Linear-Time SNARKs for R1CS and Friends

Justin Thaler

Georgetown University

Talk outline

- Survey of SNARKs
  - 15 minutes
- Linear-time SNARKs
  - 5 minutes
- SNARK security issues
  - 20 minutes
- Squash: sum-check-based SNARKs for AIR and Plonkish (upcoming work)
  - 0-3 minutes

Survey of SNARKs

SNARKs: What are they

- Prover P claims to know witness w satisfying some property.
  - For example:
    1. A preimage w for a designated output y of a cryptographic hash function h.
       - A w such that $h(w) = y$.
- Trivial proof: send w to V, who directly checks it satisfies the claimed property.
- A SNARK (Succinct Non-interactive ARgument of Knowledge) achieves the same, but with much better costs to V.
  - SNARK proof π should be shorter than the witness w.
    - This is what "succinct" means.
  - Checking π should be much faster than checking w directly.
    - Also called "work-saving" for V.

SNARKs: What are they

- Prover P claims to know witness w satisfying some property.
  - For example:
    1. A preimage w for a designated output y of a cryptographic hash function h.
       - A w such that $h(w) = y$.
- Trivial proof: P sends w to V, who directly checks it satisfies the claimed property.

- A SNARK (Succinct Non-interactive ARgument of Knowledge) achieves the same, but with much better costs to V.
  - SNARK proof π should be shorter than the witness w.
    - This is what "succinct" means.
  - Checking π should be much faster than checking w directly.
    - Also called "work-saving" for V.

SNARKs: What are they

- Prover P claims to know witness w satisfying some property.
  - For example:
    1. A preimage w for a designated output y of a cryptographic hash function h.
       - A w such that $h(w) = y$.
- Trivial proof: P sends w to V, who directly checks it satisfies the claimed property.

- A SNARK (**S**uccinct **N**on-interactive **AR**gument of **K**nowledge) achieves the same, but with better costs to V.
  - SNARK proof π should be shorter than the witness w.
    - This is what "succinct" means.
  - Checking π should be much faster than checking w directly.
    - Also called "work-saving" for V.

SNARKs: What are they

- Prover P claims to know witness w satisfying some property.
  - For example:
    1. A preimage w for a designated output y of a cryptographic hash function h.
       - A w such that $h(w) = y$.
- Trivial proof: P sends w to V, who directly checks it satisfies the claimed property.

- A SNARK (**S**uccinct **N**on-interactive **AR**gument of **K**nowledge) achieves the same, but with better costs to V.
  - SNARK proof π **must** be shorter than the witness w.
    - This is what "succinct" means.
  - Ideally, checking π is faster than checking w directly.
    - Called "work-saving" for V.

Part I: How are SNARKs designed and used?

General-Purpose SNARKs: Standard Paradigm

- Start with a computer program ψ written in high-level programming language (C, Java, etc.).
  - Hash pre-image example:
    - Program takes w as input, applies h to it, and checks that output equals y.
- Step 1: Turn ψ into an equivalent model amenable to probabilistic checking.
  - Typically some variant of circuit-satisfiability.
    - The "circuit" is called an intermediate representation (IR).
    - Examples: arithmetic circuits, R1CS, AIR, "Plonkish IR".
  - Called the Front End of the system.
- Step 2: Run a SNARK for circuit-satisfiability/R1CS/AIR/Plonkish/etc.
  - Called the Back End of the system.

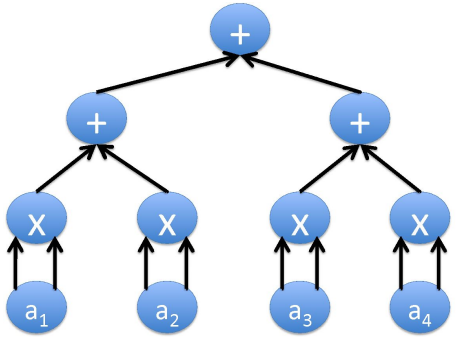
General-Purpose SNARKs: Standard Paradigm

- Start with a computer program ψ written in high-level programming language (C, Java, etc.).
  - Hash pre-image example:
    - Program takes w as input, applies h to it, and checks that output equals y.
- Step 1: Turn ψ into an equivalent model amenable to probabilistic checking.
  - Typically some variant of circuit-satisfiability.
    - The "circuit" is called an **intermediate representation (IR).**
    - Examples: arithmetic circuits, R1CS, AIR, "Plonkish IR".
  - Called the **Front End** of the system.
- Step 2: Run a SNARK for circuit-satisfiability/R1CS/AIR/Plonkish/etc.
  - Called the Back End of the system.

