BGIN SR 011 (Draft)
(Study Report)

ZKP and its application

Part1: ZKP Technology

Leona Hioki BGIN
Contributer
28.07.2023

v0.0.1

Introduction

Part1: ZKP Technology -

-Definition of ZKP:
ZKP,
NIZK
- Terminology
- Composition and Characteristics of NIZK
- Recursive ZKP

- Classification of NIZK:
-History of NIZK:
-Backend:
- Frontend
- Classification

Part2: ZKP and Society
- Programming Languages and Libraries Exclusive to ZKP:
 - Trade-offs in ZKP Performance
- History of Vulnerability Discoveries in ZKP
- Examples of ZKP Applications
- zkRollup: Definition of zkRollup,
-History of zkRollup
-Concerns about zkRollup

Disclaimer

Researcher map

Table of contents

1 Definition of ZKP **7**

1. Scope

The scope of this document is

2. Normative reference

This document has no normative reference.

3. Terms and definitions

This document uses the following terms as the shortcut for more complete wording provided as the definition. When the term appears within this document, it should be read as being replaced by the term.

3.1
Confidentiality
amount of information about the plaintext remains unchanged before and after viewing the ciphertext

3.2
Finite Field
mathematical structure where all values are the remainder when divided by an order n

EXAMPLE: For an order of 5, it's represented as F_5. In this case, 7 in F_5 is equivalent to 2

a%n + b%n = (a+b)%n
a%n x b%n = ab%n

3.3
Extension Field
structure formed when a value not present in a field is added

Note to entry: When real numbers are termed as a Field, complex numbers are considered an extension of real numbers by adding a value i. Similar values can be

added to a finite field, adding dimensions akin to complex numbers. In this context, "extension field" refers to the extension of a finite field.

3.4
Elliptic Curve
equation that can be represented as y^2 = x^3 + ax + b

Note to entry: In this context, an elliptic curve refers to one over a finite field. Points P and Q on the elliptic curve have a defined addition operation, making P + Q = R. While direct multiplication, such as P x Q, isn't defined, scalar multiplication like P+P=2P and nP (n times P) is defined. Due to the discrete logarithm problem, it's challenging to derive n from nP.

3.5
order
order of the finite field that defines the elliptic curve

3.6
characteristic
total number of points on that curve given an elliptic curve's functions and its order

Note to entry: If the characteristic is q, then for any point P on that curve, qP equals P.

3.7
hash function
function that ensures different outputs with high probability for different inputs, is challenging to discern the input from its output, and always produces the same output for the same input

3.8
bit Commitment
verification system that guarantees the committed value remains unchanged and remains unknown (hiding) until it's revealed. It can be described that most of today's blockchains are this system with a timestamp added

3.9
hiding
state of commitment without revealing

Note to entry: If various verifications can be done without revealing, it can be applied to confidentiality.

3.10
arithmetic circuit
circuit that performs arithmetic operations on the input, not a digital circuit

Note to entry: The operation receiving each input is called a gate, and in the context of ZKP, this operation's equation becomes a constraint. In many NIZK systems, there's a strong positive correlation between the number of constraints and the proof time.

3.11
polynomial problem
problem of whether one knows a polynomial that meets certain conditions

Note to entry: In many cases of ZKP, knowing certain input eventually gets converted or reduced to a polynomial problem.

3.12
witness
value needed for proof that can be obtained during the calculation process when input into a circuit

3.13
instance
dataset where the witness is actually substituted for the gate

3.14
fiat-shamir transformation
method to convert interactive ZKP where a random oracle is used into a non-interactive format

Note to entry: Using a hash function, the prover side can produce a value that they cannot control during the computation process, allowing the verifier side's random oracle role to be transferred to the prover side.

4. Abbreviations and symbols

In this document, the following abbreviations and symbols are used.

IPA —

FRI —
NIZK — non-Interactive zero-knowledge
R1CS —
ZKP — zero-knowledge proof
zkSNARKs —

5 Definition of ZKP

The invention and explicit mention of Zero Knowledge Proof (ZKP) was first demonstrated in 1985 in "The Knowledge Complexity of Interactive Proof Systems" by authors Shafi Goldwasser, Silvio Micali, and Charles Rackoff.

The features generally defined for it can be described as follows:

For a given statement, within a protocol where a Prover and Verifier exist, a ZKP satisfies the following:

Completeness: When the statement is true, the verifier can be convinced of its truthfulness by the prover.

Soundness: If the statement is false, the prover cannot convince the verifier to accept it as true.

Zero-Knowledge: The verifier learns only the truth of the statement and no other information.

