SNARKPack
Practical SNARK Aggregation

Joint work with
Nicolas Gailly, Mary Maller

Anca Nitulescu
Protocol Labs
In Brief

SNARKs

SNARKPack

log n
SNARK

Succinctness
proof size independent
of NP witness size
SNARK

Succinctness
proof size independent of NP witness size

Non-Interactivity
no exchange between prover and verifier

SNARK
SNARK

- **Succinctness**: proof size independent of NP witness size
- **Non-Interactivity**: no exchange between prover and verifier
- **Argument**: soundness holds only against computationally bounded provers
SNARK

Succinctness
proof size independent of NP witness size

Non-Interactivity
no exchange between prover and verifier

Knowledge Soundness
a witness can be efficiently extracted from the prover

Argument
soundness holds only against computationally bounded provers
zk-SNARK

Zero-Knowledge
does not leak anything about the witness

Succinctness
proof size independent of NP witness size

Non-Interactivity
no exchange between prover and verifier

Knowledge Soundness
a witness can be efficiently extracted from the prover

Argument
soundness holds only against computationally bounded provers
Proof of storage

**Storage Providers**
- onboard storage capacity
- earn block rewards
- regularly prove the storage
  = **Provers**

**Nodes in network**
- ensure data is being stored, maintained, and secured
- need to check proofs of space
  = **Verifiers**
Proof of storage

Proves 32GB

Proves 32GB

Proves 32GB

submit on chain

10 SNARKs

10 SNARKs

10 SNARKs

Block

40PiB per day collective storage onboarding limit
Verify many SNARKs

Batch Verification

Proof Size

Verification Time
Verify many **SNARKs**

- **Batch Verification**
  - Proof Size
  - Verification Time

- **Aggregation**
  - Proof Size
  - Verification Time
How does it work?
Groth16

Bilinear Groups

\[ \langle g \rangle = \mathbb{G}_1, \langle h \rangle = \mathbb{G}_2 \]
\[ e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T \]
\[ e(g^a, h^b) = e(g, h)^{ab} \]
Groth16

Bilinear Groups
\[ \langle g \rangle = G_1, \; \langle h \rangle = G_2 \]
\[ e : G_1 \times G_2 \rightarrow G_T \]
\[ e(g^a, h^b) = e(g, h)^{ab} \]
Groth16

\[ A \pi B = C \]

Bilinear Groups
\[ \langle g \rangle = G_1, \quad \langle h \rangle = G_2 \]
\[ e : G_1 \times G_2 \to G_T \]
\[ e(g^a, h^b) = e(g, h)^{ab} \]
Groth16

SNARK

\[ e(A, B) = e(C, D) \]

Bilinear Groups

\[ \langle g \rangle = G_1, \langle h \rangle = G_2 \]

\[ e : G_1 \times G_2 \rightarrow G_T \]

\[ e(g^a, h^b) = e(g, h)^{ab} \]
Many SNARKs

Proofs

\[ \{ A_i, B_i, C_i \}_{i=1,n} \]

Verification \((D = g^d)\)

\[ e(A_i, B_i) = e(C_i, D) \]
Many SNARKs

Verification ($D = g^d$)

Proofs

$$e(A_1, B_1) = e(C_1, D)$$

$$e(A_2, B_2) = e(C_2, D)$$

$$\ldots$$

$$e(A_n, B_n) = e(C_n, D)$$
SNARK Batching

Verification

\[ e(A_1, B_1) = e(C_1, D) \]
\[ e(A_2, B_2) = e(C_2, D) \]
\[ \ldots \]
\[ e(A_n, B_n) = e(C_n, D) \]
**SNARK Batching**

**Verification**

\[ e(A_1, B_1) = e(C_1, D) \]
\[ e(A_2, B_2) = e(C_2, D) \]

\[ \ldots \]

\[ e(A_n, B_n) = e(C_n, D) \]

\[ r \leftarrow $ \]

\[ r \times e(A_1, B_1) = e(C_1, D) \]
\[ r^2 \times e(A_2, B_2) = e(C_2, D) \]

\[ \ldots \]

\[ r^n \times e(A_n, B_n) = e(C_n, D) \]
SNARK Batching

Verification

\[ e(A_1, B_1) = e(C_1, D) \]
\[ e(A_2, B_2) = e(C_2, D) \]
\[ \ldots \]
\[ e(A_n, B_n) = e(C_n, D) \]