General-Purpose SNARKs: Standard Paradigm

- Start with a computer program ψ written in high-level programming language (C, Java, etc.).
  - Hash pre-image example:
    - Program takes w as input, applies h to it, and checks that output equals y.
- Step 1: Turn ψ into an equivalent model amenable to probabilistic checking.
  - Typically some variant of circuit-satisfiability.
    - The "circuit" is called an **intermediate representation (IR).**
    - Examples: arithmetic circuits, R1CS, AIR, "Plonkish IR".
  - Called the **Front End** of the system.
- Step 2: Run a SNARK for circuit-satisfiability/R1CS/AIR/Plonkish/etc.
  - Called the **Back End** of the system.

Front End



P and V run SNARK (back end) for circuit-satisfiability.

Example frontends

- **Bellman and CirCom**:
  - "Witness-checking" program ψ written in low-level hardware description language.
- **Cairo, ZoKrates, xJSnark, etc**:
  - ψ is written in a restricted domain-specific language.
  - Example restrictions: write-once memory, no support for important datatypes like strings, limited or no control flow.
- Steps toward supporting various higher-level languages:
  - **Various zkEVM projects**:
    - Aim to support EVM bytecode---to which Solidity programs can be compiled---or Solidity directly.
  - **RISC-Zero**:
    - Aims to support the RISC-V instruction set.
    - To which many high-level languages can be compiled.
  - **CirC**: a compiler construction toolkit ("LLVM for SNARK front-ends").
- Linear-time P means:
  - P's runtime within a constant factor of the cost to evaluate the circuit on the witness.

Backends

Backend Goals

- Want a SNARK for circuit-SAT or or other IR for which:
  1. P runs in linear time in the size of the circuit, or as close as possible.
  2. Proof size is as small as possible.
  3. V is as fast as possible.
- Also ideal:
  4. Transparent (no “toxic waste” produced in a trusted setup).
  5. Plausibly post-quantum secure.

Key Paradigm for Backend Design

1. Give a "**polynomial IOP**" for R1CS or circuit-satisfiability.
2. Combine with **polynomial commitment scheme** to get succinct interactive argument.
3. Apply Fiat-Shamir transformation to render non-interactive.

Key Paradigm for Backend Design

1. Give a "**polynomial IOP**" for R1CS or circuit-satisfiability.
2. Combine with **polynomial commitment scheme** to get succinct interactive argument.
3. Apply Fiat-Shamir transformation to render non-interactive.

- This is how all SNARKs are designed other than "linear-PCP based" ones (GGPR13, Pinocchio, Groth16, etc.)
  - Even linear-PCP based SNARKs use techniques **very** similar to certain polynomial commitments (KZG).

What is a Polynomial-IOP?

- P's first message in the protocol is a **polynomial** h**.**
  - V does **not** learn h in full.
    - The description size of h is as large as the circuit.
  - Rather, V is permitted to evaluate h at, say, **one** point.
  - After that, P and V execute a standard interactive proof.

- Ignoring cost of polynomial commitment, P runs in linear time, and proof length is logarithmic [VTBW14, XZZPS19, Setty20].

What is a Polynomial Commitment Scheme?

- High-level idea:
  - P binds itself to a polynomial h by sending a short string $\mathrm{Com}(h)$.
  - V can choose x and ask P to evaluate $h(x)$.
  - P sends y, the purported evaluation, plus a proof π that y is consistent with $\mathrm{Com}(h)$ and x.
- Goals:
  - P cannot produce a convincing proof for an incorrect evaluation.
  - $\mathrm{Com}(h)$ and π are short and easy to generate; π is easy to check.

