Insights from/on Taxonomy of ZKP Vulnerabilities

Gyumin Roh

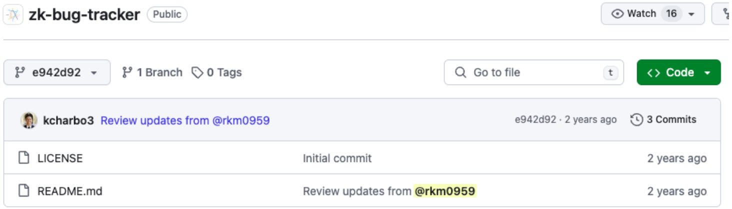
Security Researcher, KALOS

April 10th

Table of Contents

History of the Taxonomy

September 2022, Kyle Charbonnet



zk-bug-tracker Public

Watch 16

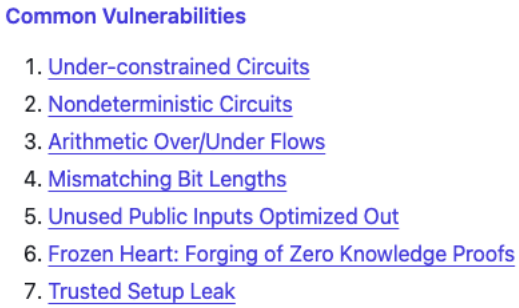
e942d92 · 1 Branch · 0 Tags · Go to file t · Code

kcharbo3 Review updates from @rkm0959 · e942d92 · 2 years ago · 3 Commits

LICENSE	Initial commit	2 years ago
README.md	Review updates from @rkm0959	2 years ago

History of the Taxonomy

September 2022, Kyle Charbonnet

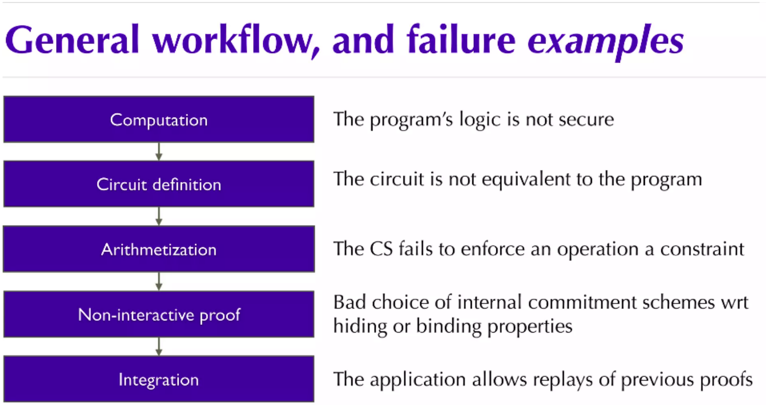


Common Vulnerabilities

1. Under-constrained Circuits
2. Nondeterministic Circuits
3. Arithmetic Over/Under Flows
4. Mismatching Bit Lengths
5. Unused Public Inputs Optimized Out
6. Frozen Heart: Forging of Zero Knowledge Proofs
7. Trusted Setup Leak

History of Taxonomy

April 2022, JP Aumasson, zkStudyClub


General workflow, and failure *examples*

The program's logic is not secure

The circuit is not equivalent to the program

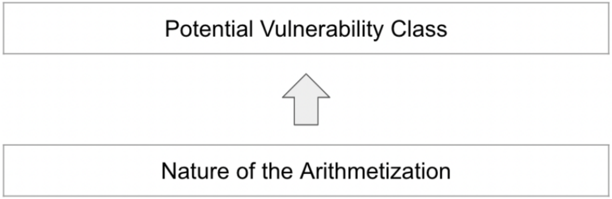
The CS fails to enforce an operation a constraint

