Verifiable ASICs: trustworthy hardware with untrusted components

Riad S. Wahby, Max Howald, Siddharth Garg, abhi shelat, and Michael Walfish

Stanford University
New York University
The Cooper Union
The University of Virginia

April 8, 2016
You (probably) shouldn’t trust your hardware...
You (probably) shouldn’t trust your hardware...
You (probably) shouldn’t trust your hardware...
You (probably) shouldn’t trust your hardware... 

...because fabs sometimes make mistakes.
You (probably) shouldn’t trust your hardware... because fabs sometimes make “mistakes”

What’s a chip designer to do?

- Post-fab testing
- Hardware obfuscation
- Trusted manufacturer

What’s a chip designer to do?

- Post-fab testing
- Hardware obfuscation
- Trusted manufacturer

What’s a chip designer to do?

- Post-fab testing
- Hardware obfuscation
- Trusted manufacturer
  - but a fab is expensive and hard to build...

What’s a chip designer to do?

- Post-fab testing

- Hardware obfuscation

- Trusted manufacturer
  - but a fab is expensive and hard to build...
  - ...so trusted fab might have $10^8 \times$ worse performance!

Roadmap

1. Problem statement: verifiable ASICs
2. Probabilistic proof systems, briefly
3. Zebra: a system for verifiable ASICs
4. Implementation and evaluation
Roadmap

1. Problem statement: verifiable ASICs
2. Probabilistic proof systems, briefly
3. Zebra: a system for verifiable ASICs
4. Implementation and evaluation
Problem statement: verifiable ASICs

Principal

\[ \Psi \rightarrow \text{specs} \]
for \( P, V \)

▶ \( P \) is efficient, but can deviate arbitrarily from the protocol
▶ Honest \( P \) always convinces \( V \) that \( y = \Psi(x) \)
▶ \( V \) must catch dishonest \( P \) except with negligible probability
▶ \( P \) cannot attack or disable \( V \), or communicate with outside world (see paper for more discussion)

▶ Goal: \( V \) and \( P \) together should outperform \( \Psi \) executed in trusted substrate
Problem statement: verifiable ASICs

Suppliers
(foundry, processor vendor, etc.)

Principal
\(\psi \rightarrow \text{specs for } \mathcal{P}, \mathcal{V}\)

Foundry

- \(\mathcal{P}\) is efficient, but can deviate arbitrarily from the protocol
- Honest \(\mathcal{P}\) always convinces \(\mathcal{V}\) that \(y = \psi(x)\)
- \(\mathcal{V}\) must catch dishonest \(\mathcal{P}\) except with negligible probability
- \(\mathcal{P}\) cannot attack or disable \(\mathcal{V}\), or communicate with the outside world (see paper for more discussion)

Goal:
\(\mathcal{V}\) and \(\mathcal{P}\) together should outperform \(\psi\) executed in trusted substrate
Problem statement: verifiable ASICs

Principal
\( \Psi \rightarrow \text{specs for } \mathcal{P}, \mathcal{V} \)

Supplier
(foundry, processor vendor, etc.)

Integrator

Foundry

\( \mathcal{V} \)

\( \mathcal{P} \)

\( \forall \) is efficient, but can deviate arbitrarily from the protocol

Honest \( \mathcal{P} \) always convinces \( \mathcal{V} \) that \( y = \Psi(x) \)

\( \mathcal{V} \) must catch dishonest \( \mathcal{P} \) except with negligible probability

\( \mathcal{P} \) cannot attack or disable \( \mathcal{V} \), or communicate with outside world (see paper for more discussion)

Goal:
\( \mathcal{V} \) and \( \mathcal{P} \) together should outperform \( \Psi \) executed in trusted substrate
Problem statement: verifiable ASICs

- **Principal**: \( \psi \rightarrow \text{specs for } \mathcal{P}, \mathcal{V} \)
- **Supplier**: (foundry, processor vendor, etc.)
- **Foundry**
- **Integrator**
- **Operator**: \( \mathcal{V} \) and \( \mathcal{P} \)

- **Goal**: \( \mathcal{V} \) and \( \mathcal{P} \) together should outperform \( \psi \) executed in trusted substrate

- **Properties**:
  - \( \mathcal{P} \) is efficient, but can deviate arbitrarily from the protocol
  - Honest \( \mathcal{P} \) always convinces \( \mathcal{V} \) that \( y = \psi(x) \)
  - \( \mathcal{V} \) must catch dishonest \( \mathcal{P} \) except with negligible probability
  - \( \mathcal{P} \) cannot attack or disable \( \mathcal{V} \), or communicate with outside world (see paper for more discussion)
Problem statement: verifiable ASICs

Operator

\( \nu \)

\( \mathcal{P} \)

\( \nu \) is efficient, but can deviate arbitrarily from the protocol.

