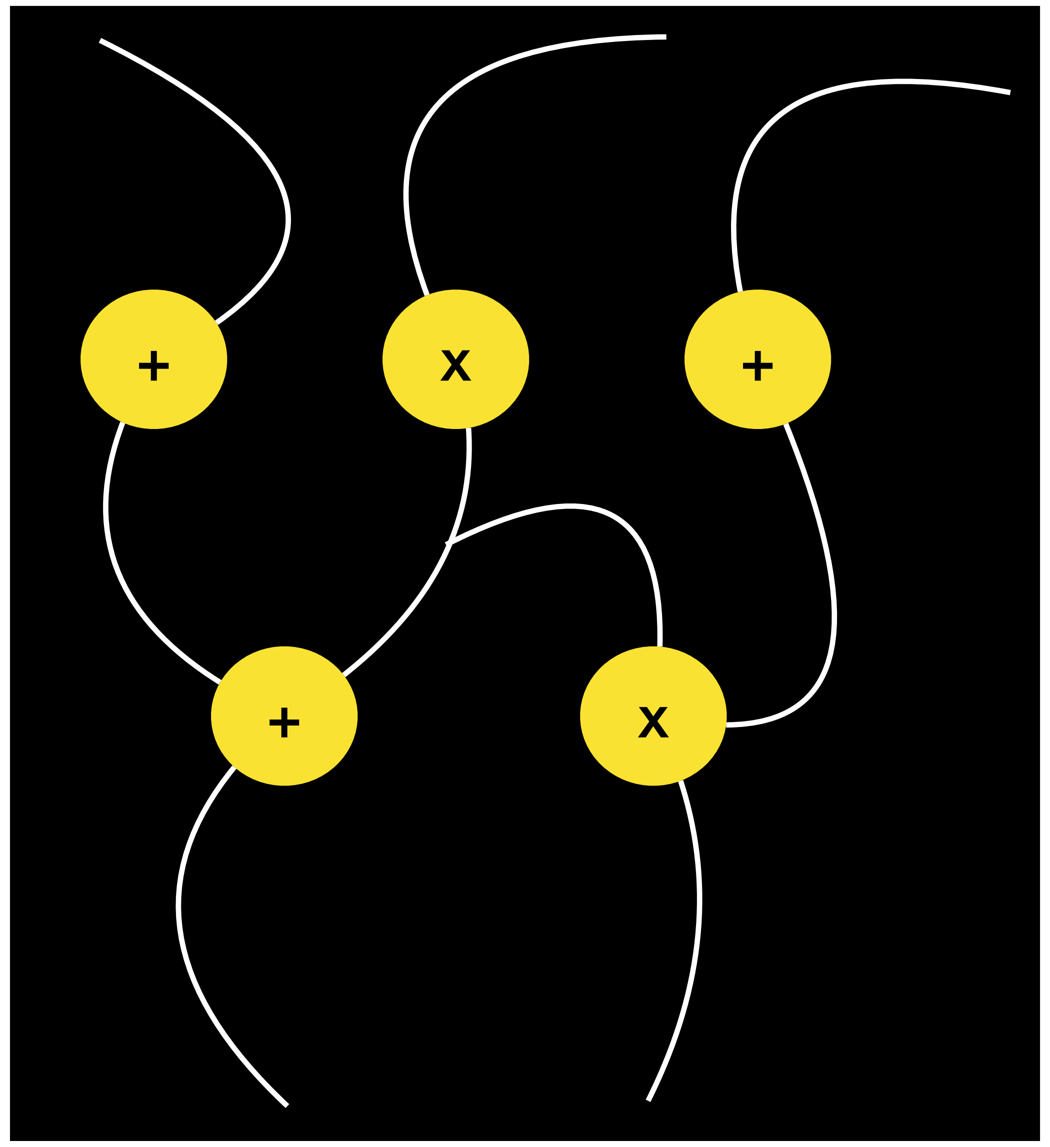
- Don't check with arithmetic gates that gadget A is correct, instead "lookup" whether the result is in a precomputed set.



Most circuits are composed of several sub circuits that are used multiple times.

The Power of Plookup

- Suppose Gadget A is used over and over and over again in the circuit.
- Don't check with arithmetic gates that gadget A is correct, instead "lookup" whether the result is in a precomputed set.



Most circuits are composed of several sub circuits that are used multiple times.