Bad choice of internal commitment schemes wrt hiding or binding properties

The application allows replays of previous proofs

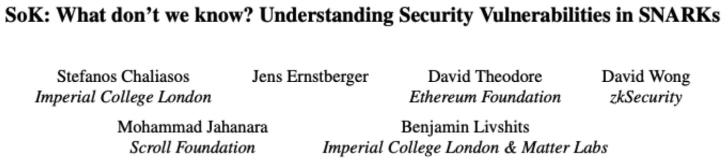
History of Taxonomy

June 2023, Gyumin Roh, ETH Seoul



History of Taxonomy

March 2024

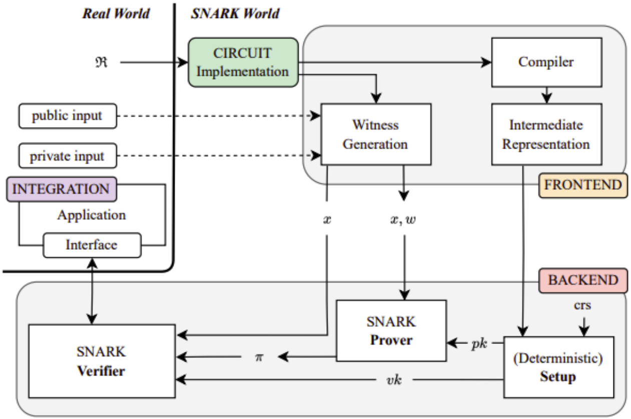


SoK: What don't we know? Understanding Security Vulnerabilities in SNARKs

Stefanos Chaliasos, *Imperial College London*
Jens Ernstberger
David Theodore, *Ethereum Foundation*
David Wong, *zkSecurity*
Mohammad Jahanara, *Scroll Foundation*
Benjamin Livshits, *Imperial College London & Matter Labs*

History of Taxonomy

March 2024



Challenges

- better classification for circuit layer issues
- we really need more dataset/work on frontend/backend

Challenges

- better classification for circuit layer issues
- we really need more dataset/work on frontend/backend
- **we need more insights from these surveys**

Ultimate Goal

- we work on this taxonomy to improve ZKP security
- it seems that ZKP security needs more strong auditors

Ultimate Goal

- we work on this taxonomy to improve ZKP security
- it seems that ZKP security needs more strong auditors
- **surveys should provide insights to lower the barrier**

The Game Plan

- the new factor is “cryptography exposed to the surface”

The Game Plan

- the new factor is "cryptography exposed to the surface"
- **focus on this set** to onboard strong web3 security researchers

The Game Plan

Focusing on this set leads to the following questions

- what properties of arithmetization makes implementing a circuit hard?
- how should one view ZKP systems as black boxes?

The Game Plan

Focusing on this set leads to the following questions

- what properties of arithmetization makes implementing a circuit hard?
- how should one view ZKP systems as black boxes?
- dividing layers allow *separation* - we should utilize it!

Contributions

This line of thinking directly gives

- a “natural” way of understanding this taxonomy (ETH Seoul ’23)

Contributions

This line of thinking directly gives

- a “natural” way of understanding this taxonomy (ETH Seoul ’23)
- the additional prerequisites for auditors
- especially, **how much, and what math auditors need**

Contributions

This line of thinking directly gives

- a "natural" way of understanding this taxonomy (ETH Seoul '23)
- the additional prerequisites for auditors
- especially, **how much, and what math auditors need**
- **plug-and-playable to the ZKP development metagame**

Table of Contents

The Circuit Layer's Essence

We write circuits to encode a computation as a constraint system

- this constraint system should be ZKP-friendly

Taxonomy from "Nature of Arithmetization"

- limited set of constraints $\rightarrow$ Assigned, not Constrained
- constraints are expensive $\rightarrow$ Assumptions Handling
- various proof configurations $\rightarrow$ Unsafe Reuse of Circuit
- common usage of selectors $\rightarrow$ Incorrect Custom Gates
- usage of $\mathbb{F}_p$ $\rightarrow$ Overflows/Underflows
- the usual mistakes $\rightarrow$ Design, Programming Errors

Taxonomy from "Nature of Arithmetization"

Further work can be done

- cryptographic tricks $\rightarrow$ Incorrect Cryptographic Tricks

Taxonomy from “Nature of Arithmetization”

Further work can be done

- cryptographic tricks $\rightarrow$ Incorrect Cryptographic Tricks
- fixed witness board for PLONKish $\rightarrow$ Variable Length Objects

Taxonomy from "Nature of Arithmetization"

Further work can be done

- cryptographic tricks $\rightarrow$ Incorrect Cryptographic Tricks
- fixed witness board for PLONKish $\rightarrow$ Variable Length Objects
- PIOP for PLONKish $\rightarrow$ Incorrect Boundary/Transition Constraints

Incorrect Cryptographic Tricks

- Lookups: dynamic lookups, vector lookups (RLC)
- Fiat-Shamir: RLC is commonly used within the circuit, aka "Phases"
- Memory-Checking: relatively new technique, sometimes unintuitive

and in general, mathy tricks are used (emulated fields, multiplexers...)