Batch Verification

\[ r \leftarrow \$ \]

\[ \prod e(A_i, B_i)^{r_i} = \prod e(C_i, D)^{r_i} \]
SNARK Aggregation

Batch Verification

\[ \prod e(A_i, B_i)^{r_i} = \prod e(C_i, D)^{r_i} \]

\[ \prod e(A_i, B_i^{r_i}) = e(\prod C_i^{r_i}, D) \]

\[ \Leftrightarrow \]

Bilinear Groups

\( \langle g \rangle = G_1, \quad \langle h \rangle = G_2 \)

\( e : G_1 \times G_2 \rightarrow G_T \)

\( e(g^a, h^b) = e(g, h)^{ab} \)
SNARK Aggregation

Batch Verification

\[ \prod e(A_i, B_i)^{r_i} = \prod e(C_i, D)^{r_i} \]

\[ \prod e(A_i, B_i^{r_i}) = e(\prod C_i^{r_i}, D) \]

Aggregation

\[ \prod e(A_i, B_i^{r_i}) = e(\prod C_i^{r_i}, D) \]

\[ i = 1, n \]
**SNARK Aggregation**

Batch Verification

\[
\prod e(A_i, B_i)^{r_i} = \prod e(C_i, D)^{r_i}
\]

\[
\prod e(A_i, B_i^{r_i}) = e(\prod C_i^{r_i}, D)
\]

Aggregation

\[
Z_{AB} = \prod e(A_i, B_i^{r_i})
\]

\[
Z_c = \prod C_i^{r_i}
\]
SNARK Aggregation

Batch Verification

\[ \prod e(A_i, B_i)^{r_i} = \prod e(C_i, D)^{r_i} \]

\[ Z_{AB} = e(Z_C, D) \]

Aggregation

\[ Z_{AB} = \prod e(A_i, B_i^{r_i}) \]

\[ Z_C = \prod C_i^{r_i} \]
SNARK Aggregation

Batch Verification

\[ \prod e(A_i, B_i)^{r_i} = \prod e(C_i, D)^{r_i} \]

\[ Z_{AB} = e(Z_C, D) \]

Aggregation

\[ Z_{AB} = \prod e(A_i, B_i)^{r_i} \]

\[ Z_C = \prod C_i^{r_i} \]
Construction
Tools: MIPP & TIPP

Proofs for Inner Pairing Products and Applications - Bünz, Maller, Mishra, Tyagi, Vesely

\[ \langle A, b \rangle = \prod A_i^{b_i} \]

\[ \langle A, B \rangle = \prod e(A_i, B_i) \]

\[ A_i \in G_1, B_i \in G_2, b_i \in \mathbb{Z}_q \]
Tools: **MIPP & TIPP**

Proofs for Inner Pairing Products and Applications - Bünz, Maller, Mishra, Tyagi, Vesely

\[ \langle A, b \rangle = \prod A_i^{b_i} \]

\[ \langle A, B \rangle = \prod e(A_i, B_i) \]

\[ Z_c = \prod C_i^{r_i} \]

\[ Z_{AB} = \prod e(A_i, B_i^{r_i}) \]

Aggregation
Tools: MIPP & TIPP

Proofs for Inner Pairing Products and Applications - Bünz, Maller, Mishra, Tyagi, Vesely

\[ \langle A, b \rangle = \prod A_i^{b_i} \]

\[ \langle A, B \rangle = \prod e(A_i, B_i) \]

\[ Z_C = \langle C, r \rangle \]

\[ Z_{AB} = \langle A, B^r \rangle \]

