Zero-knowledge proofs security, in practice

JP Aumasson
@veorq

CSO @ taurushq.com

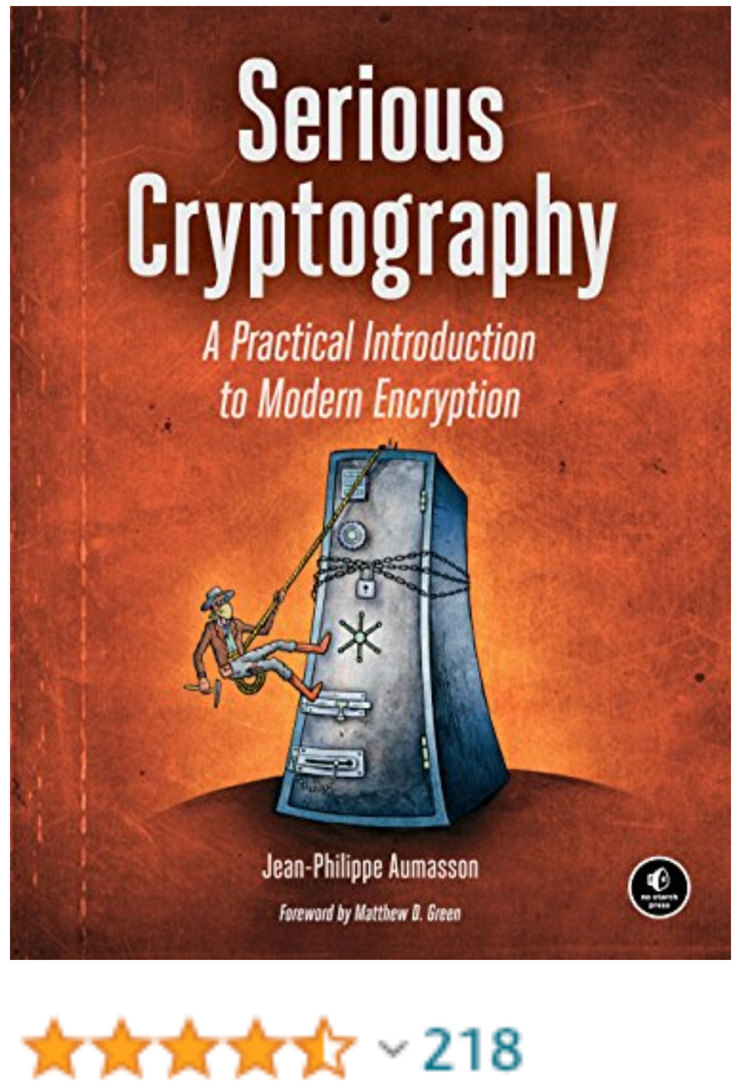
/me

Co-founder & CSO of a Swiss fintech (**Taurus**)

- High-assurance crypto custody tech https://taurushq.com
- Used by banks to protect and manage their BTC/ETH/etc.
- Running a regulated exchange https://t-dx.com

Cryptography and vulnerability research since ~2006

- Designed crypto in the Linux kernel, Bitcoin, etc. (SipHash, BLAKE2, BLAKE3)
- Wrote some books about crypto*graphy*



https://aumasson.jp. https://twitter.com/veorq

Zero-knowledge proof?

!= “zero-knowledge architecture” (a.k.a. “zero-trust”)

!= “zero-knowledge encryption” (marketing term for client-side encryption)

vixentael
Dec 28, 2017 · 7 min read · Listen

Zero Knowledge Architectures for Mobile Applications

WHAT IS ZERO-KNOWLEDGE ENCRYPTION, AND HOW DOES IT WORK?

WRITTEN BY MATT AHLGREN RESEARCHED BY WSR TEAM | JUNE 22, 2022 | IN CLOUD STORAGE, PASSWORD MANAGERS

Zero-knowledge encryption is arguably one of the **most secure ways of protecting your data.** In a nutshell, it means that cloud storage or backup providers know nothing (i.e. have “zero-knowledge”) about the data you store on their servers.

Protocolo de conocimiento cero

!= “zero-knowledge architecture” (a.k.a. “zero-trust”)

!= “zero-knowledge encryption” (marketing term for client-side encryption)

A class of **cryptography protocols**…

- Between a *prover* and a *verifier*
- Which can be *non-interactive*
- Known since the 1980s, only recently used in practice at scale (*zkSNARKS*)

The Knowledge Complexity of Interactive Proof-Systems

(Extended Abstract)

Shafi Goldwasser, MIT
Silvio Micali, MIT
Charles Rackoff, University of Toronto

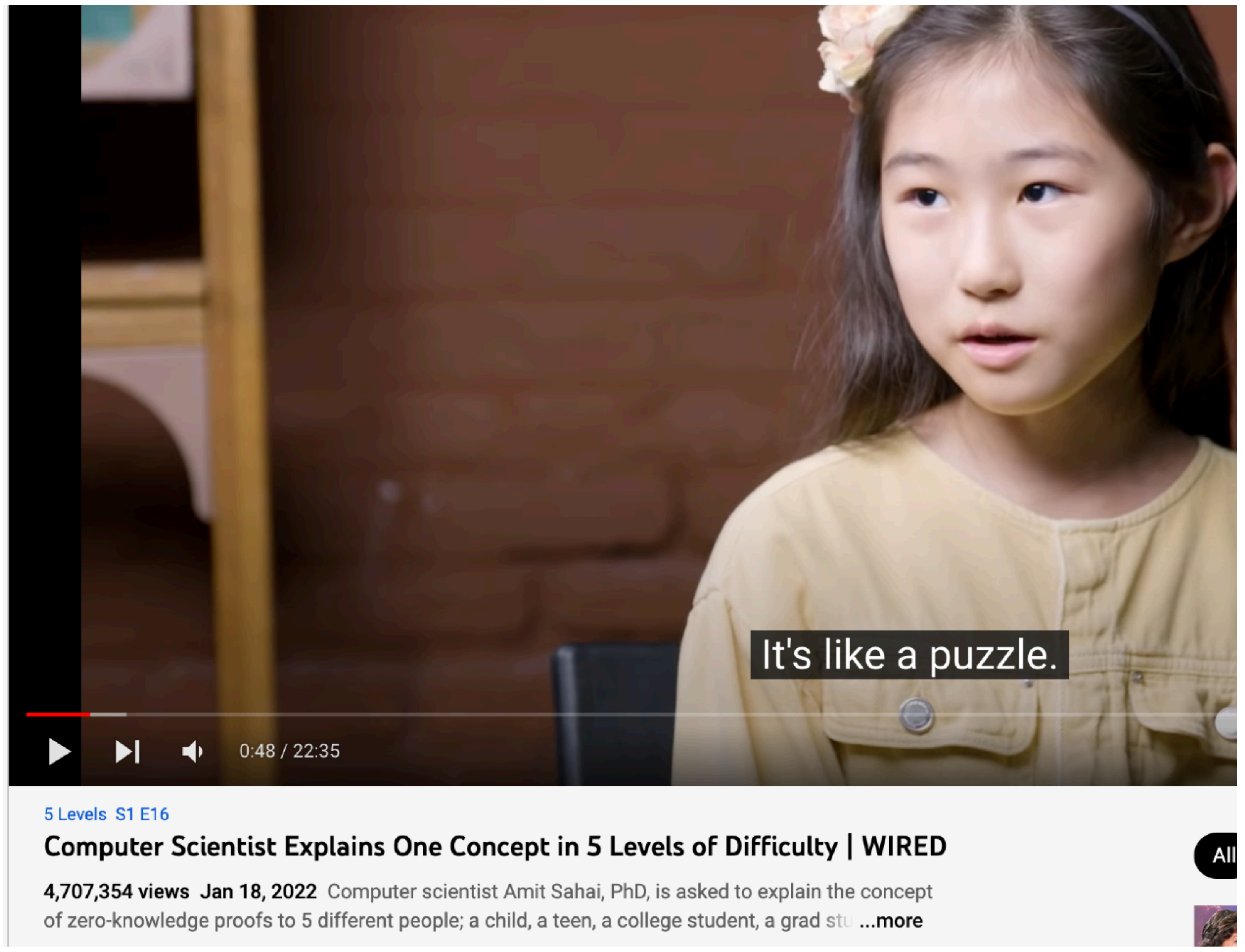
Protocolo de conocimiento cero

!= "zero-knowledge architecture" (a.k.a. "zero-trust")

