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

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

\((x, y, \pi)\)

y is correct
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

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

\( (x, y, \pi) \rightarrow y \text{ is correct} \)
Why do we care about SNARKs?

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

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
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.

\[ \pi \in \text{SNARK}(x, y) \]

(y, \pi)

exists x such that y is correct
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.

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

(exists x such that y is correct)
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 Privacy applications

Introducing Multi-Party ECDSA library

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.
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 from 10s 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.

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

- 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.
- Storage requirements are reasonable.

Prover storage can depend on the computation.

Verifier storage starting from 200 bytes.
• 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

https://en.wikipedia.org/wiki/P_versus_NP_problem
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.
Implementing SNARKs securely is really really hard.

Prover time is slow.
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 \textit{really} matters that the security proof is correct.

- As a community we are still learning the best practices for how to ensure this.
Case Study: The Groth16 SNARK

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

On the Size of Pairing-based Non-interactive Arguments

Jens Groth**
University College London, UK
groth@ucl.ac.uk

Snarky Ceremonies
Markulf Kohlwein1, Mary Mall1, Janno Sim4, Mikhail Volkov2

Audits
Waiting a While
Independent Proofs
Peer Review

The Algebraic Group Model and its Applications
Georg Fuchsbauer1 Eike Kiltz2 Julian Loss2
April 15, 2019

1 INRIA, CNRS, PSL, France
gregor.fuchsbauer@inria.fr
2 Sik University Bonn, Germany
eike.kiltz@uni-bonn.de

Another Look at Extraction and Randomization of Groth’s zk-SNARK
Karim Baghery1, Markulf Kohlwein1,3, Janno Sim4,4, and Mikhail Volkov3

1 Inno-CORS, KU Leuven, Belgium
2 KU Leuven
3 The University of Edinburgh, UK
4 University of Tartu, Estonia
ashkiwi, janno, mikhail.vechkov@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.

This is the new block.
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.
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...

15 DAYS LATER

Killarti

Whisk sounds really complex and heavy. Have you considered Algorand's model? It seems to be a lot simpler solution to this problem. One drawback I can see with it is that each slot gets multiple proposals which results into extra communication, but its significantly less than Whisk requires.
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.
WIP: A Shuffle Argument for ETH 2.0

The Challenges

- 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.
Case Study: Zero-Knowledge Standardisation Effort

- 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

- 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.
• 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.
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.
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.


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

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

https://zcash.github.io/halo2/concepts/arithmetization.html
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.
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/ rics/ cairo/ tiny ram etc.

lookup: 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 value of a committed polynomial \( f \in \mathbb{F}_n[X] \) over a multiplicative subgroup \( H \subset \mathbb{F} \) of size \( n \), contained in the values of a table \( t \in \mathbb{F}^n \). Our protocol can be viewed as a simplification of one from Bootle et al. [CCC ’18] for a similar problem, with potential efficiency improvements when \( n \ll \mathbb{F} \).

In particular, [CCC ’18]’s protocol requires committing 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 of \( n \).

Our 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.

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

Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, Mary Mallor **
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 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.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>
Recent personal research: Caulk
Very New Result

Caulk: Lookup Arguments in Sublinear Time

Arantxa Zapico\textsuperscript{1}, Vitalik Buterin\textsuperscript{2}, Dmitry Khovratovich\textsuperscript{2}, Mary Maller\textsuperscript{2}, Anca Nitulescu\textsuperscript{1}, and Mark Sinkin\textsuperscript{3}

\textsuperscript{1} Universitat Pompeu Fabra\textsuperscript{1}
\textsuperscript{2} Ethereum Foundation\textsuperscript{2}
\textsuperscript{3} Protocol Labs\textsuperscript{3}

• 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\textsuperscript{1}, Vitalik Buterin\textsuperscript{2}, Dmitry Khovratovich\textsuperscript{2}, Mary Maller\textsuperscript{2}, Anca Nitulescu\textsuperscript{1}, and Mark Sinkin\textsuperscript{3}

\textsuperscript{1} Universitat Pompeu Fabra\textsuperscript{1}  
\textsuperscript{2} Ethereum Foundation  
\textsuperscript{3} Protocol Labs\textsuperscript{4}

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

\[ w^N = 1 \]

\[ f(w) = y \text{ for } w \text{ a “root of unity”} \]
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 Sinkin\(^3\)

\(^1\) 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 \(pi = \text{Proof that} f(w) = y\)
Very New Result

Caulk: Lookup Arguments in Sublinear Time

Arantxa Zapico¹, Vitalik Buterin², Dmitry Khovratovich², Mary Maller², Anca Nitulescu³, and Mark Sinkin³

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.

\[ f(w) = y \text{ for } w \text{ a “root of unity”} \]

\[ w^N = 1 \text{ for each } w \]

store \( \pi = \text{Proof} \) that \( f(w) = y \)

Given \( y \) prove knowledge of \( \pi \) and \( w \)

Also prove \( w^N = 1 \)

This proof is fast

This proof is \( \log(N) \)
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 Sinkin$^3$

$^1$ 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.

\[ f(w) = y \text{ for } w \text{ a "root of unity"} \]

for each \( w \), store \( \pi = \text{Proof} \) that \( f(w) = y \)

Given \( y \) prove knowledge of \( \pi \) and \( w \)

Also prove \( w^N = 1 \)
Very New Result

- 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.
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 Sinkin$^3$

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