Part 2: Taxonomy of SNARKs

A Zoo of SNARKs

- There are several different polynomial IOPs in the literature.
- And several different polynomial commitments.
- Can mix-and-match to get different tradeoffs between P time, proof size, setup assumptions, etc.
  - Transparency and plausible post-quantum security determined entirely by the polynomial commitment scheme used.
  - Caveat: when people say they use a "polynomial commitment" they should specify what kind of polynomial is being committed and how it's represented.
  - Typically either:
    - Univariate polynomial $p(X)$ with coefficients given in the standard basis $\{1, X, X^2, \ldots, X^n\}$
    - Multilinear ℓ-variate polynomial with coefficients given in the Lagrange basis.
      - Meaning the coefficients are equal to the polynomial's evaluations over domain $\{0,1\}^\ell$

Polynomial IOPs: Three classes

1. Based on interactive proofs (IPs).
   - Examples: Hyrax, vSQL, Libra, Virgo.
2. Based on multi-prover interactive proofs (MIPs).
   - Examples: Spartan, Brakedown, Xiphos.
3. Based on constant-round polynomial IOPs.
   - Examples: Marlin, PlonK.

- Above SNARKs roughly listed in increasing order of P costs and decreasing order of proof length and V cost.

Polynomial Commitment Schemes: Four Classes

1. Based on pairings (**not** transparent **nor** post-quantum).
   - e.g., **KZG10**, PST13, ZGKPP18.
   - Unique property: constant sized evaluation proofs.
2. Based on discrete logarithm (transparent, **not** post-quantum).
   - Examples: **BCCGP16/Bulletproofs**, Hyrax, Dory*.
3. Based on IOPs + hashing (transparent **and** post-quantum)
   - e.g., Ligero, **FRI**, Brakedown.
4. Groups of unknown order (transparent if using class groups, **not** post-quantum)
   - E.g., DARK, Dew
   - P very slow due to use of class groups.

*Dory requires pairings, not just hardness of discrete-log.

Highlights of SNARK Taxonomy: **Non-transparent SNARKS**

1. Linear-PCP based:
   - **Ex: Groth16**
   - Pros: Shortest proofs (3 group elements), fastest V.
   - Cons: Circuit-specific trusted setup, slow and space-intensive P, not post-quantum
2. Constant-round polynomial IOP + KZG polynomial commitment:
   - **Ex: Marlin, PlonK**
   - Pros: Universal trusted setup.
   - Cons: Proofs are ~4x-6x larger than Groth16, P is slower than Groth16, also not post-quantum.
     - Counterpoint for P: can use more flexible intermediate representations than circuits and R1CS.

Highlights of SNARK Taxonomy: **Transparent SNARKS**

1. [Any polynomial IOP] + Bulletproofs polynomial commitment.
   - **Ex: Halo2**
   - Pros: Shortest proofs among transparent SNARKs.
   - Cons: Slow V
2. [Any polynomial IOP] + FRI polynomial commitment.
   - **Ex: STARKs, Fractal, Aurora, Virgo, Ligero++**
   - Pros: Shortest proofs amongst plausibly post-quantum SNARKs.
   - Cons: Slow P, proofs still quite large
3. MIPs & IPs (a.k.a. sum-check-based) + [fast-prover polynomial commitments].
   - **Ex: Spartan, Brakedown, Orion.**
   - Pros: Fastest P in the literature, plausibly post-quantum + transparent if polynomial commitment is.
   - Cons: Bigger proofs than 1. and 2. above.

Linear-Time SNARKs for R1CS

Brakedown (2021) summary

- Combines a polynomial IOP for R1CS implicit in Spartan (2020) with a new concretely-efficient polynomial commitment scheme.
- The SNARK has:
  - A linear-time prover.
    - Allowed: linear number of field operations, Merkle-hashing
    - Not allowed: FFT of linear size, multi-exponentiation of linear size.
  - Field agnostic (works over any sufficiently-large field).
  - **First** implemented SNARK with either property.
- Proof size is $O(\sqrt{\lambda N})$ where λ is security parameter and N is size of R1CS instance.
  - Sublinear but concretely large (MBs).