!= "zero-knowledge encryption" (marketing term for client-side encryption)

A class of **cryptography protocols**…

- Between a *prover* and a *verifier*
- Which can be *non-interactive*
- Known since the 1980s, only recently used in practice at scale (*zkSNARKS*)



https://www.youtube.com/watch?v=fOGdb1CTu5c

The simplest ZK proof

Schnorr's proof of knowledge of discrete logarithm ($\boldsymbol{x}$ in $y = g^{\boldsymbol{x}} \bmod p$)

Probador

Verificador

Pick a random r, send $t = g^r \bmod p$ →

← Send a random c

Send $s = r + c\boldsymbol{x} \bmod p$ →

Verify that $g^s = t \times y^c$

It works because $g^s = g^{r + c\boldsymbol{x}} = g^r \times (g^{\boldsymbol{x}})^c = t \times y^c$

Zero-knowledge proofs applications

- **Privacy** of payments (à la Zcash and Monero), and of general computation (Aleo)
- **Scalability** – via "**ZK rollups**", preventing re-computation (though not always private)
- **Storage** proofs, as in Filecoin's proofs of spacetime
- **Mining**, as in Aleo's proofs of succinct work

> Our proof-of-concept system allows the Police to prove to the public that the DNA profile of a Presidential Candidate does not appear in the forensic DNA profile database maintained by the Police. The proof, which is generated by the Police, relies on no external trusted party, and reveals no further information about the contents of the database, nor about the candidate's profile. In particular, no DNA information is disclosed to any party outside the Police. The proof is shorter than the size of the DNA database, and verified faster than the time needed to examine that database naïvely.

https://eprint.iacr.org/2018/046

Vibrant ecosystem

Examples of major projects in the ZK space, many other initiatives and research groups

aleo.org

anoma.network

arkworks.rs

aztec.network

celo.org

protocol.ai

starkware.co

z.cash

This talk

Focus on **zkSNARKs**, a class of zero-knowledge proof systems

- *Fully succinct* = O(1) proof size and O(circuit size) verification time

Based on my experience looking for bugs in

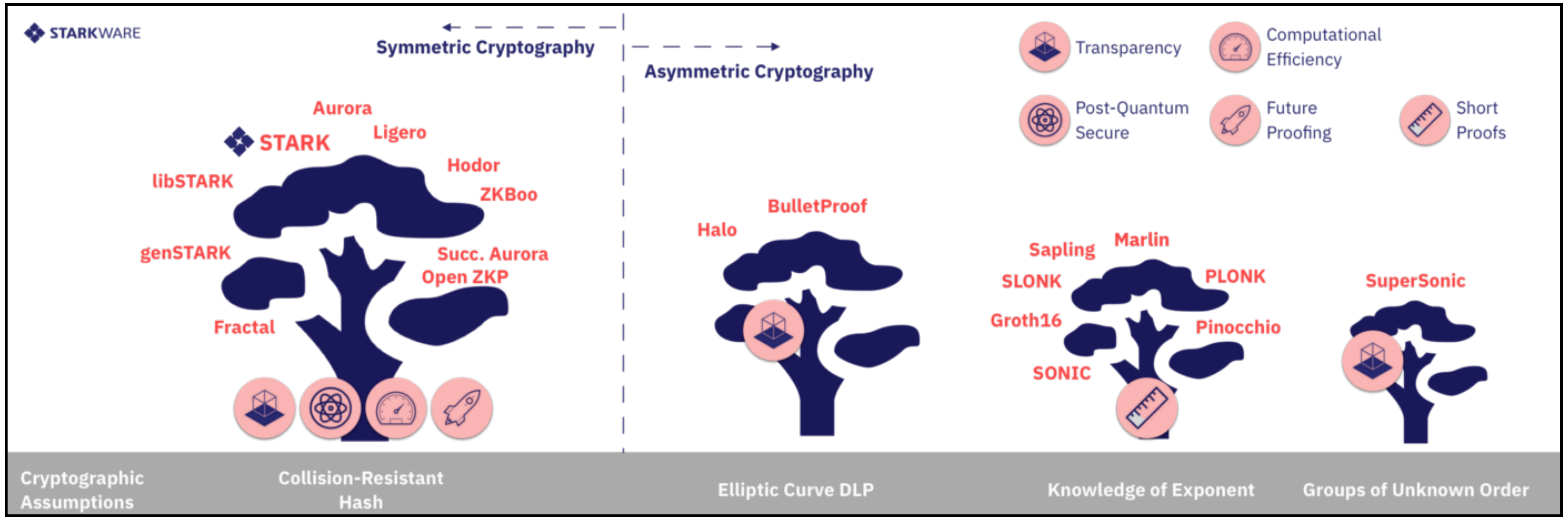
- **Groth16**, used in Zcash, Filecoin, and many others
- **Marlin**, a universal zkSNARK, used in Aleo
- **Circuits**, and in many other related crypto

Lessons applies to other systems (Plonk, SONIC, etc.), and other complex systems

zkSNARKs and friends

zkSNARKs are **not the only proof systems** used in practice

- STARKs: no trusted setup, proof size not constant, post-quantum (StarkWare)
- Bulletproofs: simpler, no trusted setup, but slower verification (Monero)



STARK = Scalable, Transparent ARgument of Knowledge

zkSNARKs' best years: 2018-2020

Zero-knowledge proof (ZKP) systems

ZKP System	Publication year	Protocol	Transparent	Universal	Plausibly Post-Quantum Secure	Programming Paradigm
Pinocchio[31]	2013	zk-SNARK	No	No	No	Procedural
Geppetto[32]	2015	zk-SNARK	No	No	No	Procedural
TinyRAM[33]	2013	zk-SNARK	No	No	No	Procedural
Buffet[34]	2015	zk-SNARK	No	No	No	Procedural
ZoKrates[35]	2018	zk-SNARK	No	No	No	Procedural
xJsnark[36]	2018	zk-SNARK	No	No	No	Procedural
vRAM[37]	2018	zk-SNARG	No	Yes	No	Assembly
vnTinyRAM[38]	2014	zk-SNARK	No	Yes	No	Procedural
MIRAGE[39]	2020	zk-SNARK	No	Yes	No	Arithmetic Circuits
Sonic[40]	2019	zk-SNARK	No	Yes	No	Arithmetic Circuits
Marlin[41]	2020	zk-SNARK	No	Yes	No	Arithmetic Circuits
PLONK[42]	2019	zk-SNARK	No	Yes	No	Arithmetic Circuits
SuperSonic[43]	2020	zk-SNARK	Yes	Yes	No	Arithmetic Circuits
Bulletproofs[44]	2018	Bulletproofs	Yes	Yes	No	Arithmetic Circuits
Hyrax[45]	2018	zk-SNARK	Yes	Yes	No	Arithmetic Circuits
Halo[46]	2019	zk-SNARK	Yes	Yes	No	Arithmetic Circuits
Virgo[47]	2020	zk-SNARK	Yes	Yes	Yes	Arithmetic Circuits
Ligero[48]	2017	zk-SNARK	Yes	Yes	Yes	Arithmetic Circuits
Aurora[49]	2019	zk-SNARK	Yes	Yes	Yes	Arithmetic Circuits
zk-STARK[50]	2019	zk-STARK	Yes	Yes	Yes	Assembly
Zilch[30] [51]	2021	zk-STARK	Yes	Yes	Yes	Object-Oriented