Variable Length Objects

We need a fixed maximum length array and assign a variable for its length. Therefore, via prefix/suffix zeros, one could have

$$\mathrm{RLC}(a) = \mathrm{RLC}(b), \quad a \neq b$$

Sometimes this leads to fixed columns not being fixed for each instance.

Incorrect Boundary/Transition Constraints

The first, last rows are very important

Selector	Value	Accumulator
1	5	5
1	8	58
0	7	587

$$sel \cdot (10 \cdot accu + val' - accu') = 0$$

Integration Layer: Black Box

The integration stack *can* be black-boxed as follows

- The user has public input x_{pub} and secret w_{priv}
- The prover computes witnesses $w \leftarrow W(x_{pub}, w_{priv})$
- The prover computes proof $\pi \leftarrow P(x_{pub}, w)$
- The verifier checks π with x_{pub}: this checks $f(x_{pub}, w) = 0$.

Integration Layer: Black Box

The integration stack *can* be black-boxed as follows

- The user has public input x_{pub} and secret w_{priv}
- The prover computes witnesses $w \leftarrow W(x_{pub}, w_{priv})$
- The prover computes proof $\pi \leftarrow P(x_{pub}, w)$
- The verifier checks π with x_{pub}: this checks $f(x_{pub}, w) = 0$.
- **Main Characters:** $f, x_{pub}, w_{priv}, \pi$

Integration Layer: f (R10)

We can assume that the circuit correctly encodes the computation f

Integration Layer: f (R10)

We can assume that the circuit correctly encodes the computation f

f vulnerabilities

The goal is to check f and the verifier's additional logic are sufficient.
This is a **specification** issue: so these should be familiar.

Integration Layer: x_{pub} (R10)

Public inputs “connect” the ZKP world and the verifier logic itself.
The “native type” for ZKP is $\mathbb{F}_p$, but on L1 it’s usually uint256

Problem

x and $x + p$ are different in uint256 but same in $\mathbb{F}_p$ - double spending...

Integration Layer: x_{pub} (R10)

Public inputs "connect" the ZKP world and the verifier logic itself.
The "native type" for ZKP is $\mathbb{F}_p$, but on L1 it's usually uint256

Problem

x and $x + p$ are different in uint256 but same in $\mathbb{F}_p$ - double spending...

Solution

Force $x \in [0, p)$ to make the underlying type equal

Integration Layer: w_{priv} (R11)

The secrecy of w_{priv} is important for Zero-Knowledge applications.

Integration Layer: w_{priv} (R11)

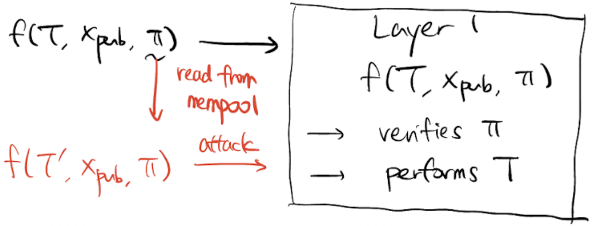
The secrecy of w_{priv} is important for Zero-Knowledge applications.

w_{priv} vulnerabilities

The task is to see leakage of w_{priv}, which is usually a design issue.

Integration Layer: π (R11)

The main idea: attacker front-runs π without knowing w_{priv}



usual defenses include adding T to x_{pub} to stop malleability

Table of Contents

Current Status

We're focusing on the circuit layer and integration layer. You need

- high level understanding of ZKP and its properties
- arithmetization and $\mathbb{F}_p$ **(no need for PIOP/PCP)**

Current Status

We're focusing on the circuit layer and integration layer. You need

- high level understanding of ZKP and its properties
- arithmetization and $\mathbb{F}_p$ **(no need for PIOP/PCP)**
- good command of cryptographic tricks and math tricks

Current Status

We're focusing on the circuit layer and integration layer. You need

- high level understanding of ZKP and its properties
- arithmetization and $\mathbb{F}_p$ **(no need for PIOP/PCP)**
- good command of cryptographic tricks and math tricks
- knowledge on malleability regarding SNARK proofs

Optimization Tactics

- The goal is to understand **“cryptography exposed to the surface”**.