Honest \( \nu \) always convinces \( \mathcal{P} \) that \( y = \Psi(x) \).

\( \nu \) must catch dishonest \( \mathcal{P} \) except with negligible probability.

\( \mathcal{P} \) cannot attack or disable \( \nu \), or communicate with the outside world (see paper for more discussion).

Goal: \( \nu \) and \( \mathcal{P} \) together should outperform \( \Psi \) executed in trusted substrate.
Problem statement: verifiable ASICs

\[ V \] and \( P \) together should outperform \( \Psi \) executed in trusted substrate.
Problem statement: verifiable ASICs

- $P$ is efficient, but can deviate arbitrarily from the protocol.
- Honest $P$ always convinces $V$ that $y = \Psi(x)$.
- $V$ must catch dishonest $P$ except with negligible probability.
- $P$ cannot attack or disable $V$, or communicate with the outside world (see paper for more discussion).

Goal: $V$ and $P$ together should outperform $\Psi$ executed in trusted substrate.
Problem statement: verifiable ASICs

Operator $\mathcal{V}$ and $\mathcal{P}$ together should outperform $\Psi$ executed in trusted substrate.

$\mathcal{P}$ is efficient, but can deviate arbitrarily from the protocol.

$\mathcal{V}$ must catch dishonest $\mathcal{P}$ except with negligible probability.

$\mathcal{P}$ cannot attack or disable $\mathcal{V}$, or communicate with the outside world (see paper for more discussion).
Problem statement: verifiable ASICs

- $\mathcal{P}$ is efficient, but can deviate arbitrarily from the protocol.
Problem statement: verifiable ASICs

- $\mathcal{P}$ is efficient, but can deviate arbitrarily from the protocol
- Honest $\mathcal{P}$ always convinces $\mathcal{V}$ that $y = \Psi(x)$
Problem statement: verifiable ASICs

- $\mathcal{P}$ is efficient, but can deviate arbitrarily from the protocol
- Honest $\mathcal{P}$ always convinces $\mathcal{V}$ that $y = \Psi(x)$
- $\mathcal{V}$ must catch dishonest $\mathcal{P}$ except with negligible probability
Problem statement: verifiable ASICs

- \( \mathcal{P} \) is efficient, but can deviate arbitrarily from the protocol
- Honest \( \mathcal{P} \) always convinces \( \mathcal{V} \) that \( y = \Psi(x) \)
- \( \mathcal{V} \) must catch dishonest \( \mathcal{P} \) except with negligible probability
- \( \mathcal{P} \) cannot attack or disable \( \mathcal{V} \), or communicate with outside world (see paper for more discussion)
Problem statement: verifiable ASICs

- $\mathcal{P}$ is efficient, but can deviate arbitrarily from the protocol
- Honest $\mathcal{P}$ always convinces $\mathcal{V}$ that $y = \Psi(x)$
- $\mathcal{V}$ must catch dishonest $\mathcal{P}$ except with negligible probability
- $\mathcal{P}$ cannot attack or disable $\mathcal{V}$, or communicate with outside world (see paper for more discussion)

Goal: $\mathcal{V}$ and $\mathcal{P}$ together should outperform $\Psi$ executed in trusted substrate
Roadmap

1. Problem statement: verifiable ASICs

2. Probabilistic proof systems, briefly

3. Zebra: a system for verifiable ASICs

4. Implementation and evaluation
Probabilistic proof systems, briefly

A weak verifier checks the work of a powerful prover

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak *verifier* checks the work of a powerful *prover*

Idea: checking *proof* should be *easier* for verifier than executing program

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak \textit{verifier} checks the work of a powerful \textit{prover}

Recent work is in three strands:

▷ Interactive arguments
  ▷ [Pepper12, Ginger12, Zaatar13]

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak verifier checks the work of a powerful prover

Recent work is in three strands:

- Interactive arguments
  - [Pepper12, Ginger12, Zaatar13]