Key new technical components in Brakedown

- A practical error-correcting code with linear-time encoding.
  - We give a code for messages in $\boldsymbol{F}^m$ with encoding time ≈ $15m$ and relative distance ≈ 1/25.
  - Previous constructions that we build on [Spielman-96, Druk-Ishai-2014] are impractical.
- The encoding function is randomly generated.
- Probability that code fails to satisfy the relative distance guarantee is at most 2^{-100}.
  - The code is generated **once** for all time.
- We show the resulting polynomial commitment scheme is extractable despite it lacking fast decoding.

Follow-on works: Orion and Hyperplonk

Orion (Xie, Zhang, Song, CRYPTO 2022)

- Composes Brakedown with Virgo, a SNARK with slower prover but smaller proofs.
  - Means representing Brakedown V as a circuit, and supplying a Virgo-proof of a satisfying assignment to that circuit.
    - Makes some modifications to Brakedown V to make it more "SNARK-friendly" [BCG20].
  - Orion leaves Merkle-authentication paths "out of the composition".
    - Because there are so many of them in Brakedown that a circuit to verify them would be large.
  - Results in ~6.5x proof size reduction (caveats later).
  - By using Virgo, which uses the FRI polynomial commitment, Orion **gives up field-agnosticism**. Needs an "FFT-friendly" field.

Hyperplonk [Chen- Bünz-Boneh-Zhang 2022]

- Recall that Orion left Merkle-authentication paths out of the composition with Virgo, leaving the proofs pretty large.
- Hyperplonk proposes to replace Merkle trees in Orion with KZG/PST commitments.
- And use efficient batching properties of KZG/PST evaluation proofs to more succinctly authenticate the values requested by the Brakedown verifier.
- Not yet implemented, but estimated to yield proofs under 10 KBs.
- Gives up transparency, field-agnosticism, and post-quantum security.

Back to Orion

- The probability Brakedown's randomly-generated code fails to satisfy requisite distance property is less than 2^{-100}.
  - Asymptotically, it is N^{-c} for some large constant c.
- Orion **asymptotically** reduces this by checking the code using a densest subgraph algorithm.
- But Orion's concrete security is actually lower (as implemented).