Can instead check if a wire is included in a set of precomputed values.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

Recent personal research: Caulk

Very New Result

Caulk: Lookup Arguments in Sublinear Time

Arantxa Zapico*[1], Vitalik Buterin[2], Dmitry Khovratovich[2], Mary Maller[2], Anca Nitulescu[3], and Mark Simkin[2]

[1] Universitat Pompeu Fabra†
[2] Ethereum Foundation‡
[3] Protocol Labs§

- We build a zero-knowledge lookup argument that has fast prover time.

- The prover is m^2 + m log(N) for N the size of the table and m the number of lookups.

- Proof size is constant.

- Verification is a constant number of pairings.

Very New Result

Caulk: Lookup Arguments in Sublinear Time

Arantxa Zapico*[1], Vitalik Buterin[2], Dmitry Khovratovich[2], Mary Maller[2], Anca Nitulescu[3], and Mark Simkin[2]

[1] Universitat Pompeu Fabra[†]
[2] Ethereum Foundation[‡]
[3] Protocol Labs[§]

- Story: Vitalik had an idea for how to do fast membership proofs .
- i.e. membership proofs up to 100x faster than Poseidon Merkle trees.

$w^N = 1$

f(w) = y for w a "root of unity"

Very New Result

Caulk: Lookup Arguments in Sublinear Time

Arantxa Zapico[*1], Vitalik Buterin[2], Dmitry Khovratovich[2], Mary Maller[2], Anca Nitulescu[3], and Mark Simkin[2]

[1] Universitat Pompeu Fabra[†]
[2] Ethereum Foundation[‡]
[3] Protocol Labs[§]

- Story: Vitalik had an idea for how to do fast membership proofs .
- i.e. membership proofs up to 100x faster than Poseidon Merkle trees.

w^N = 1

f(w) = y for w a "root of unity"

for each w, store pi = Proof that f(w) = y

Very New Result

Caulk: Lookup Arguments in Sublinear Time

Arantxa Zapico*[1], Vitalik Buterin[2], Dmitry Khovratovich[2], Mary Maller[2], Anca Nitulescu[3], and Mark Simkin[2]

[1] Universitat Pompeu Fabra[†]
[2] Ethereum Foundation[‡]
[3] Protocol Labs[§]

- Story: Vitalik had an idea for how to do fast membership proofs .
- i.e. membership proofs up to 100x faster than Poseidon Merkle trees.

w^N = 1

This proof is fast

This proof is log(N)

f(w) = y for w a “root of unity”

for each w, store pi = Proof that f(w) = y

Given y prove knowledge of pi and w

Also prove w^N = 1

Very New Result

Caulk: Lookup Arguments in Sublinear Time

Arantxa Zapico[*1], Vitalik Buterin[2], Dmitry Khovratovich[2], Mary Maller[2], Anca Nitulescu[3], and Mark Simkin[2]

[1] Universitat Pompeu Fabra[†]
[2] Ethereum Foundation[‡]
[3] Protocol Labs[§]

- Story: Vitalik had an idea for how to do fast membership proofs .
- i.e. membership proofs up to 100x faster than Poseidon Merkle trees.
- I made the proof zero-knowledge and started formalising.

f(w) = y for w a “root of unity”

for each w, store pi = Proof that f(w) = y

Given y prove knowledge of pi and w

Also prove w^N = 1

Very New Result

Caulk: Lookup Arguments in Sublinear Time

Arantxa Zapico[*1], Vitalik Buterin[2], Dmitry Khovratovich[2], Mary Maller[2],
Anca Nitulescu[3], and Mark Simkin[2]

[1] Universitat Pompeu Fabra[†]
[2] Ethereum Foundation[‡]
[3] Protocol Labs[§]

- Arantxa started an internship with me at the EF.
- Anca suggested extending the results to "batch" membership proofs (i.e. lookup arguments).
- Arantxa and I explored how to do this efficiently.
- Mark and Arantxa explore definitions of "linkability"

We use a "non-zk" membership proof as a starting point.