- Non-interactive arguments (SNARKs)
  - [PGHR13, BCGTV13, BCTV14]

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak \textit{verifier} checks the work of a powerful \textit{prover}

Recent work is in three strands:

- Interactive arguments
  - [Pepper12, Ginger12, Zaatar13]

- Non-interactive arguments (SNARKs)
  - [PGHR13, BCGTV13, BCTV14]

- Interactive proofs
  - [CMT12, TRMP12, Allspice13, Tha13]

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak *verifier* checks the work of a powerful *prover*

Recent work is in three strands:

- **Interactive arguments**
  - [Pepper12, Ginger12, Zaatar13]
  - Low round complexity
  - Mild cryptographic assumptions

- **Non-interactive arguments (SNARKs)**
  - [PGHR13, BCGTV13, BCTV14]

- **Interactive proofs**
  - [CMT12, TRMP12, Allspice13, Tha13]

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak *verifier* checks the work of a powerful *prover*

Recent work is in three strands:

- **Interactive arguments**
  - [Pepper12, Ginger12, Zaatar13]
  - Low round complexity
  - Mild cryptographic assumptions

- **Non-interactive arguments (SNARKs)**
  - [PGHR13, BCGTV13, BCTV14]
  - Public verifiability, zero knowledge
    - Non-falsifiable cryptographic assumptions [GW10]

- **Interactive proofs**
  - [CMT12, TRMP12, Allspice13, Tha13]

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak *verifier* checks the work of a powerful *prover*

Recent work is in three strands:

- **Interactive arguments**
  - [Pepper12, Ginger12, Zaatar13]
  - Low round complexity
  - Mild cryptographic assumptions

- **Non-interactive arguments (SNARKs)**
  - [PGHR13, BCGTV13, BCTV14]
  - Public verifiability, zero knowledge
  - Non-falsifiable cryptographic assumptions [GW10]

- **Interactive proofs**
  - [CMT12, TRMP12, Allspice13, Tha13]
  - Simple and efficient prover and verifier
  - Information theoretic guarantees (no crypto)

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak *verifier* checks the work of a powerful *prover*

For all systems, expressiveness is somewhat limited:

- **Arguments (interactive & non-interactive)**
  - Computation must be expressed as an arithmetic circuit

- **Interactive proofs**
  - Computation must be expressed as an arithmetic circuit

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak *verifier* checks the work of a powerful *prover*

For all systems, expressiveness is somewhat limited:

- **Arguments (interactive & non-interactive)**
  - Computation must be expressed as an arithmetic circuit

  generalized boolean circuit over $\mathbb{F}_p$

  $\land \rightarrow \times \quad \lor \rightarrow +$

- **Interactive proofs**
  - Computation must be expressed as an arithmetic circuit

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak verifier checks the work of a powerful prover

For all systems, expressiveness is somewhat limited:

- **Arguments (interactive & non-interactive)**
  - Computation must be expressed as an arithmetic circuit
  - Arithmetic circuit can have arbitrary connectivity, shape

- **Interactive proofs**
  - Computation must be expressed as an arithmetic circuit
  - Arithmetic circuit must be layered, depth \(\ll\) width

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak verifier checks the work of a powerful prover

For all systems, expressiveness is somewhat limited:

▶ Arguments (interactive & non-interactive)
  - Computation must be expressed as an arithmetic circuit
  - Arithmetic circuit can have arbitrary connectivity, shape

▶ Interactive proofs
  - Computation must be expressed as an arithmetic circuit
  - Arithmetic circuit must be layered, depth $\ll$ width

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Probabilistic proof systems, briefly

A weak **verifier** checks the work of a powerful **prover**

For all systems, expressiveness is somewhat limited:

- **Arguments (interactive & non-interactive)**
  - Computation must be expressed as an arithmetic circuit
  + Arithmetic circuit can have arbitrary connectivity, shape

- **Interactive proofs**
  - Computation must be expressed in an arithmetic circuit
  - Arithmetic circuit must be layered, \( \text{depth} \ll \text{width} \)

[Walfish and Blumberg. “Verifying computations without reexecuting them: from theoretical possibility to near practicality.” CACM, Feb. 2015.]
Roadmap

1. Problem statement: verifiable ASICs

2. Probabilistic proof systems, briefly

3. Zebra: a system for verifiable ASICs

4. Implementation and evaluation
Zebra’s starting point: Interactive proofs

Zebra is an IP-based protocol [CMT12, Allspice13]