・NIZK

Among ZKPs, those where the verifier and prover do not require interactive communication and it concludes solely with the verifier checking the proof sent by the prover are called Non-Interactive Zero-Knowledge (NIZK). zkSNARKs, zkSTARKs, and BulletProofs fall under this NIZK category

・Common reference string generated from random numbers

The defining features for this can be described as:

Procedure:

・There exists a value called a common reference string (crs) that is generated from random numbers.
・From the crs, input, and witness, the prover calculates a proof.

・The verifier either accepts or rejects the proof.

For this procedure's protocol, what satisfies the characteristics of a NIZK is the following.

Completeness:
When the proof holds true, the verifier can be assured of its accuracy by the prover.

Non-Adaptive Soundness:
If the proof isn't true, the prover cannot make the verifier accept it as such.

Adaptive Soundness:
If the proof isn't true, the prover cannot make the verifier accept it as such, even when the prover makes a proof after seeing the crs.

Non-Adaptive Zero-Knowledge:
The verifier only learns about the proof's truthfulness and no additional information.

Adaptive Zero-Knowledge:
Even the best attacker can not make any proof distinguishable from another after seeing the crs.

End of definition.

In many cases, a Non-Interactive ZKP can be obtained by applying the Fiat-Shamir transformation to an Interactive ZKP.

6.Composition and Characteristics of NIZK

NIZK (Non-Interactive Zero-Knowledge) is a protocol within ZKP (Zero-Knowledge Proof) that has the characteristic of reducing the computational cost for the verifier while increasing the computational cost for the prover. Also, the communication costs for both sides have been significantly reduced. Many developers are trying to improve the high computational cost of the prover. At the same time, many are attempting to ensure that the verification cost remains within a tolerable range, even on inherently inefficient computational systems like blockchains.

Given these characteristics, it's a system that offers significant benefits to both verifiers and provers, which can be summarized as:

Verifier: Low computational cost, ability to verify input or properties of the prover.
Prover: High computational cost, can undergo verification while avoiding disclosure of sensitive or unnecessary data.

It's evident that while the verifier gains pure efficiency, the prover benefits from privacy and efficiency through data omission.

Many methods make up the properties of zkSNARKs, a representative of NIZK. While there are many types of zkSNARK schemes, most of them commonly follow this structure:

1)Circuit: A circuit representing constraints related to the input.

2)Transformation: Converts the circuit into polynomial problems or SAT problems.

3)Proving: The prover demonstrates that they know the solution to the transformed problem.

4)Verifying: The verifier checks the proof and decides to accept or reject it.

1)and 2) are sometimes referred to as the frontend, while 3) and 4) are called the backend.

-Regarding Trusted Setup:

In some NIZKs, when generating the Proving key or Verifying key from the circuit, if one abuses the process information, there's a possibility to forge the proof and lose the zero-knowledge property of the proof. This kind of information is called "toxic waste," and sometimes circuit creators are obliged to discard this information. There are mechanisms, like ptau ceremony, to ensure that toxic waste can't be abused

unless all participants collude. It's a system where multiple people produce toxic waste.

There are cases where this toxic ceremony is produced for each circuit and cases where it can be reused across various circuits worldwide, with the latter particularly called the Universal Trusted Setup. In practical applications, with mechanisms that allow this, such as PlonK, security can be significantly enhanced, leading to the state where having a Trusted Setup does not directly compromise Trustlessness.

7. Recursive zkSNARKs

In zkSNARKs, when a circuit that represents constraints can be expressed in a Turing-complete language or in a generic manner, it becomes possible to write the code for zkSNARKs verification within that circuit. This means that within a zkSNARKs proof, the verification of another zkSNARKs' Proof can be incorporated.

Such Recursive zkSNARKs are vital in many zkSNARKs applications because of improved versatility, the ability to split and parallelize proof computations, and the ability to customize data confidentiality. As a result, most of the ZKP libraries used in recent applications have this recursive feature implemented.

Previously, this recursive characteristic was seen as rare and challenging. The difficulties can be described as follows:

Consider the task of performing the computation for zkSNARKs verification, specifically the pairing computation, within a circuit. The pairing computation itself is a calculation in the context of elliptic curves. However, when executed within the circuit, the calculation is done within the bounds defined by the elliptic curve. Just like how 9 behaves differently in F5 compared to F7, calculations might not match due to discrepancies, causing a loss of information and making verification impossible.