https://www.wikiwand.com/en/Zero-knowledge_proof

Why study zkSNARKs security?

A major risk for decentralised platforms (L2 protocols, private transactions):

- Complexity + Novelty => Non-trivial **bugs**
- A lot **at stake** ($$$, user data, user privacy)

Why study zkSNARKs security?

A major risk for decentralised platforms (L2 protocols, private transactions):

- Complexity + Novelty => Non-trivial **bugs**
- A lot **at stake** ($$$, user data, user privacy)

As a cryptographer since ~2005, **the most interesting** crypto I've seen:

- Intricate constructions with non-trivial components
- "Simple but complex" – non-interactive, but many moving parts
- "Multidimensional" way to reason about security
- "Real-worldness": not just papers – "code is specs"

What's zkSNARKs security? (it depends™)

Soundness: Invalid proofs should always be rejected (*solvencia*)

- Most obvious attack, often the *highest risk* in practice:
- Forging, altering, replaying valid proofs should be impossible

What's zkSNARKs security? (it depends™)

Soundness: Invalid proofs should always be rejected (*solvencia*)

- Most obvious attack, often the *highest risk* in practice:
- Forging, altering, replaying valid proofs should be impossible

Zero-knowledge: Proofs should not leak secret information (*conocimiento cero*)

- In practice, succinct proofs of large programs can leak only little data

What's zkSNARKs security? (it depends™)

Soundness: Invalid proofs should always be rejected (*solvencia*)

- Most obvious attack, often the *highest risk* in practice:
- Forging, altering, replaying valid proofs should be impossible

Zero-knowledge: Proofs should not leak secret information (*conocimiento cero*)

- In practice, succinct proofs of large programs can leak only little data

Completeness: Valid proofs should always be accepted (*totalidad*)

- Often a DoS/usability risk that may be further exploited
- All programs/circuits supported should be correctly processed

Who can find bugs?

A. Developers of the code (manually or via testing)

B. Developers of other projects' code

C. External auditors of the code

D. Users of the code, accidentally 😇

E. External "attackers" 😈

Security goal: you want A|B|C to find bugs before D|E

Bug hunting challenges

Practical zkSNARKs are recent, thus auditors often have

- Limited **experience** auditing zkSNARKs
- Limited **knowledge** of the theory and of implementations' tricks
- Limited **"checklist"** of bugs and bug classes
- Limited **tooling** and methodologies
- Limited **documentation** from the projects

How to make useful work nonetheless?

Bug hunting challenges

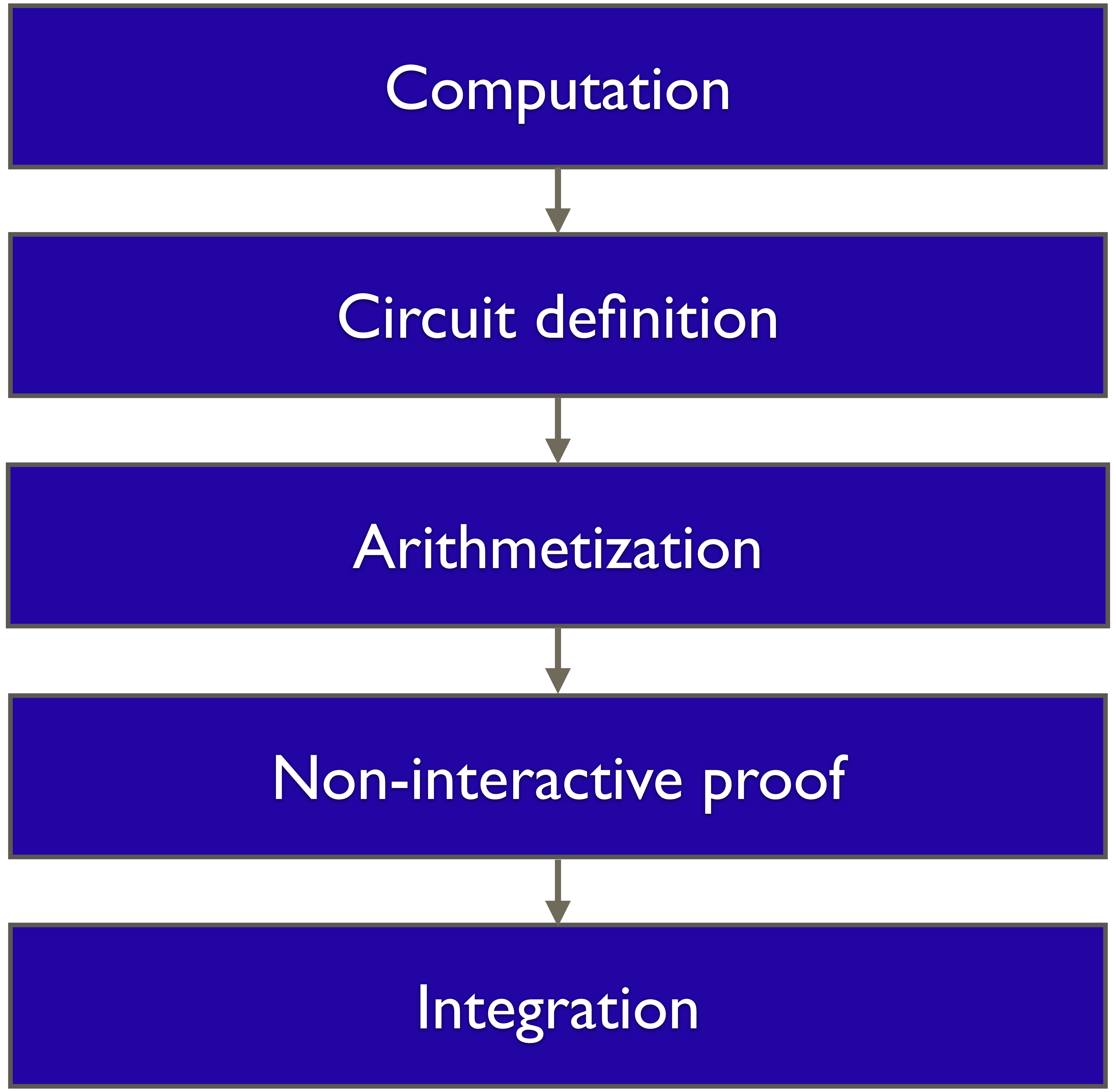
People think that finding vulnerabilities is about finding holes in code. But at some level it's not really about that. It's about understanding that the code itself is a hole in the swirling chaos of the world and just letting a little bit of that chaos in allows you to illuminate the whole thing.