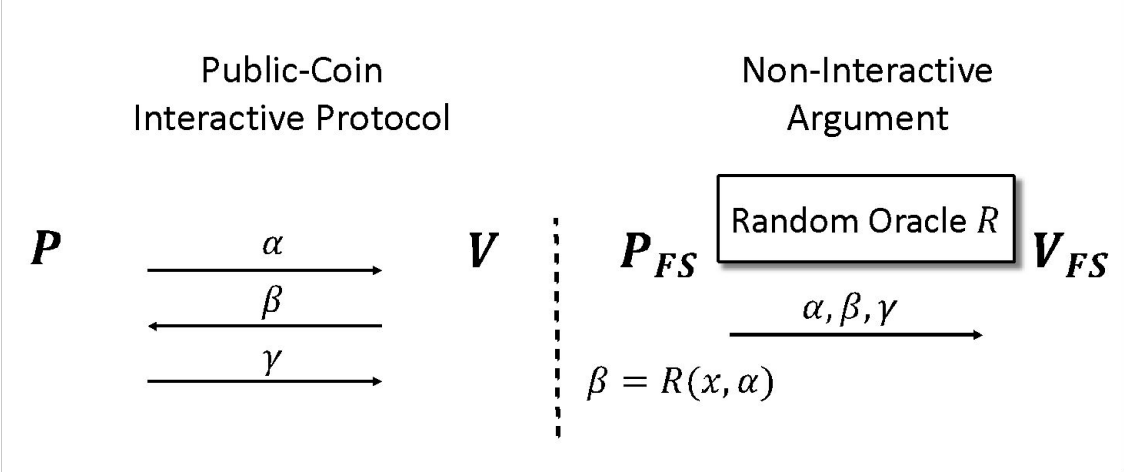
Concrete Security

ZKProof Community Reference Document

"The benchmarks for each technique shall include at least one parametrization **achieving a conjectured computational security level approximately equal to, or greater than, 128 bits**. Each technique should also be benchmarked for at least one additional higher computational security, such as 192 or 256 bits. (If only one, the latter is preferred.)... **The interest in a security level as high as 256 bits can be considered a cautious (and heuristic) safety margin, compared for example with intended 128 bits. This is intended to handle the possibility that the conjectured level of security is later found to have been overestimated.** The evaluation at computational security below 128 bits may be justified for the purpose of clarifying how the execution complexity or time varies with the security parameter, but should not be construed as a recommendation for practical security."

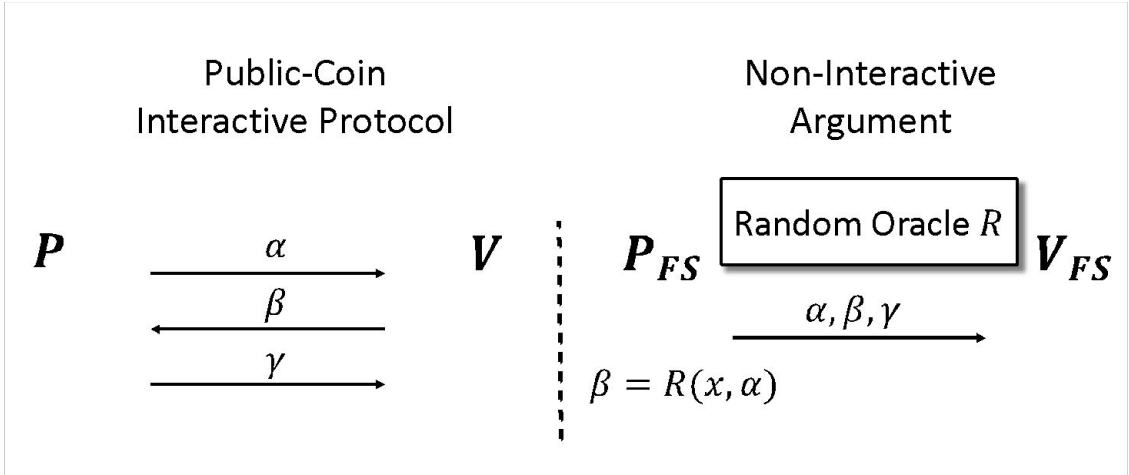
Concrete Security of SNARKs derived from Fiat-Shamir

Background: Fiat-Shamir Transformation



- Grinding attack on Fiat-Shamir:
  - P_{FS} iterates over first-messages α until it finds one such that $R(x, \alpha)$ is "lucky"
  - Suppose you apply Fiat-Shamir to an interactive protocol with, say, 80 bits of statistical security (soundness error 2^{-80}).
  - With 2^b hash evaluations, grinding attack will succeed with probability 2^{-80+b}.
  - E.g., with 2^{50} hashes, can successfully attack with probability 2^{-30}.

Background: Fiat-Shamir Transformation



- Grinding attack on Fiat-Shamir:
  - P_{FS} iterates over first-messages α until it finds one such that $R(x, \alpha)$ is "lucky"
  - Suppose you apply Fiat-Shamir to an interactive protocol with 80 bits of statistical security (soundness error 2^{-80}).
  - With 2^b hash evaluations, grinding attack will succeed with probability 2^{-80+b}.
  - E.g., with 2^{70} hashes, successfully attack with probability about 2^{-10}.

More discussion

- Grinding attack on Fiat-Shamir at "80 bits of security";
  - With 2^{70} hashes, can successfully attack with probability about $\mathbf{2^{-10}}$.
- Comparison:
  - For a collision-resistant hash function (CRHF) configured to 80 bits of security, the fastest collision-finding procedure should be a **birthday attack.**
    - With 2^k hash evaluations, finds a collision with a probability of only 2^{2k-160}.
    - For example, 2^{70} hash evaluations will yield a collision with a probability of $\mathbf{2^{-20}}$.
  - Algorithms for computing discrete logarithms are similar to birthday attacks.
    - E.g., For a group with 100 bits of security, 2^{70} group operations will yield a specified discrete logarithm with a probability of 2^{-20}.