Optimization Tactics

- The goal is to understand **“cryptography exposed to the surface”**.
- difficult cryptography, but simple conclusions/APIs

Optimization Tactics

- The goal is to understand **"cryptography exposed to the surface"**.
- difficult cryptography, but simple conclusions/APIs
- $\rightarrow$ these conclusions can be **black-boxed** for use

Optimization Tactics

- The goal is to understand **“cryptography exposed to the surface”**.
- difficult cryptography, but simple conclusions/APIs
- $\rightarrow$ these conclusions can be **black-boxed** for use
- **additional layering** leads to more available black-boxes

Optimization Tactics

- The goal is to understand **“cryptography exposed to the surface”**.
- difficult cryptography, but simple conclusions/APIs
- $\rightarrow$ these conclusions can be **black-boxed** for use
- **additional layering** leads to more available black-boxes
- layering and black-boxing help to **reduce the workload**

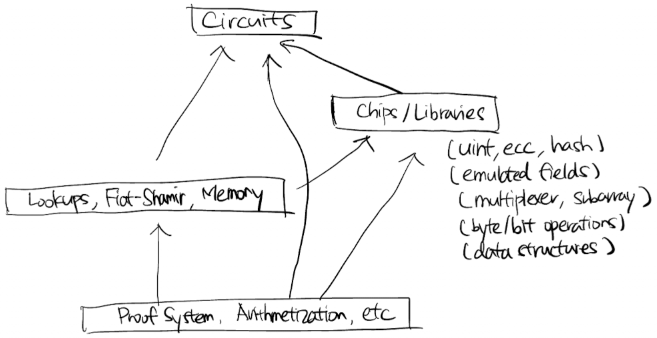
Malleability

Thankfully, most of the work is done already

- KZG-based: simulation-extractable, so non-malleable (2023/569)
- Groth16: malleable, but dummy constraints in the backend layer

remaining attack ideas on the proof π should be familiar.

Various Trickery



libraries can reduce the “exposed” cryptographic tricks

Intuition: Fiat-Shamir and RLC

Transcripts must have *everything computable so far*.

- in this case, the hash can be considered as uniform random
- note that this is exactly how ROM works anyway
- also applies to vector lookups

Intuition: Fiat-Shamir and RLC

Consider a classic RLC application - to prove c is the concat of a and b, where a, b, c are strings of fixed length n_a, n_b, n_c, one checks that

$$\mathrm{RLC}(a) \cdot \gamma^{n_b} + \mathrm{RLC}(b) = \mathrm{RLC}(c)$$

where we define RLC as, with randomness γ,

$$\mathrm{RLC}(a_0, \cdots, a_{n-1}) = \sum_{i=0}^{n-1} a_i \cdot \gamma^{n-1-i}$$

Note that a, b, c are computable already, so $\gamma = H(a, b, c)$.

If $\gamma = H(a, b)$, the intuition of attack is that one fixes a, b, γ and works around incorrect values of c that satisfy the constraint.

Remark: Fiat-Shamir and RLC

Note that launching a full attack may be more involved

- Last Challenge Attack from OpenZeppelin
- Frozen Heart Vulnerability from Trail of Bits
- Incorrect RLC Attack via LLL from Zellic & KALOS

however, *finding* the vulnerability can be done with mostly “intuition”

Memory Checking Models

Consider a log-derivative based single array memory checking

$$\sum_{(m_i, a_i, v_i, t_i) \in IN} \frac{m_i}{\beta - (a_i + \gamma v_i + \gamma^2 t_i)} = \sum_{(m_o, a_o, v_o, t_o) \in OUT} \frac{m_o}{\beta - (a_o + \gamma v_o + \gamma^2 t_o)}$$

what we are really doing is $\sum m_i = \sum m_o$ over each tuple (a, v, t).