Given this, it might seem preferable at first to compute the pairing within a ZKP circuit using elliptic curves that have matching orders. But this approach results in curves that are vulnerable to the SSSA attack. A solution was to create two sets of elliptic curves. The first set, E1, has an order of p and a characteristic of q. The

second set, E2, has an order of q and a characteristic of p. The challenge was to find a pair in which the order from E1 would match the characteristic from E2, and vice versa. This is where the Pasta Curve comes into play. Even though the order and characteristic aren't identical, by using the alternating nature of these numbers, it became possible to achieve recursion without repetition.

In contrast, zkSTARKs doesn't rely on elliptic curves, so there's no risk of the SSSA attack. This makes it feasible to construct a Recursive ZKP that can undergo any number of repetitions within the bounds of a single finite field.

7.1 On Cyclic Recursive ZKP

Even when performing Recursive ZKP an unlimited number of times, usually a different ZKP circuit is required for each layer, necessitating the preparation of a circuit for each of those layers. However, when the proof being verified originates from that very circuit, it is possible to generate an infinite hierarchical Recursive Proof using just a single circuit. This is called a Cyclic Recursive ZKP. In the case of a Recursive ZKP, it is customary to embed a verifying key within the circuit. However, for Cyclic Recursive ZKP, it is not possible to embed its own verifying key when creating the circuit, so this issue is addressed by verifying the identity of the circuit.

Here are the defining features of Cyclic Recursive ZKP:

- A proof P, produced by a given circuit C, can be reintroduced into C to generate the subsequent proof P'.

8. Classification of NIZKs

The development of practical NIZKs, such as zkSNARKs and zkSTARKs, has been taking place over the past decade, notably beginning with the Pinocchio Protocol in 2013, and greatly propelled by Groth16 in 2016.

In the Pinocchio Protocol, constraints represented in the R1CS format were converted to QAP, achieving verifiable computing. Although it lacked properties of confidentiality and non-interactiveness, Groth16 effectively addressed these issues, becoming, in practice, the first widely-used NIZK system.

While Groth16 utilized pairings, zkSTARKs in 2017 replaced this with FRI, leading to a significant improvement in the computation speed of proof creation. In 2019, PlonK addressed a major inefficiency in Groth16's R1CS, where the polynomial degree increased inefficiently for constraints (Copy Constraints) using the same variables. This inefficiency was dramatically optimized using a technology called coordinate accumulator. Additionally, Ultra PlonK made it possible to write constraints in higher-degree polynomials, allowing for the assembly of various free gates and custom gates.

Subsequent NIZK developments have mainly focused on the previously mentioned achievement of Recursive ZKP. Notable systems include Redshift, a combination of FRI and PlonKish, Plonky2, Halo2 which backs Ultra PlonK with IPA, and Nova, which can sum the computational processes of R1CS. Plonky2, renowned for significantly accelerating the generation speed of Recursive ZKP, achieved faster processing performance by reducing the number of finite fields in the definition body while ensuring security did not fall below 100 bits. By expanding the definition body and increasing the dimensions, it was made possible to perform individual computational costs within a smaller finite field.

8.2 Backend

On Pairing and PCS

Pairing refers to the operation between points on the elliptic curve, characterized by its bilinearity. The bilinearity can be expressed as:

For points P, P', Q, Q' on the twisted elliptic curve group concerning pairing: en(P+P',Q') = en(P,Q) en(P',Q) en(P,Q+Q') = en(P,Q) en(P,Q')

en(P,P) = 1

en(P,Q≠O)=1 => P=O

This can verify equations containing both addition and multiplication once each, allowing calculations to be performed while data remains confidential. This meets the requirements of zkSNARKs. Specifically, in the QAP format after converting polynomial problems in zkSNARKs, it allows for the verification of polynomial equations at a single point while the polynomial remains concealed.

For A(x)B(x)-C(x) = Z(x)H(x), verification is conducted at point x=s. In this instance, the equation can be resolved with a single multiplication, hence the use of pairing.

For more flexible polynomial equations like in PlonK, Polynomial Commitment Schemes (PCS) are employed. Unlike regular bit commitments, polynomial commitments can commit to functions, allowing for the insertion of various values to obtain flexible outputs.

About FRI

FRI is an algorithm to ascertain and verify knowledge of a polynomial. It leverages the difficulty of interpolating a high-degree polynomial using a small number of points for those unfamiliar with the polynomial.

Example: For values between 0 and n, if a polynomial satisfies the condition P(Q(x)) = 0, then P(Q(x)) can be represented as Z(x)H(x).

To prove knowledge of such a Q(x) when P and Z are known, one needs to demonstrate: "I am aware of several points on Q and the corresponding points on H for the same x values."