Aggregation
Tools: MIPP & TIPP

Proofs for Inner Pairing Products and Applications - Bünz, Maller, Mishra, Tyagi, Vesely

\[ \langle C, r \rangle = \prod C_i^{r_i} \]

\[ \langle A, B^r \rangle = \prod e(A_i, B_i^{r_i}) \]

\[ Z_C = \langle C, r \rangle \]

\[ Z_{AB} = \langle A, B^r \rangle \]

Aggregation
MIPP & TIPP Strategy

Proofs for Inner Pairing Products and Applications - Bünz, Maller, Mishra, Tyagi, Vesely

\[ Z_{C} = \langle C, r \rangle \]

\[ Z_{AB} = \langle A, B^{r} \rangle \]
Problem: **Trusted Setup**

Proofs for Inner Pairing Products and Applications - Bünz, Maller, Mishra, Tyagi, Vesely

\[ Z_C = \langle C, r \rangle \]

\[ Z_{AB} = \langle A, B^r \rangle \]
Trusted Setup
Aggregation from existing CRS
Trusted Setup

Groth16: Monomials/ Powers of tau

Bilinear Groups

\[ \langle g \rangle = G_1, \quad \langle h \rangle = G_2 \]
\[ e : G_1 \times G_2 \rightarrow G_T \]
\[ e(g^a, h^b) = e(g, h)^{ab} \]
SNARK Aggregation

Filecoin: Powers of Tau

Zcash: Powers of Tau
Commitments

\[ \text{Com}(C) = \langle C, v_1 \rangle, \langle C, v_2 \rangle \]
Commitments

\[ v_1 = h^\alpha, h^{\alpha^2}, \ldots, h^{\alpha^d} \]
\[ w_1 = g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^d} \]
\[ v_2 = h^\beta, h^{\beta^2}, \ldots, h^{\beta^d} \]
\[ w_2 = g^\beta, g^{\beta^2}, \ldots, g^{\beta^d} \]

\[ \text{Com}(A, B) = \langle A, v_1 \rangle \langle w_1, B \rangle, \langle A, v_2 \rangle \langle w_2, B \rangle \]
SNARK Aggregation

\[
\{ A_i, B_i, C_i \} \quad i=1,n
\]
SNARK Aggregation

\[ \text{Aggregation} \]

\[ \{ A_i, B_i, C_i \} \rightarrow \text{Com}(A, B) \text{ and } \text{Com}(C) \]

\[ i=1,n \]
SNARK Aggregation

Aggregation

\[ \text{MIPP: } \langle C, r \rangle = \prod C_i^{r_i} \]

\[ \text{TIPP: } \langle A, B^r \rangle = \prod e(A_i, B_i^{r_i}) \]

\[ \log n \]
SNARK Aggregation

Aggregation

\[
\{ A_i, B_i, C_i \}_{i=1}^{n} \]

**MIPP:**

\[
\langle C, r \rangle = \prod C_i^{r_i}
\]

**TIPP:**

\[
\langle A, B^r \rangle = \prod e(A_i, B_i^{r_i})
\]

- \( \log n \)
- \( Z_C = \langle C, r \rangle \)
- \( Z_{AB} = \langle A, B^r \rangle \)
SNARK Aggregation

\[
\begin{align*}
Z_C &= \langle C, r \rangle \\
Z_{AB} &= \langle A, B^r \rangle
\end{align*}
\]

Verification

\[
Z_{AB} = e(Z_C, D)
\]
Implementation
Library

- Coded in **Rust**, available at [https://github.com/filecoin-project/bellperson](https://github.com/filecoin-project/bellperson)
- Initial code from the **arkworks library** [https://github.com/arkworks-rs/ripp/](https://github.com/arkworks-rs/ripp/)
- Ported & optimized in the **bellman** framework (bellperson fork)
- Using **BLS12-381 curves** from the **blst library** [https://github.com/supranational/blst](https://github.com/supranational/blst)
- **SRS** combined from Filecoin & Zcash “power of taus”
  - Code at [https://github.com/nikkolasg/taupipp](https://github.com/nikkolasg/taupipp)
  - Up to $2^{19}$
- Benchmark performed on 32c/64t **AMD Raizen Threadripper**
- **Audited** by NCC
Verifier Time