Table of Contents

Hidden Prerequisite from f

We assumed that the computation f itself is familiar: however in many cases f is also quite mathematical, so the there is a hidden prerequisite.

- Elliptic Curves, Hash Functions, SNARK verifier itself, etc...

better fundamentals $\rightarrow$ more f auditable $\rightarrow$ more applications to audit

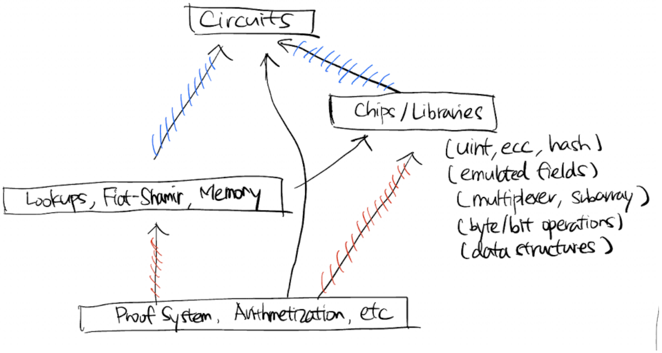
Prerequisites $\neq$ Difficulty

We mostly discussed prerequisites, but difficulty is a different thing.

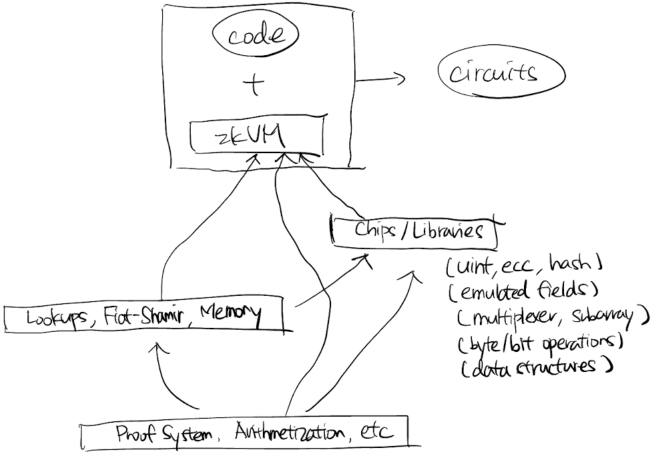
Implementing complex data structures with possible instructions is difficult, even if they don't require deep understanding of ZKP cryptography.

Also, understanding PLONKish arithmetization deeply is not easy.

if you look close enough



The dawn of zkVMs (or very strong DSLs)



zkVM's implications on ZKP audits

What happens when **"cryptography exposed to the surface"** is zero?

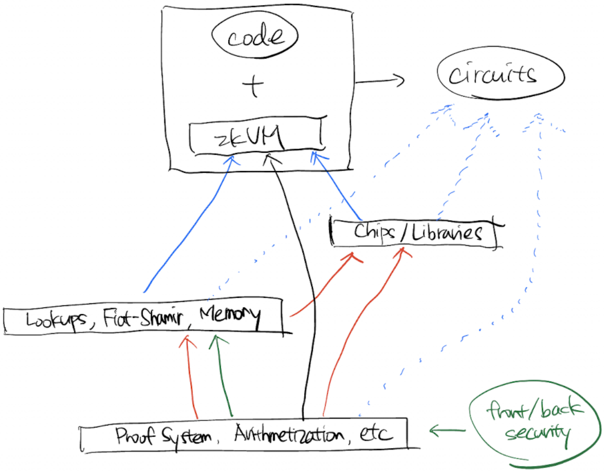
- "write code not circuits" meta implies "audit code not circuits"
- a good time to think about the strength of zkVM

zkVM's implications on ZKP audits

What happens when **"cryptography exposed to the surface"** is zero?

- "write code not circuits" meta implies "audit code not circuits"
- a good time to think about the strength of zkVM
- specialized circuits are important for optimization, but for how long?