Concrete Security of Orion

Concrete Security Part I: Known Attacks

- Orion's implementation works over a 122-bit field.
  - The statistical soundness error of the interactive protocol is less than 120 bits.
  - Perhaps below ~100 bits.
- Implications for the SNARK obtained via Fiat-Shamir:
  - With about 2^{60} hashes, **can** attack with probability about 2^{-60}.
    - Perhaps as high as 2^{-40} or more.
  - Compare to Brakedown's code generation step:
    - Failure probability at most 2^{-100}.
    - Code generated once for all time, so no opportunity for "grinding attack".

Concrete Security Part I: Known Attacks

- Orion's implementation works over a 122-bit field.
  - The statistical soundness error of the interactive protocol is less than 120 bits.
  - Perhaps below ~100 bits.
- Implications for the SNARK obtained via Fiat-Shamir:
  - With about 2^{60} hashes, **can** attack with probability about 2^{-60}.
    - Perhaps as high as 2^{-40} or more.
  - Compare to Brakedown's code generation step:
    - Failure probability at most 2^{-100}.
    - Code generated once for all time, so no opportunity for "grinding attack".

Concrete Security Part I: Known Attacks

- Orion's implementation works over a 122-bit field.
  - The statistical soundness error of the interactive protocol is less than 120 bits.
  - Perhaps below ~100 bits.
- Implications for the SNARK obtained via Fiat-Shamir:
  - With about 2^{60} hashes, **can** attack with probability about 2^{-60}.
    - Perhaps as high as 2^{-40} or more.
  - Compare to Brakedown's code generation step:
    - Failure probability at most 2^{-100}.
    - Code generated once for all time, so no opportunity for "grinding attack".
    - Even if the code fails to have requisite minimum distance, there is not necessarily an attack.

Concrete Security Part 2: Provable vs. Conjectured

- Orion's claims 128 bits of statistical security before applying Fiat-Shamir.
  - This may hold, but known analyses do not justify this.
  - They only justify $\frac{128}{3} \approx 43$ bits of security.
  - **Implicitly, the Orion paper is conjecturing that known attacks are optimal.**
    - This conjecture is not stated in the paper.
- If Brakedown used the same method to determine the number of column openings, its proof size would fall by a factor of about $\sqrt{3} \approx 1.7$.
- Orion's codebase configures FRI within Virgo to offer only about 80 bits of provable security (33 "FRI verifier queries" using a Reed-Sol

Concrete Security Part 2: Provable vs. Conjectured

- Orion's claims 128 bits of statistical security before applying Fiat-Shamir.
  - This may hold, but known analyses do not justify this.
  - They only justify $\frac{128}{3} \approx 43$ bits of security.
  - **Implicitly, the Orion paper is conjecturing that known attacks are optimal.**
    - This conjecture is not stated in the paper.
- If Brakedown used the same assumption, its proof size would fall by a factor of about $\sqrt{3} \approx 1.7$.
- Orion's codebase configures FRI within Virgo to offer only about 80 bits of provable security (33 "FRI verifier queries" using a Reed-Sol

Takeaways for researchers

- Clearly state what assumptions/conjectures security is based on.
  - It's okay if your conjecture is:

    "Known attacks on the statistical security of my interactive protocol are optimal"
  - But say so!

- When comparing the performance of two related systems, give them both "access" to the same conjectures.

- There's a difference between statistical and computational "bits of security".

Takeaways for researchers

- Clearly state what assumptions/conjectures security is based on.
  - It's okay if your conjecture is:

    "Known attacks on the statistical security of my interactive protocol are optimal"
  - But say so!

- When comparing the performance of two related systems, give them both "access" to the same conjectures.

- Performance point: Stop calling multi-exponentiations linear-time.
  - Naively implemented, size-N multi-exponentiation takes $O(N \lambda)$ group operations.
  - With Pippenger's algorithm, falls to $O(N \lambda / \log(N))$. Still super-linear.