Dave Aitel, unintentionally on ZK proofs bug hunting
https://seclists.org/dailydave/2022/q2/3

New crypto, new approach

- More **collaboration** with the devs/designers (joint review sessions, Q&As, etc.)
- More **threat analysis**, to understand the application's unique/novel risks
- Practical **experience**: writing PoCs, circuits, proof systems, etc.
- Learn **previous failures**, for example from…
  - Public disclosures and exploits
  - Other audit reports
  - Issue trackers / PRs
  - Community

General workflow, and failure *examples*



General workflow, and failure *examples*

Workflow stage	Failure example
Computation	The program's logic is not secure
Circuit definition	The circuit is not equivalent to the program
Arithmetization	The constraint system fails to enforce a constraint
Non-interactive proof	Insecure choice of primitives/parameters/properties
Integration	The application allows replays of previous proofs

How to break zkSNARKs? (1/2)

Break soundness, for example by exploiting

- Constraint system not effectively enforcing certain constraints
- Insecure generation or protection of private values

How to break zkSNARKs? (1/2)

Break soundness, for example by exploiting

- Constraint system not effectively enforcing certain constraints
- Insecure generation or protection of private values

Break zero-knowledge, for example by exploiting

- Private data treated as public variables
- Application-level "metadata attacks"

How to break zkSNARKs? (1/2)

Break soundness, for example by exploiting

- Constraint system not effectively enforcing certain constraints
- Insecure generation or protection of private values

Break zero-knowledge, for example by exploiting

- Private data treated as public variables
- Application-level "metadata attacks"

Break completeness, for example by exploiting

- Incorrect constraint synthesis behavior on edge cases (e.g. number of private vars)
- Gadget composition failure caused by type mismatch between gadget i/o values

How to break zkSNARKs? (2/2)

Break (off-chain) software, via any bug leading to

- Leakage of data, including via side channels (timing, oracles, etc.)
- Any form in insecure state (code execution, DoS)

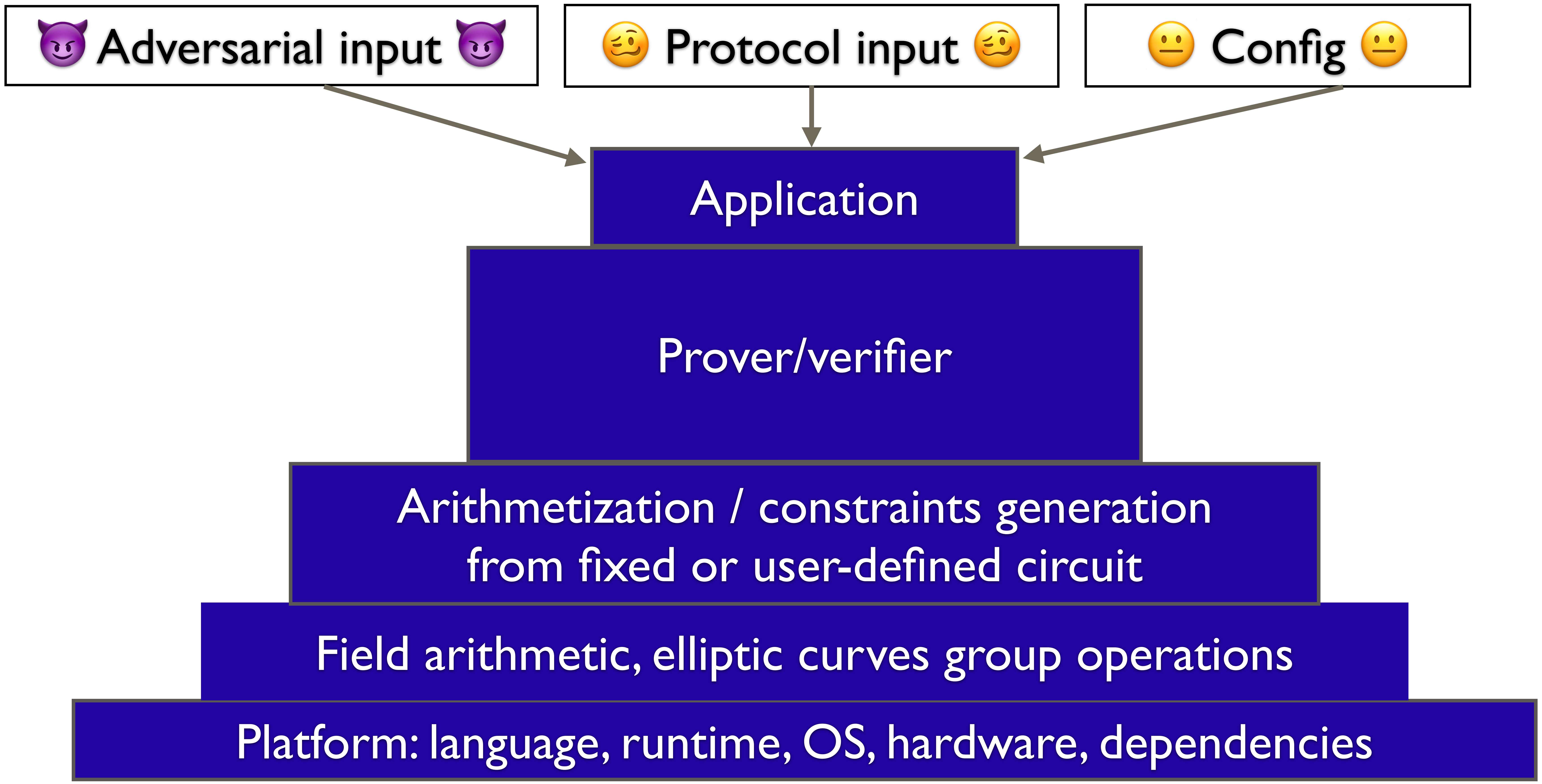
Compromise the "supply-chain", via

- Trusted setup's code and execution
- Build and release process integrity
- Software dependencies

Break (on-chain) software (incl. verifier) via smart contract bugs, logic flaws, etc.