(We discuss why not arguments in the paper)
Zebra’s starting point: Interactive proofs

1. $V$ sends inputs
2. $P$ evaluates circuit, returns output
3. $V$ constructs polynomial relating $y$ to values of last layer’s input wires
4. $V$ cross-examines $P$
5. $V$ iterates
6. $V$ checks consistency with the inputs
Zebra’s starting point: Interactive proofs

1. $V$ sends inputs

2. $P$ evaluates circuit, returns output

3. $V$ constructs polynomial relating $y$ to values of last layer’s input wires

4. $V$ cross-examines $P$

5. $V$ iterates

6. $V$ checks consistency with the inputs
Zebra’s starting point: Interactive proofs

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output $y$
Zebra’s starting point: Interactive proofs

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output $y$
Zebra’s starting point: Interactive proofs

1. $\mathcal{V}$ sends inputs

2. $\mathcal{P}$ evaluates circuit, returns output $y$
Zebra’s starting point: Interactive proofs

1. $V$ sends inputs
2. $P$ evaluates circuit, returns output $y$
Zebra’s starting point: Interactive proofs

1. $V$ sends inputs
2. $P$ evaluates circuit, returns output $y$
3. $V$ constructs polynomial relating $y$ to values of last layer’s input wires
Zebra’s starting point: Interactive proofs

1. $V$ sends inputs
2. $P$ evaluates circuit, returns output $y$
3. $V$ constructs polynomial relating $y$ to values of last layer’s input wires
4. $V$ cross-examines $P$
Zebra’s starting point: Interactive proofs

1. \( \mathcal{V} \) sends inputs
2. \( \mathcal{P} \) evaluates circuit, returns output \( y \)
3. \( \mathcal{V} \) constructs polynomial relating \( y \) to values of last layer’s input wires
4. \( \mathcal{V} \) cross-examines \( \mathcal{P} \), ends up with claim about second-last layer
5. \( \mathcal{V} \) iterates
6. \( \mathcal{V} \) checks consistency with the inputs
Zebra’s starting point: Interactive proofs

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output $y$
3. $\mathcal{V}$ constructs polynomial relating $y$ to values of last layer’s input wires
4. $\mathcal{V}$ cross-examines $\mathcal{P}$, ends up with claim about second-last layer
5. $\mathcal{V}$ iterates
Zebra’s starting point: Interactive proofs

1. $\mathcal{V}$ sends inputs

2. $\mathcal{P}$ evaluates circuit, returns output $y$

3. $\mathcal{V}$ constructs polynomial relating $y$ to values of last layer’s input wires

4. $\mathcal{V}$ cross-examines $\mathcal{P}$, ends up with claim about second-last layer

5. $\mathcal{V}$ iterates
Zebra’s starting point: Interactive proofs

1. $V$ sends inputs
2. $P$ evaluates circuit, returns output $y$
3. $V$ constructs polynomial relating $y$ to values of last layer’s input wires
4. $V$ cross-examines $P$, ends up with claim about second-last layer
5. $V$ iterates
1. $V$ sends inputs

2. $P$ evaluates circuit, returns output $y$

3. $V$ constructs polynomial relating $y$ to values of last layer’s input wires

4. $V$ cross-examines $P$, ends up with claim about second-last layer

5. $V$ iterates, ends up with claim about inputs
Zebra’s starting point: Interactive proofs

1. $V$ sends inputs

2. $P$ evaluates circuit, returns output $y$

3. $V$ constructs polynomial relating $y$ to values of last layer’s input wires

4. $V$ cross-examines $P$, ends up with claim about second-last layer

5. $V$ iterates, ends up with claim about inputs

6. $V$ checks consistency with the inputs
Zebra’s starting point: Interactive proofs

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates circuit, returns output $y$
3. $\mathcal{V}$ constructs polynomial relating $y$ to values of last layer’s input wires
4. $\mathcal{V}$ cross-examines $\mathcal{P}$, ends up with claim about second-last layer
5. $\mathcal{V}$ iterates, ends up with claim about inputs
6. $\mathcal{V}$ checks consistency with the inputs

$\mathcal{V}$'s work is $\approx O(\text{depth} \cdot \log \text{width})$
$\forall$ questions $\mathcal{P}$ about $\Psi(x_1)$’s output layer.
Pipelined proving in Zebra

\( \mathcal{V} \) questions \( \mathcal{P} \) about \( \Psi(x_1) \)'s output layer.