However, given Z is known, for each point x=s, one can compute P(t) and divide it by Z(s). This allows even arbitrary values of t to satisfy the equation. Thus, proving these points lie on the same polynomial becomes essential. The difficulty of ensuring many sample points belong to a lower-degree polynomial unless you're aware of that polynomial forms the core of the security concept. This concept is foundational to FRI.

Among mechanisms that ensure ZKP's security, FRI is rare in its quantum resilience and does not depend on security assumptions like the discrete logarithm problem.

About the Folding Scheme

In R1CS, the expression a times b equals c is termed an instance. Combining different instances born from different witnesses within the same circuit, like through addition, is tricky. Yet, this becomes possible when applying an extended R1CS called Relaxed R1CS. This method of merging two instances into one is termed the Folding Scheme. By continually merging two into one, the final proof becomes highly compact.

About the sumcheck protocol

It's a succinct zero-knowledge proof protocol. By repeatedly substituting boolean values and random values into the target polynomial, one obtains the proof. It's employed within the Folding Scheme.

About IPA

IPA is a security mechanism for zero-knowledge proofs, reliant solely on elliptic curve point computations. Suitable for straightforward computation verifications like Bullet Proof, it's also used in blockchain…

8.3 *Front-end*

About R1CS

R1CS is the foundational ZKP form, where the constraints a circuit must satisfy are denoted as a times b equals c.

About Plonkish gate

Plonkish denotes constraints of circuits presented in pioneering papers like PlonK or its extensions. Initially denoted as XXX in the paper, it can transition to higher-degree polynomials, and these flexible constraints for high-degree polynomials are dubbed custom gates. PlonK encompassing custom gates is commonly called Ultra PlonK.

About lookuptable

It's an Ultra PlonK feature, drawing inspiration from digital circuit lookuptables. It's essentially a truth table correlating input and output bits. Instead of conducting intricate bit computations unsuited for PlonK, it directly fetches the resultant boolean value, thereby downsizing the circuit. This is notably valuable when calculating hashes using PlonK.

About AIR

It's a format to inscribe constraints of zkSTARKs-provable polynomials.

About CCS

CCS is a constraint system crafted to simultaneously depict R1CS, Plonkish, and AIR.

8.4 *Classification*

Using the front-end and back-end, they can be classified as follows:

ps) Pedersen commitment

front \ back	Paring/PCS	FRI	Sumcheck	IPA
R1CS/ Relaxed R1CS	Groth16/ Marin/ Sonic	Aurora	Nova Spartan	
Plonkish	PlonK	Plonky2/ Redshift		Halo/Halo2
AIR		zkSTARKs		
CCS			HyperNova	
Simpler one				Bullet Proofs

9 Tools for ZKPs

1. Library
2. language
3. security
4. IDE
5. acceleration (GPU/ASIC)

Bellman
Bellman is an initial Rust library that made it somewhat straightforward to implement zero-knowledge proofs. With it, Groth16 circuits could be implemented. Later community forks incorporated support for PlonK, among others.

- Circom
  Circom is the most widely-used dedicated language for zero-knowledge proofs, capable of outputting R1CS. In tandem with snark.js, it allows for easy generation of zero-knowledge proofs. Both Groth16 and PlonK are supported.

Additionally, compilations from Circom to various zero-knowledge proofs like zkSTARKs or Halo2 are separately supported by diverse communities.

- Cairo
  Cairo is a language dedicated to zkSTARKs and compiles to AIR.
- Lurk
  Lurk is the most suitable language for producing Nova outputs.
- Winterfell
  Winterfell is an open-source zkSTARKs library in Rust.
- Solidity/zkEVM
  Originally a language for smart contracts, Solidity, with the introduction of a new compilation environment, can now be translated into various zkEVM circuits for diverse zero-knowledge proofs. Translations to Halo2 and PlonK are particularly renowned.
- Aleo

10 ZKP Performance

When evaluating the performance of ZKPs, it is clear to consider the following basic criteria:

1) Shortness of proof generation time relative to the number of constraints.
2) Shortness of verification time.
3) Smallness of proof data relative to the number of constraints

As a note, the proof generation time is generally constant in each ZKP and does not depend on the size of the circuit. Each of these metrics can be hard to quantify definitively since they can be parallelized and made redundant by Recursive Proofs. Whether Recursive Proof is feasible can also affect these metrics.

With this in mind, generally speaking, if we disregard Recursion, a negative correlation can be observed between proof generation time and proof data size.

A negative correlation is also observed between the shortness of verification time and the shortness of proof generation time relative to the number of constraints. For instance, in cases like FRI, a trade-off can be manipulated with parameters. This means it's adjustable whether to burden the verification side or the proof side.