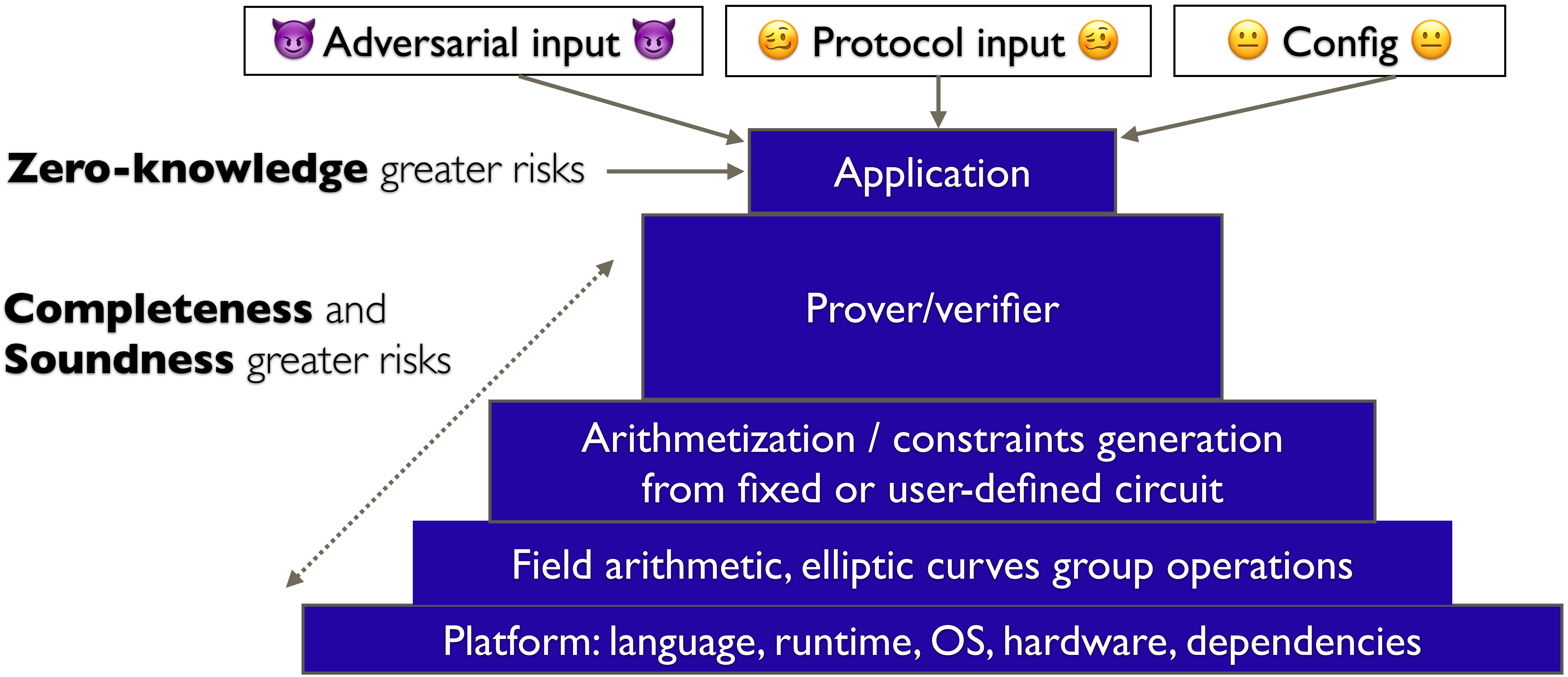
Need structure/methodology..

A failure in a **lower layer** can jeopardise the security of all upper layers



What to look for, and where?

A failure in a **lower layer** can jeopardise the security of all upper layers



Divide and conquer..

A failure in a **subcomponent** can jeopardise the security of all upper layers

Understand composability conditions..

Security 101: **Input validation** must be defined, implemented, and tested

😈 Adversarial input 😈 | 🥴 Protocol input 🥴 | 😐 Config 😐

Key management, Testing **Application** Interface, Side channels

Contracts between components must be defined to prevent **insecure composition**

Example: which component is responsible for **group membership** checks?

Elliptic curves, Pairings, Hash functions, PRF, Algebraic commitment
Randomness, Merkle trees **Prover/verifier** Linear algebra, Multi-exp.
Polynomial commitments, Fiat-Shamir transforms, etc. etc.

Real-word crypto bugs..

Soundness – Field arithmetic (1/n)

Vulnerability allowing double spend #16

Closed **poma** opened this issue on 26 Jul 2019 · 2 comments

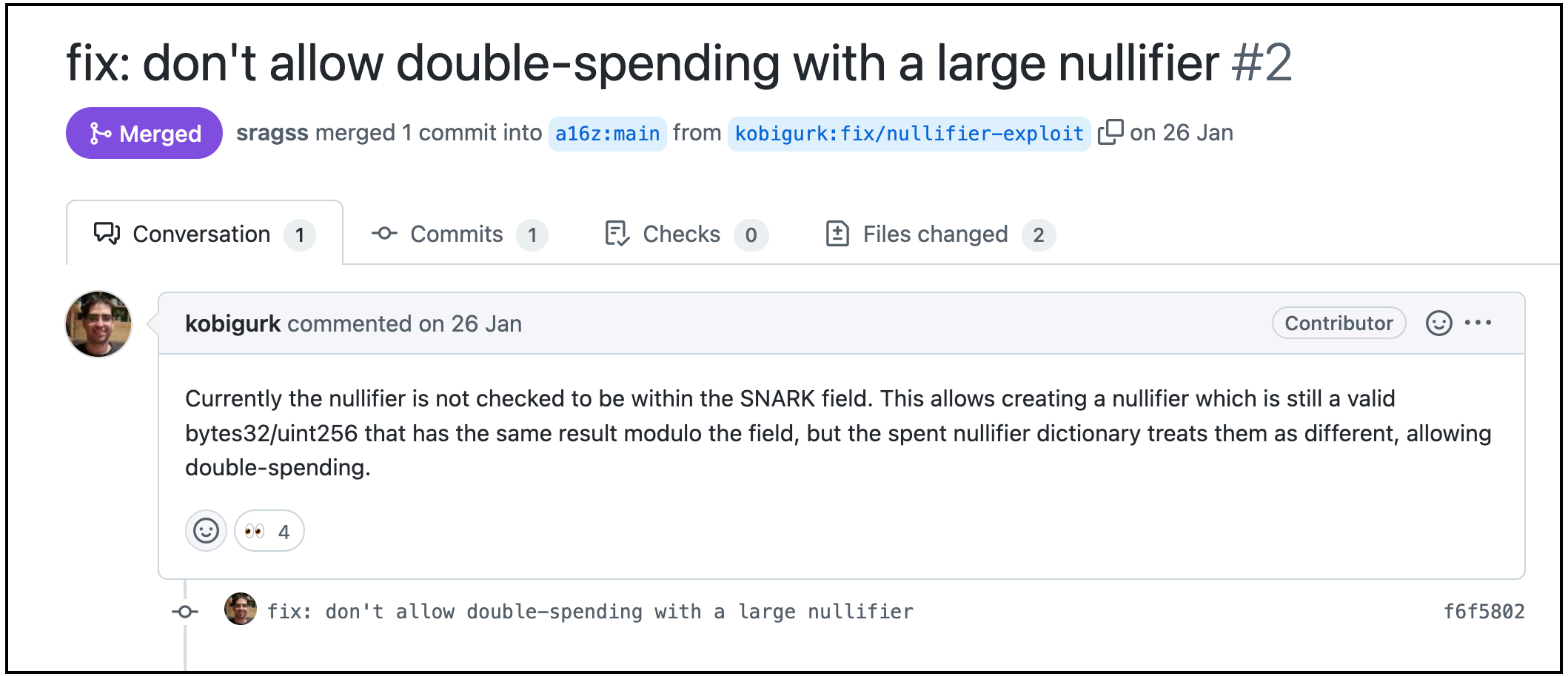
poma commented on 26 Jul 2019 • edited

Looks like in Semaphore.sol#L83 we don't check that nullifier length is less than field modulus. So `nullifier_hash + 21888242871839275222246405745257275088548364400416034343698204186575808495617` will also pass snark proof verification if it fits into uint256, allowing double spend.