The Three Layers


code
+
zkVM
circuits
Chips / Libraries
Lookups, Fiat-Shamir, Memory
Proof System, Arithmetization, etc
front/back security

The Three Layers

- blue: puzzle-like skillset, fundamentals and intuition
- red: strong command of cryptography/math tricks and their basis
- green: strong command on the entire ZKP stack

Plug-and-Play on the zkVM meta

- there may be less circuits to audit
- circuit layer audits won't be this dominant

Plug-and-Play on the zkVM meta

- there may be less circuits to audit
- circuit layer audits won't be this dominant
- front/backend layers will be a big target

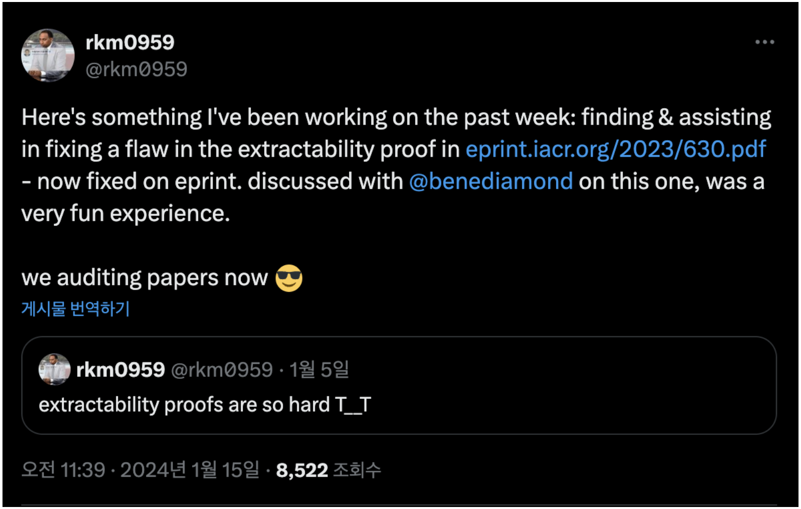
Plug-and-Play on the zkVM meta

- there may be less circuits to audit
- circuit layer audits won't be this dominant
- front/backend layers will be a big target
- now f is about building a VM, so you need to know about VMs

Plug-and-Play on the zkVM meta

- there may be less circuits to audit
- circuit layer audits won't be this dominant
- front/backend layers will be a big target
- now f is about building a VM, so you need to know about VMs
- **overall, rewards heavy cryptography/math/compiler fans**

lol?



rkm0959
@rkm0959

Here's something I've been working on the past week: finding & assisting in fixing a flaw in the extractability proof in eprint.iacr.org/2023/630.pdf - now fixed on eprint. discussed with @benediamond on this one, was a very fun experience.

we auditing papers now 😎

게시물 번역하기

rkm0959 @rkm0959 · 1월 5일
extractability proofs are so hard T_T

오전 11:39 · 2024년 1월 15일 · **8,522** 조회수

Plug-and-Play Conclusions

Currently, I believe that

In the zkVM meta, I believe

Plug-and-Play Conclusions

Currently, I believe that

- plenty of areas that don't require super heavy math/cryptography
- black-box reductions and layer separation help build intuition
- these intuitions and insights can help onboard auditors

In the zkVM meta, I believe

Plug-and-Play Conclusions

Currently, I believe that

- plenty of areas that don't require super heavy math/cryptography
- black-box reductions and layer separation help build intuition
- these intuitions and insights can help onboard auditors

In the zkVM meta, I believe

- strong knowledge on the entire ZKP stack will be needed
- knowledge on VMs, compilers will be also important

Final Review

This presentation's contributions were

Final Review

This presentation's contributions were

- we give insights *on* taxonomy, by considering *why* these bugs happen: to do so, we focused on the "cryptography exposed to the surface"

Final Review

This presentation's contributions were

- we give insights *on* taxonomy, by considering *why* these bugs happen: to do so, we focused on the "cryptography exposed to the surface"
- we derive the auditor's prerequisites *from* the taxonomy, and by using black boxes and additional layers, we separate the stack into parts requiring different skill levels - helping auditors onboard

Final Review

This presentation's contributions were

- we give insights *on* taxonomy, by considering *why* these bugs happen: to do so, we focused on the "cryptography exposed to the surface"
- we derive the auditor's prerequisites *from* the taxonomy, and by using black boxes and additional layers, we separate the stack into parts requiring different skill levels - helping auditors onboard
- we show how to use the big picture to view the ZKP security landscape based on the ZKP development metagame, with a focus on zkVM