Simultaneously, \( \mathcal{P} \) returns \( \Psi(x_2) \).
Pipelined proving in Zebra

\( \forall \) questions \( \mathcal{P} \) about \( \Psi(x_1) \)'s next layer
Pipelined proving in Zebra

\( \forall \) questions \( \mathcal{P} \) about \( \Psi(x_1)'s \) next layer, and \( \Psi(x_2)'s \) output layer.
Pipelined proving in Zebra

\( \mathcal{N} \) questions \( \mathcal{P} \) about \( \psi(x_1) \)'s next layer, and \( \psi(x_2) \)'s output layer.

Meanwhile, \( \mathcal{P} \) returns \( \psi(x_3) \).
Pipelined proving in Zebra

This process continues until the pipeline is full.
Pipelined proving in Zebra

This process continues until the pipeline is full.
Pipelined proving in Zebra

This process continues until the pipeline is full.

\( \nu \) and \( \mathcal{P} \) can complete one proof in each time step.
Other hardware optimizations

Input \((x)\)  
Output \((y)\)  
prove  
prove  
prove  
Sub-prover, layer \(d - 1\)  
prove  
\(+\, \times\, \ominus\, \oplus\)  
Sub-prover, layer \(1\)  
prove  
\(+\, \ominus\, +\, \times\)  
Sub-prover, layer \(0\)  
prove  
\(\times\, \times\, \ominus\, \ominus\)  

▶ P’s work is mainly local to each gate  
▶ P’s design leverages this with gate prover circuits  
▶ Gate provers reuse work from previous rounds by maintaining local state  
▶ Further low-level and protocol optimizations (see paper)
Other hardware optimizations

- $\mathcal{P}$’s work is mainly local to each gate
Other hardware optimizations

- $\mathcal{P}$’s work is mainly local to each gate
- $\mathcal{P}$’s design leverages this with gate prover circuits
Other hardware optimizations

- $\mathcal{P}$’s work is mainly local to each gate
- $\mathcal{P}$’s design leverages this with gate prover circuits
- Gate provers reuse work from previous rounds by maintaining local state
Other hardware optimizations

- $\mathcal{P}$'s work is mainly local to each gate
- $\mathcal{P}$'s design leverages this with gate prover circuits
- Gate provers reuse work from previous rounds by maintaining local state
- Further low-level and protocol optimizations (see paper)
Architectural challenges and limitations

- Interaction between $\mathcal{V}$ and $\mathcal{P}$ requires a lot of bandwidth
Architectural challenges and limitations

- Interaction between $\mathcal{V}$ and $\mathcal{P}$ requires a lot of bandwidth
  $\Rightarrow$ Zebra uses 3D integration
Architectural challenges and limitations

- Interaction between $\mathcal{V}$ and $\mathcal{P}$ requires a lot of bandwidth
  $\Rightarrow$ Zebra uses *3D integration*

- IP protocol requires precomputation for most $\Psi$
  [Allspice13]
Architectural challenges and limitations

- Interaction between $\mathcal{V}$ and $\mathcal{P}$ requires a lot of bandwidth
  $\Rightarrow$ Zebra uses 3D integration

- IP protocol requires precomputation for most $\Psi$
  [Allspice13]
  $\Rightarrow$ Zebra amortizes precomputations over many $\mathcal{V}$-$\mathcal{P}$ pairs
Architectural challenges and limitations

- Interaction between $\mathcal{V}$ and $\mathcal{P}$ requires a lot of bandwidth
  $\Rightarrow$ Zebra uses 3D integration

- IP protocol requires precomputation for most $\Psi$
  \[\text{[Allspice13]}\]
  $\Rightarrow$ Zebra amortizes precomputations over many $\mathcal{V}$-$\mathcal{P}$ pairs

- Several other details (see paper)
Roadmap

1. Problem statement: verifiable ASICs

2. Probabilistic proof systems, briefly

3. Zebra: a system for verifiable ASICs

4. Implementation and evaluation
Evaluation

How does Zebra perform on computations of “real world” interest?

- Number theoretic transform
- Curve25519 point multiplication
Evaluation

How does Zebra perform on computations of “real world” interest?