Concrete security of deployed FRI-based SNARKs

Examples

1. StarkWare runs at **80 bits** of conjectured statistical soundness before applying Fiat-Shamir.
   - There is a **known attack on the Fiat-Shamired protocol** that performs 2^x hashes and succeeds with probability about 2^{-80+x}.
   - The attack produces a convincing proof of **any desired false statement**.
   - 2^{80} hashes $\Longrightarrow$ success probability close to 1.
     - Over 100 trillion times faster than the community reference recommends any known attack to be.
   - 2^{70} hashes $\Longrightarrow$ success probability about 2^{-10}.
   - The proved statistical security before applying Fiat-Shamir is only about 54 bits.
2. Plonky2 runs at 100 bits rather than 80.
   - "FRI verifier queries" using a Reed-Sol

Examples

1. StarkWare runs at **80 bits** of conjectured statistical soundness before applying Fiat-Shamir.
   - There is a **known attack on the Fiat-Shamired protocol** that performs 2^x hashes and succeeds with probability about 2^{-80+x}.
   - The attack produces a convincing proof of **any desired false statement**.
   - 2^{80} hashes $\Longrightarrow$ success probability close to 1.
     - Over 100 trillion times faster than the community reference recommends for known attacks.
   - 2^{70} hashes $\Longrightarrow$ success probability about 2^{-10}.
   - The **proved** statistical security before applying Fiat-Shamir is only about 54 bits.
2. Plonky2 runs at 100 bits rather than 80.
   - "FRI verifier queries" using a Reed-Sol

Examples

1. StarkWare runs at **80 bits** of conjectured statistical soundness before applying Fiat-Shamir.
   - There is a **known attack on the Fiat-Shamired protocol** that performs 2^x hashes and succeeds with probability about 2^{-80+x}.
   - The attack produces a convincing proof of **any desired false statement**.
   - 2^{80} hashes $\Longrightarrow$ success probability close to 1.
     - Over 100 trillion times faster than the community reference recommends for known attacks.
   - 2^{70} hashes $\Longrightarrow$ success probability about 2^{-10}.
   - The **proved** statistical security before applying Fiat-Shamir is only about 54 bits.
2. Plonky2 runs at **100 bits** rather than 80.
   - "FRI verifier queries" using a Reed-Sol

How many hashes are feasible?

1. In January 2020, the cost of computing just shy of 2^{64} SHA-1 evaluations using GPUs was $45,000.
   - This puts **2^{70}** hashes at about $3,000,000.
   - Likely much less today, post-Ethereum-merge.
2. Today, the bitcoin network performs **2^{80}** SHA-256 hashes every 1.25 hours.
   - At current prices, those hashes typically earn **less than $1 million** worth of block rewards.

How many hashes are feasible?

1. In January 2020, the cost of computing just shy of 2^{64} SHA-1 evaluations using GPUs was $45,000.
   - This puts **2^{70}** hashes at about $3,000,000.
   - Likely much less today, post-Ethereum-merge.
2. Today, the bitcoin network performs **2^{80}** SHA-256 hashes every 1.25 hours.
   - At current prices, those hashes typically earn **less than $1 million** worth of block rewards.

- Profiting off an attack on a rollup does not require spending stolen funds.
  - Attacker can short the rollups token, then attack to cause token value to fall.

Proposals to resolve the disconnect

Proposed actions

- Update the community reference to explicitly discuss SNARK deployments, not only benchmarking.
- Encourage projects that decide on lower-than-recommended security to explain why they think their security level suffices.
- Talk about the disconnect. Publicly.
  - The cryptography is much too advanced for end users to assess.
  - Especially given the community's rhetoric:
    - "Rollups inherit the security of L1"
    - "Future-proof"
    - Etc.
  - Yet end users are the ones who ultimately absorb the risk of weak security.