Root cause: Missing overflow check of a nullifier (~ unique ID of a shielded payment)

https://github.com/appliedzkp/semaphore/issues/16

Soundness – Field arithmetic (2/n)



Root cause: Missing overflow check of a nullifier (~ unique ID of a shielded payment)

https://github.com/a16z/zkp-merkle-airdrop-contracts/pull/2

Soundness – Field arithmetic (3/n)

Potential security bug with the zk-SNARK verifier

Closed **weijiekoh** opened this issue on 21 Mar 2020 · 2 comments · Fixed by #43

weijiekoh commented on 21 Mar 2020

Expected Behavior

The `Verifier.verify()` function, not the function that calls it (i.e. `Shield.createMSA()` and `Shield.createPO()`, should require that each public input to the snark is less than the scalar field:

Missing overflow check (of a public circuit input)

https://github.com/eea-oasis/baseline/issues/34

Soundness – Field arithmetic (4/n)

```diff
-     // If the values are not in the correct range, the pairing check will fail.
+     // If the values are not in the correct range, the pairing check will fail
+     // because by EIP197 it verfies all input.
      Proof memory proof;
      proof.A = Pairing.G1Point(a[0], a[1]);
      proof.B = Pairing.G2Point([b[0][0], b[0][1]], [b[1][0], b[1][1]]);
@@ -219,7 +221,7 @@ contract Verifier {
      if (input.length + 1 != vk.IC.length) revert Pairing.InvalidProof();
      Pairing.G1Point memory vk_x = vk.IC[0];
      for (uint256 i = 0; i < input.length; i++) {
-       if (input[i] >= Pairing.SCALAR_MODULUS) revert Pairing.InvalidProof();
+       // By EIP196 the scalar_mul verifies it's input is in the correct range.
        vk_x = Pairing.addition(vk_x, Pairing.scalar_mul(vk.IC[i + 1], input[i]));
```

Missing overflow check (of a public circuit input)

https://github.com/appliedzkp/semaphore/pull/96/

Soundness – R1CS

Discuss: enforce `mul_by_inverse` #70

Merged weikengchen merged 7 commits into `master` from `fix-mul-by-inverse` on 6 Jul

Conversation 12 · Commits 7 · Checks 5 · Files changed 3

weikengchen commented on 4 Jul 2021 • edited — Member

Description

It seems that the `mul_by_inverse` implementation has a soundness issue that the newly allocated `d_inv` does not need to be the inverse of `d` but could be any value. This can be a soundness issue as the `poly` gadgets have used this API.

```
fn mul_by_inverse(&self, d: &Self) -> Result<Self, SynthesisError> {
    let d_inv = if self.is_constant() || d.is_constant() {
        d.inverse()?
    if self.is_constant() || d.is_constant() {
        let d_inv = d.inverse()?;
        Ok(d_inv * self)
    } else {
```

RUSTSEC-2021-0075 — History

Flaw in `FieldVar::mul_by_inverse` allows unsound R1CS constraint systems

Field element inverse property not enforced by the constraint system

https://github.com/arkworks-rs/r1cs-std/pull/70

Soundness – Trusted setup (paper)

> ## 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 in the paper. However, the vulnerability we discovered appears to have evaded his analysis. The vulnerability also appears in the subversion zero-knowledge SNARK scheme of [Fuchsbauer17], where an adaptation of [BCTV14] inherits the flaw. The vulnerability also appears in the ADSNARK construction described in [BBFR14]. Finally, the vulnerability evaded the Zcash Company's own cryptography team, which includes experts in the field that had identified several flaws in other parts of the system.

Theoretical flaw in the paper's setup description (sensitive values not cleared)

https://electriccoin.co/blog/zcash-counterfeiting-vulnerability-successfully-remediated/

Soundness – Fiat-Shamir (code and papers)

Coordinated disclosure of vulnerabilities affecting Girault, Bulletproofs, and PlonK

POST APRIL 13, 2022 LEAVE A COMMENT

By Jim Miller

- ZenGo's zk-paillier
- ING Bank's zkrp (deleted)
- SECBIT Labs' ckb-zkp
- Adjoint, Inc.'s bulletproofs
- Dusk Network's plonk
- Iden3's SnarkJS
- ConsenSys' gnark

The Problem

Why is this type of vulnerability so widespread? It really comes down to a combination of ambiguous descriptions in academic papers and a general lack of guidance around these protocols.

The vulnerabilities in one of these proof systems, Bulletproofs, stem from a mistake in the original academic paper, in which the authors recommend an insecure Fiat-Shamir generation. In addition to disclosing these issues to the above repositories, we've also reached out to the authors of Bulletproofs who have now fixed the mistake.

Incomplete Fiat-Shamiring of protocol transcript

https://blog.trailofbits.com/2022/04/13/part-1-coordinated-disclosure-of-vulnerabilities-affecting-girault-bulletproofs-and-plonk/

Soundness – Circuit

Filecoin —one PoREP vulnerability found by Trapdoor Tech

Trapdoor Tech discovered a serious vulnerability of the PoREP circuit (V25). Using this vulnerability, the calculation of SDR (Precommit1) can be directly omitted. Only one copy of all Sector Replica data is required. After the Trapdoor team communicated with the official in the first time, the official has quickly submitted the patch:

fix(storage-proofs-porep): add missing public input for replica-id #1088

Merged

dignifiedquire merged 6 commits into `master` from `fix/sdr-input` on 4 May 2020

Missing identifier value as public input, allowing replays

https://starli.medium.com/filecoin-one-porep-vulnerability-found-by-trapdoor-tech-7fc7beb4557b

Zero-knowledge – Application (Zcash, Monero)