- Number theoretic transform
- Curve25519 point multiplication

**Goal**: $\mathcal{V}$ and $\mathcal{P}$ together should outperform $\Psi$ executed in trusted substrate
Implementation and method

Zebra’s implementation includes

- a compiler that produces synthesizable Verilog for $\mathcal{P}$
- two $\mathcal{V}$ implementations (software and Verilog)
- library to generate $\mathcal{V}$’s precomputations
- Verilog simulator extensions to support either hardware and software $\mathcal{V}$ interacting with $\mathcal{P}$ design

For our evaluation, we build a detailed cost model based on analysis, simulation results, and published chip designs (see paper)

Baseline: direct implementation of $\Psi$ in same technology as $\mathcal{V}$

Assumption: computation is efficient as an arithmetic circuit

Metric: energy required to execute $\Psi$
Implementation and method

Zebra’s implementation includes

- a compiler that produces synthesizable Verilog for $\mathcal{P}$
- two $\mathcal{V}$ implementations (software and Verilog)
- library to generate $\mathcal{V}$’s precomputations
- Verilog simulator extensions to support either hardware and software $\mathcal{V}$ interacting with $\mathcal{P}$ design

For our evaluation, we build a detailed cost model based on analysis, simulation results, and published chip designs (see paper)
Implementation and method

Zebra’s implementation includes

- a compiler that produces synthesizable Verilog for $\mathcal{P}$
- two $\mathcal{V}$ implementations (software and Verilog)
- library to generate $\mathcal{V}$’s precomputations
- Verilog simulator extensions to support either hardware and software $\mathcal{V}$ interacting with $\mathcal{P}$ design

For our evaluation, we build a detailed cost model based on analysis, simulation results, and published chip designs (see paper)

Baseline: direct implementation of $\Psi$ in same technology as $\mathcal{V}$

$\Rightarrow$ Assumption: computation is efficient as an arithmetic circuit
Implementation and method

Zebra’s implementation includes

- a compiler that produces synthesizable Verilog for $\mathcal{P}$
- two $\mathcal{V}$ implementations (software and Verilog)
- library to generate $\mathcal{V}$’s precomputations
- Verilog simulator extensions to support either hardware and software $\mathcal{V}$ interacting with $\mathcal{P}$ design

For our evaluation, we build a detailed cost model based on analysis, simulation results, and published chip designs (see paper)

Baseline: direct implementation of $\Psi$ in same technology as $\mathcal{V}$

⇒ Assumption: computation is efficient as an arithmetic circuit

Metric: energy required to execute $\Psi$
Number theoretic transform

Performance relative to native baseline (higher is better)

$\log_2(\text{NTT size})$

![Graph showing performance relative to native baseline.](image)
Curve25519 point multiplication

Performance relative to native baseline (higher is better)
Recap

Zebra enables verifiable ASICs.

- Untrusted $\mathcal{P}$ improves the performance of trusted $\mathcal{V}$
Recap

Zebra enables verifiable ASICs.

+ Untrusted $\mathcal{P}$ improves the performance of trusted $\mathcal{V}$
+ First built system in the probabilistic proof literature where total cost of $\mathcal{V} + \mathcal{P}$ is better than baseline
Recap

Zebra enables verifiable ASICs.

+ Untrusted $\mathcal{P}$ improves the performance of trusted $\mathcal{V}$
+ First built system in the probabilistic proof literature where total cost of $\mathcal{V} + \mathcal{P}$ is better than baseline
  – But this improvement is modest
Recap

Zebra enables verifiable ASICs.

+ Untrusted $P$ improves the performance of trusted $V$
+ First built system in the probabilistic proof literature where total cost of $V + P$ is better than baseline
  - But this improvement is modest,
  - and Zebra has limitations:
    does not apply to all computations
    precomputations must be amortized
    computation needs to be “big enough”
    needs large gap between trusted and untrusted technology

https://eprint.iacr.org/2015/1243
Recap

Zebra enables verifiable ASICs.

+ Untrusted $\mathcal{P}$ improves the performance of trusted $\mathcal{V}$
+ First built system in the probabilistic proof literature where total cost of $\mathcal{V} + \mathcal{P}$ is better than baseline
  - But this improvement is modest,
  - and Zebra has limitations:
    - does not apply to all computations
    - precomputations must be amortized
    - computation needs to be “big enough”
    - needs large gap between trusted and untrusted technology

Zebra is a first step—there are plenty of improvements to be made!

https://eprint.iacr.org/2015/1243