StarkNet Terms of Use

- IMPORTANT NOTICE: DUE TO THE “ALPHA” STATE OF THE NETWORK, **THE NETWORK MAY INCLUDE SECURITY ISSUES, BUGS AND OTHER SOFTWARE ERRORS.** SUCH ISSUES, BUGS AND ERRORS MAY RESULT IN INCORRECT OR UNINTENDED FUNCTIONING OF THE NETWORK, OR **MAY BE EXPLOITED BY MALICIOUS ACTORS**, EACH OF WHICH MAY RESULT IN THE LOSS OR IRRETRIEVABLE MISPLACEMENT OF ALL OR A PORTION OF ANY OF YOUR FUNDS AND/OR DIGITAL ASSETS…

- STARKWARE SHALL NOT BE RESPONSIBLE OR LIABLE IN ANY MANNER, AND **YOU HEREBY RELEASE STARKWARE FROM ANY AND ALL LIABILITY OR RESPONSIBILITY, FOR ANY LOSS OF FUNDS AND/OR DIGITAL ASSETS AND/OR PRIVATE KEYS DUE TO THE FOREGOING FACTORS.”**

- … advances in code cracking, or technical advances such as the development of quantum computers, could present risks to the Network, which could result in the theft or loss of assets stored or transferred via the Network”

Squash: Sum-check-based SNARKs for AIR and PlonKish IR

Sum-Check-Based Backends

- SNARKs based on the sum-check protocol of [LFKN90] have the fastest prover.
  - Linearly many field operations, plus the cost of the polynomial commitment.
- Verification costs can be logarithmic.
- Previous sum-check based SNARKs were described for arithmetic circuits and R1CS (e.g., Spartan).
- Hyperplonk recently described one for Plonkish IR.

Squash [Setty-Thaler-Wahby, upcoming]

- We give simple adaptations of Spartan's polynomial IOP to handle AIR and Plonkish IR.
- Plug in whatever polynomial commitment you want.
  - Brakedown-commitment: gives a field-agnostic, linear-time SNARK for AIR (square-root size proofs).
  - Orion-commitment: polylogarithmic-sized proofs instead of square root.
  - First such SNARKs for AIR with these properties.

Thank you!

Mechanics of the Brakedown Polynomial Commitment

[AHIV17, BCGGHJ17, BCG20, GLSTW21]

- To commit to a polynomial p with N coefficients, P views its coefficients as a $N^{1/2} \times N^{1/2}$ matrix.
  - Encodes each row with our error-correcting code, sends a Merkle-commitment to the resulting "encoded" matrix.
- V "tests" the committed matrix to make sure its rows are codewords.
  - Asks P for a random linear combination of the rows.
  - Confirms P's response v is a codeword.
  - Checks v for consistency with the Merkle-committed matrix.
    - Picks $O(\lambda)$ columns of the committed matrix and demands that P open them.
      - Hidden constant is large. Concretely, about 6000 column openings at 128 bits of security.
      - For each opened column i, checks that the appropriate linear combination of its entries equals v_i.

- To commit to a polynomial p with N coefficients, P views its coefficients as a $N^{1/2} \times N^{1/2}$ matrix.
  - Encodes each row with our error-correcting code, sends a Merkle-commitment to the resulting "encoded" matrix.
- V "tests" the committed matrix to make sure its rows are codewords.
  - Asks P for a random linear combination of the rows.
  - Confirms P's response v is a codeword.
  - Checks v for consistency with the Merkle-committed matrix.
    - Picks $O(\lambda)$ columns of the committed matrix and demands that P open them.
    - Hidden constant is large. Concretely, about 6000 column openings at 128 bits of security.
    - For each opened column i, checks that the appropriate linear combination of its entries equals v_i.

- To commit to a polynomial p with N coefficients, P views its coefficients as a $N^{1/2} \times N^{1/2}$ matrix.
  - Encodes each row with our error-correcting code, sends a Merkle-commitment to the resulting "encoded" matrix.
- V "tests" the committed matrix to make sure its rows are codewords.
  - Asks P for a random linear combination of the rows.
  - Confirms P's response v is a codeword.
  - Checks v for consistency with the Merkle-committed matrix.
    - Picks $O(\lambda)$ columns of the committed matrix and demands that P open them.
    - Hidden constant is large. Concretely, about 6000 column openings at 128 bits of security.
    - For each opened column i, checks that the appropriate linear combination of its entries equals v_i.