Remote Side-Channel Attacks on Anonymous Transactions

Florian Tramèr*
Stanford University
tramer@cs.stanford.edu

Dan Boneh
Stanford University
dabo@cs.stanford.edu

Kenneth G. Paterson
ETH Zürich
kenny.paterson@inf.ethz.ch

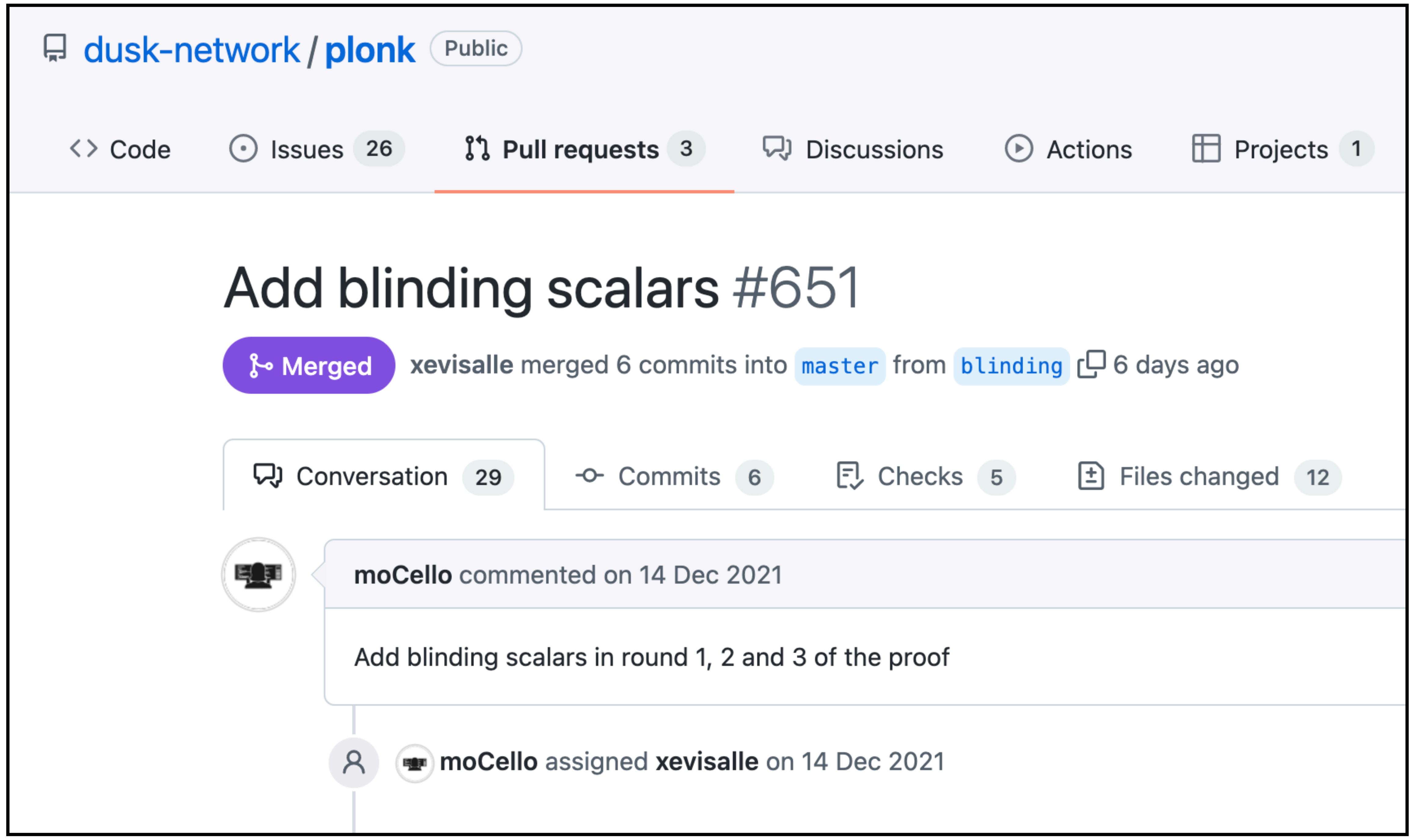
We exploit the fact that the time to produce a proof is correlated with the value of the prover's witness. As the witness contains the transaction amount, we expect this amount to be correlated with the proof time. For example, Zcash's proofs decompose the transaction amount into bits and compute an elliptic curve operation for each *non-zero* bit. The proof time is thus strongly correlated with the Hamming weight of the transaction amount, which is in turn correlated with its value.

Abstract: Privacy-focused crypto-currencies, such as Zcash or Monero, aim to provide strong cryptographic guarantees for transaction confidentiality and unlinkability. In this paper, we describe side-channel attacks that let remote adversaries bypass these protections. We present a general class of timing side-channel and traffic-analysis attacks on receiver privacy. These attacks enable an active remote adversary to identify the (secret) payee of any transaction in Zcash or Monero. The attacks violate the privacy goals of these crypto- currencies by exploiting side-channel information leaked by the implementation of different system components. Specifically, we show that a

Timing dependencies exploited to leak secrets and obtain oracles

https://eprint.iacr.org/2020/627.pdf

Zero-knowledge – Prover (Plonkup)



Missing (randomized) blinding to hide private inputs – *potential* ZK loss

https://github.com/dusk-network/plonk/pull/651

Completeness – DSL / Signatures

veorq commented yesterday

`crypto.signature.signature.verify()` rejects signatures with an `r`, inverse `s`, or message (hash) greater than `2**251 < EC_ORDER`:

cairo-lang/src/starkware/crypto/starkware/crypto/signature/signature.py
Lines 199 to 201 in 4e23351

```
assert 1 <= r < 2 ** N_ELEMENT_BITS_ECDSA, "r = %s" % r
assert 1 <= w < 2 ** N_ELEMENT_BITS_ECDSA, "w = %s" % w
assert 0 <= msg_hash < 2 ** N_ELEMENT_BITS_ECDSA, "msg_hash = %s" % msg_hash
```

There's a gap of ~2^196 values, thus a probability to hit an invalid `r` or `s` that is of the order of 2^(196-251)/2 = 2^54, when generating an ECDSA sig for some fixed message using a standard algorithm (rather than Cairo's `sign()`, which enforces these constraints).

I can't think of a specific attack scenario at the moment, but I would expect to find applications where either

1. that accidental failure rate would be unacceptably high, or
2. adversaries could bruteforce invalid sigs to do some kind of DoS, or worse (with plausible deniability)

I probably miss some of the context, and you may have a good reason to verify sigs that way.

Valid signatures rejected, risk initially deemed negligible

https://github.com/starkware-libs/cairo-lang/issues/39

Conclusions

😌 **Why not be too scared?**

- Robust code and frameworks (e.g. Rust projects such as arkworks and zkcrypto)
- Safe code easier to write with DSLs (Cairo, Leo, etc.) and reusable gadgets/chips
- Improvement in secure SDLC (initiatives like slsa.dev/, GitHub advanced security)
- Relatively narrow attack surface in practice

Conclusions

😌 **Why not be too scared?**

- Robust code and frameworks (e.g. Rust projects such as arkworks and zkcrypto)
- Safe code easier to write with DSLs (Cairo, Leo, etc.) and reusable gadgets/chips
- Improvement in secure SDLC (initiatives like slsa.dev/, GitHub advanced security)
- Relatively narrow attack surface in practice