- Verifying aggregate proofs becomes faster from **32 proofs**
- **8192 proofs in 33ms**
  - “ratio” of 0.004 ms per proof
- Including **unserialization**
- **Optimizations:**
  - Relies heavily on parallelism
  - MIPP/TIPP combined
  - Batching for pairing checks
Proof size

- Use **compression** of target group points
  - based on Torus compression
  - credits - RELIC library implementation

- Turnover at 128 proofs
  - 23kB for aggregated
  - 24kB for “all proofs”
Aggregation Time

- 8.7s for 192 proofs
- Relies heavily on parallelism
- $2^{17}$ proofs in ~2mn
Application: Filecoin
Proof of storage

Allows for 2000X more proofs of storage on chain
Proof of storage

Allows for **2000 X** more proofs of storage on chain
Proof of storage

Allows for **2000 X** more proofs of storage on chain

- Sector 32 GB (10 SNARKs)
- Sector 32 GB (10 SNARKs)
- Sector 32 GB (10 SNARKs)
- Sector 32 GB (10 SNARKs)
Proof of storage

Allows for 2000 X more proofs of storage on chain
Verify aggregated SNARKs

- Aggregated Proof (~2000 SNARKs) → Valid proofs (20 ms)
- Aggregated Proof (~2000 SNARKs) → Valid proofs (20 ms)
- Aggregated Proof (~2000 SNARKs) → Valid proofs (20 ms)

In practice, up to \(x\) 200 more Sectors on chain (~\(x\) 2000 SNARKs)
Conclusion & Questions

- **Trusted Setup:** Main feature is to rely on existing Groth16 CRS - at the cost of slightly more expensive commitment scheme
- **Transparent Aggregation:** What about Aggregating SNARKs without a trusted setup?
- **Optimisations:** Better Curves, Better Commitments, New Inner Pairing Proofs
- **Extension:** Could we extend this scheme to other pairing-based primitives? Currently only supports Groth16
Thanks!

eprint.iacr.org/2021/529
Credits

Special thanks to all those who made and released these resources for free:

- Presentation template by SlidesCarnival
- Illustrations by Iconfinder
**Motivation.** SNARKs are becoming very popular in real-world applications such as delegated computation or blockchain systems: An example of early practical use case, Zerocash showed how that we can deploy zk-SNARKs in distributed ledgers to achieve payment systems with strong privacy guarantees. More recent zk-SNARK use cases are in Ethereum smart contracts for boosting scalability and privacy. Another example of SNARK application is the Filecoin System that implements a decentralized storage solution for the internet. To date, the Filecoin Network is the largest SNARK system in production, producing and verifying over 5 million SNARKs on a daily basis. Due to their rapid and massive adoption, the SNARKs schemes used today start facing new challenges: the generation of trusted setups requires complicated ceremonies, proving large statements has significant overhead, verifying multiple proofs is expensive even with batching, so many blockchain systems have therefore scalability issues.

**Contribution.** In this work, we look into reducing proof size and verifier time for SNARKs even further in order to meet these significant scalability requirements. We design SnarkPack, an argument that allows to aggregate n Groth16 zk-SNARKs with a O(log n) proof size and verifier time. Our scheme is based on a trusted setup that can be constructed from two different ceremonies (e.g. the so-called “powers of tau” for Zcash [zca18] and Filecoin [Fil20]). Being able to rely on the security of well-known trusted setups for which the ceremonies have been largely publicly advertised is a great advantage in practice and makes SnarkPack immediately useful in real-world applications and an easy update to systems already relying on such trusted setups.

We chose to focus on Groth16 proofs and tailor optimisations for this case, since it is the most popular scheme among practitioners. Therefore, SnarkPack is the first real-world aggregation system that can be used in blockchains applications to reduce the on-chain work by employing verifiable outsourcing to process a large number of proofs off-chain. This applies broadly to any system that needs to delegate batches of state updates to an untrusted server. SnarkPack is already deployed on the live Filecoin Network.