Paper 2020/527

Aggregatable Subvector Commitments for Stateless Cryptocurrencies

Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich

Very New Result

Caulk: Lookup Arguments in Sublinear Time

Arantxa Zapico*[1], Vitalik Buterin[2], Dmitry Khovratovich[2], Mary Maller[2], Anca Nitulescu[3], and Mark Simkin[2]

[1] Universitat Pompeu Fabra[†]
[2] Ethereum Foundation[‡]
[3] Protocol Labs[§]

- Dmitry and I implement the scheme in rust.
- Results are much better than Merkle trees.
- Comparison with RSA accumulators depends on the size of m.
- Result went out on 23rd May 2022

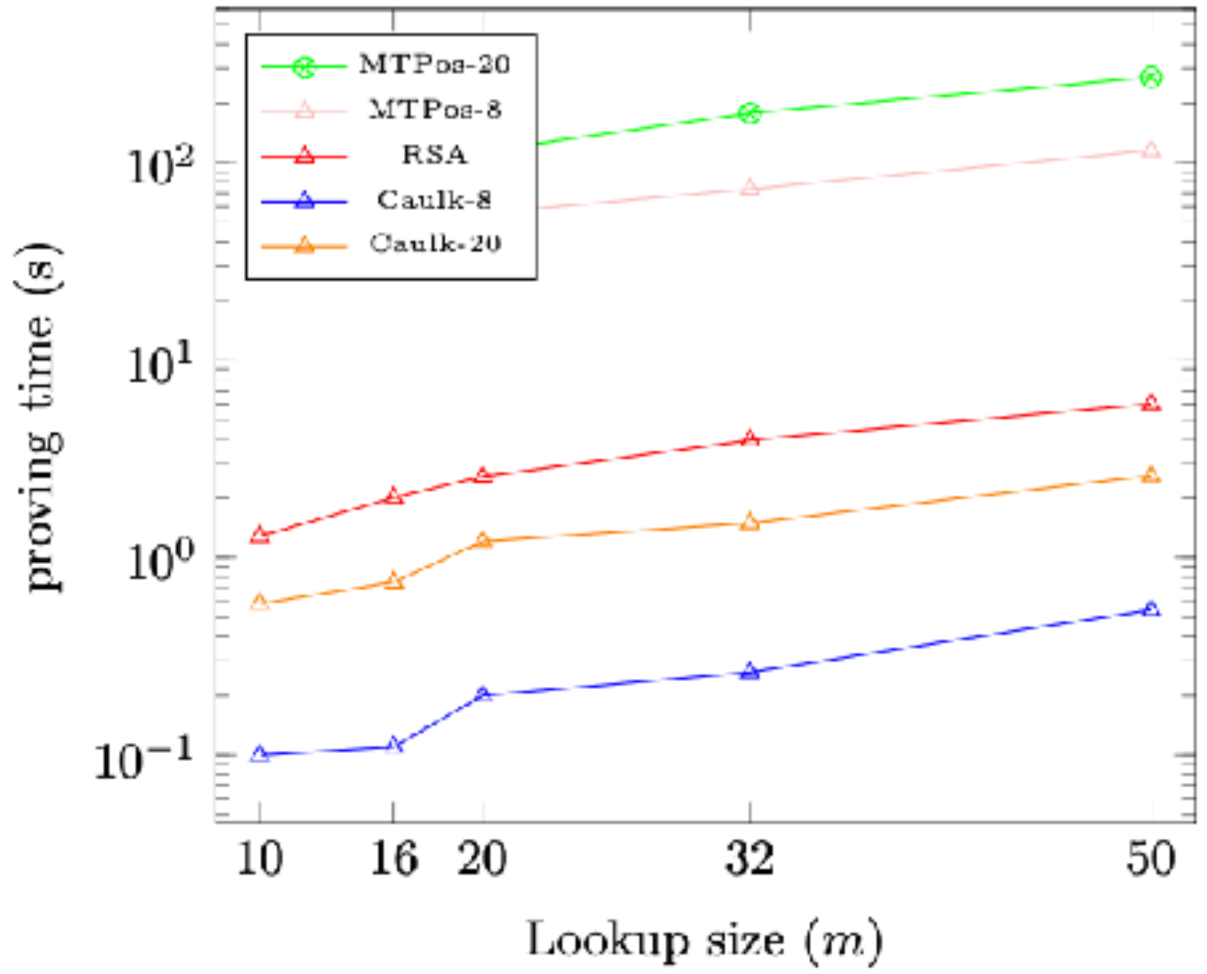


Figure 6: Comparison for lookup tables

https://github.com/caulk-crypto/caulk

Thank-you for listening