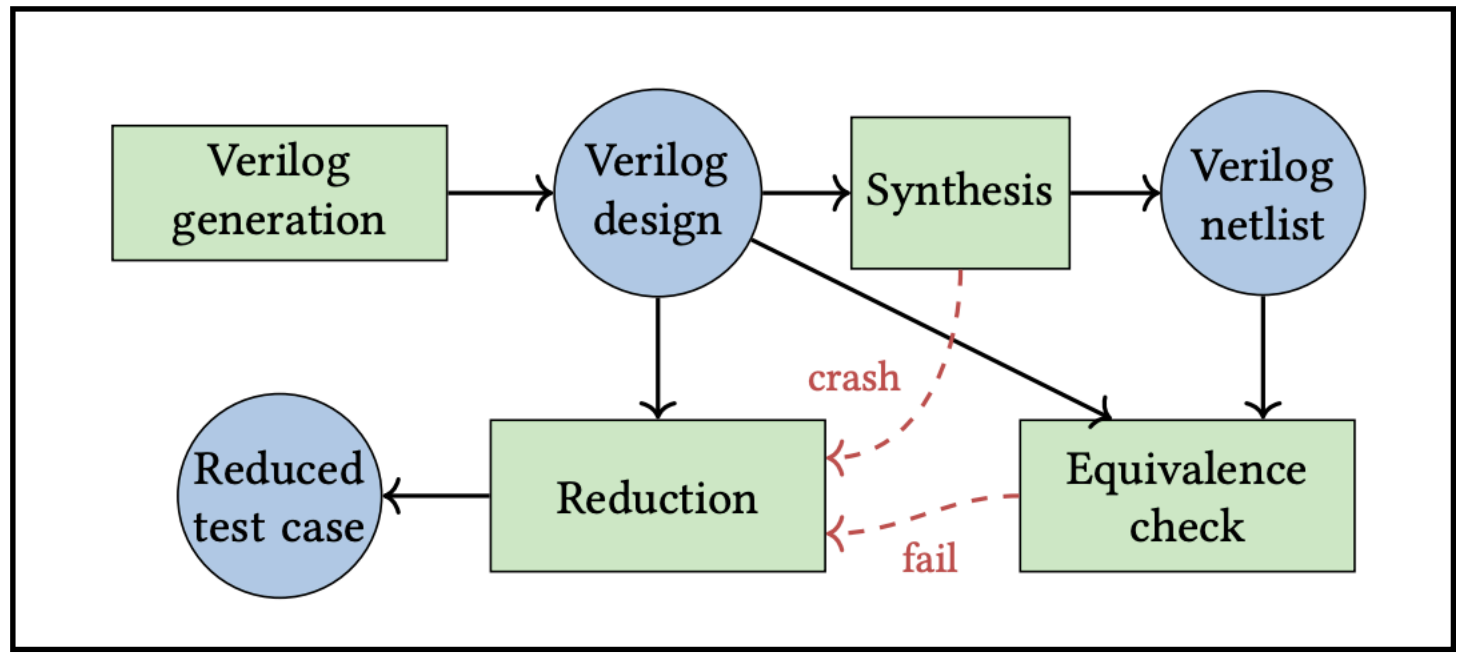
😱 **Why be scared?**

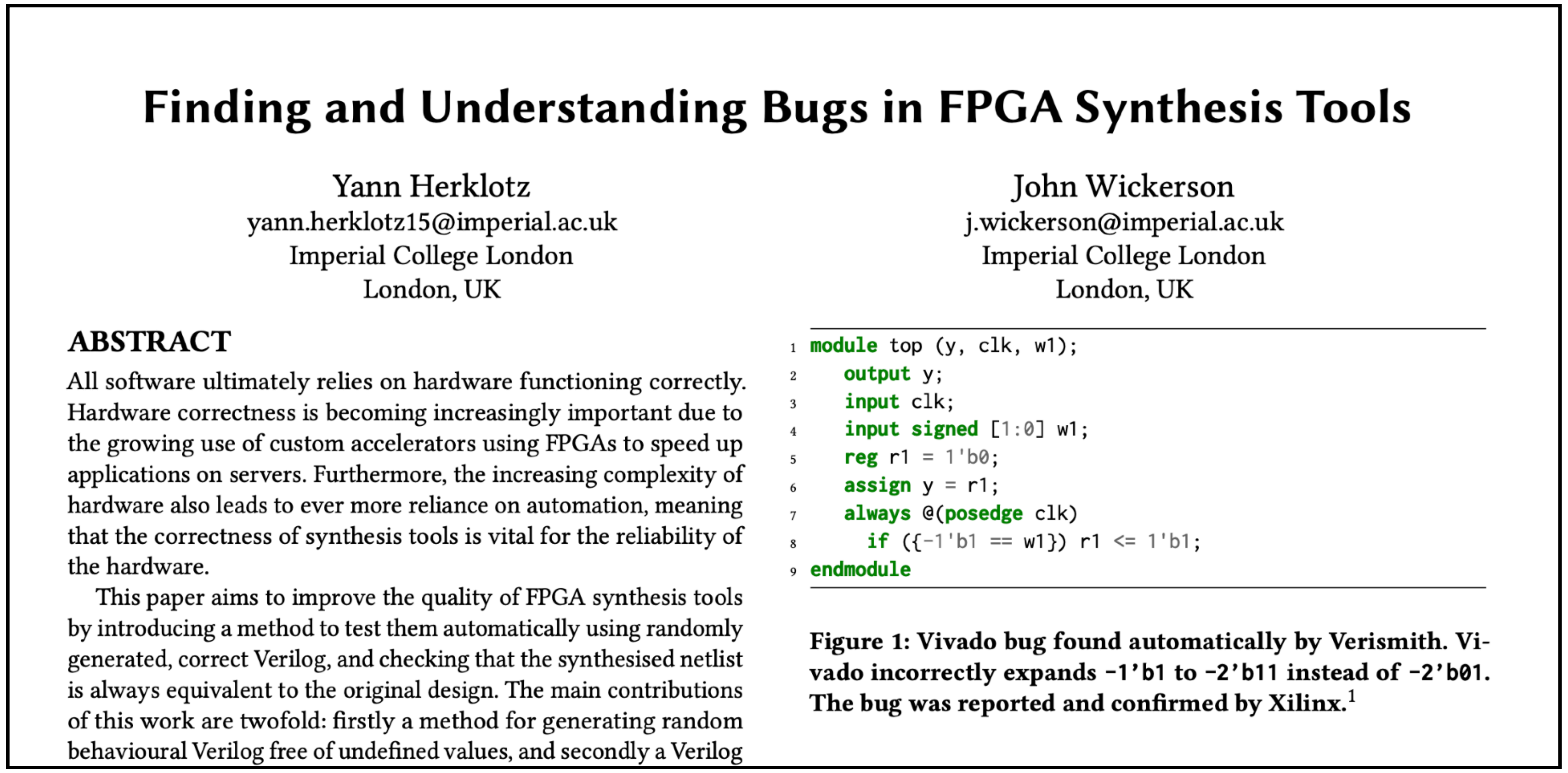
- Few people understand zkSNARKs, even fewer can find bugs
- Limited maturity level in many ZK/blockchain projects' SDLC
- Lack of tooling (testing, fuzzing, verification)
- More ZK usage => more $$$ at stake => greater RoI for attackers

Conclusions

Learn from **hardware circuit synthesizers**?

- HDL-to-netlist ≈ Program-to-constraints – same, but different
- History of bugs and tooling
- Testing methodologies





Finding and Understanding Bugs in FPGA Synthesis Tools

Yann Herklotz
yann.herklotz15@imperial.ac.uk
Imperial College London
London, UK

John Wickerson
j.wickerson@imperial.ac.uk
Imperial College London
London, UK

ABSTRACT

All software ultimately relies on hardware functioning correctly. Hardware correctness is becoming increasingly important due to the growing use of custom accelerators using FPGAs to speed up applications on servers. Furthermore, the increasing complexity of hardware also leads to ever more reliance on automation, meaning that the correctness of synthesis tools is vital for the reliability of the hardware.

This paper aims to improve the quality of FPGA synthesis tools by introducing a method to test them automatically using randomly generated, correct Verilog, and checking that the synthesised netlist is always equivalent to the original design. The main contributions of this work are twofold: firstly a method for generating random behavioural Verilog free of undefined values, and secondly a Verilog


```
module top (y, clk, w1);
  output y;
  input clk;
  input signed [1:0] w1;
  reg r1 = 1'b0;
  assign y = r1;
  always @(posedge clk)
    if ({-1'b1 == w1}) r1 <= 1'b1;
endmodule
```

Figure 1: Vivado bug found automatically by Verismith. Vivado incorrectly expands -1'b1 to -2'b11 instead of -2'b01. The bug was reported and confirmed by Xilinx.[1]

https://johnwickerson.github.io/papers/verismith_fpga20.pdf

Conclusions

Learning resources and projects:

- zkproof.org community and events
- zkhack.dev virtual event
- zkvalidator.com initiative
- zeroknowledge.fm podcast
- zkStudyClub video series
  http://youtu.be/playlist?list=PLj80z0cJm8QHm_9BdZ1BqcGbgE-BEn-3Y

zk-SNARKs: A Gentle Introduction

Anca Nitulescu

https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf

¡Muchas gracias!

JP Aumasson
@veorq

CSO @ taurushq.com

Big thank yous for their help and feedback to:
Aleo, Protocol Labs, Kobi Gurkan, Adrian Hamelink, Daira Hopwood, Daniel Jacob Bilar, David Wong, Lúcás Meier, Mathilde Raynal