From a security standpoint, whether a trusted setup is needed, or if a global one-time trusted setup suffices, or if it's unnecessary, can also be considered a performance criterion.

11 Vulnerability of ZKP

https://github.com/0xPARC/zk-bug-tracker

1) Parameters

Even if the ZKP scheme itself is secure and has a security proof, depending on how the parameters are set, a vulnerable ZKP proof can emerge. There's often a significant incentive in many cases to set parameters that might decrease security for purposes like speeding up proof generation.

Example:

2) Mistake in the Fiat-Shamir Transformation

There are instances where vulnerabilities arise when converting to non-interactive proofs. Moreover, it's not uncommon for the security degradation in the Fiat-Shamir process to go unchecked. Discussions frequently revolve around a conjectured security, which assumes the post-transformation bit security is the same as the pre-transformation bit security.

Example:

3) Trusted Setup

For ZKP schemes that require a trusted setup, if the toxic waste is misused, fake proofs might get verified, or secrets could be leaked. Especially concerning fake proofs, due to the entire scheme's zero-knowledge nature, external observers cannot detect the fakes. Therefore, for protocols employing zero-knowledge proofs for privacy, detecting attacks and estimating their scale is challenging, potentially leading to catastrophic long-term damage. Using ZKPs without a trusted setup or conducting a universal setup is decisively crucial for such applications.

4) Mistakes in Circuit Design

Even if the ZKP is secure, those designing circuits using it can introduce vulnerabilities at that stage, which is quite common. A typical example in Circom

would be confusing substitutions and constraints. Mistakes in circuit design, much like bugs in smart contracts, are expected to be frequent. Consequently, in the cryptocurrency industry, there's a thriving practice of code audits specifically for ZKPs.

12 Applications of ZKPs

About ZCash:

ZCash, launched in 2016, is a cryptocurrency focusing on privacy, aiming to completely conceal the sender, receiver, and transfer amount. It was one of the first attempts in cryptocurrency to employ zero-knowledge proofs, using Groth16.

About pay2sudoku:

This was an application experiment on Bitcoin using zero-knowledge proofs by core developer Gregory Maxwell. It allowed someone who knew the solution to a Sudoku puzzle to prove it using a zero-knowledge proof and receive coins. Notably, since the Bitcoin script doesn't have a function to verify zero-knowledge proofs, the Sudoku challenge was guaranteed using a zero-knowledge proof by substituting knowing the solution to the puzzle with knowing the preimage of a hash. Therefore, unlike conventional zero-knowledge proofs, where the circuit creator doesn't necessarily need to know the answer that meets the constraints, the pay2sudoku scheme presupposes that the challenger knows the solution to the Sudoku.

About Tornado Cash:

Tornado Cash is a mixing application on Ethereum, created with a focus on privacy like ZCash. It was written and deployed with Circom and was the first zero-knowledge proof application to attract many users.

How it works: https://github.com/tornadocash/tornado-core?tab=readme-ov-file

About Aztec Protocol/zkMoney:

zkMoney is a private cryptocurrency on Ethereum developed by the Aztec team. The main distinction from ZCash is whether it's Layer 1 or an application on Layer 1. Ideally, expressing it in ERC20 would be optimal for user-friendliness, but this is technically challenging, so it was implemented using an entirely new token standard.

About Mina Protocol:

Mina is a Layer 1 focused on zero-knowledge proofs and has an extremely small blockchain size, even smaller than Bitcoin's. It uses Cyclic Recursive ZKP, making the entire blockchain verification instantaneous. Similar methods could also be applied to Bitcoin's full nodes, and several such projects are currently identified.

About ID & KYC:

IDs using ZKP and blockchain are often referred to as DID/KYC. They range from those complying with regulations like KYC to non-compliant ones. Project Zuzalu, which began under the concept of a Network State aiming for a position equivalent to a nation by appointing diplomats to a DAO, used ZKP for its virtual nation member passports.

About Selective Disclosure:

With ZKP, selective disclosure can be implemented by introducing a trapdoor in the proof. Digital credentia

About Credit Score:

About NFT:

Since NFTs serve as proofs of ownership, their owners' addresses and on-chain histories are easily trackable by many. ZKP is applied to anonymize possession and address this issue.

About Data Availability Sampling:

Conceived by Vitalik Buterin for the purpose of completing zkRollup and detailed by Bankrad Fiest, Data Availability Sampling is a mechanism to prove data retention for a certain period. It allows nodes to efficiently prove that they're holding data fragments, utilizing two-variable polynomial commitments.

About Bridges:

With ZKP, it's possible to construct trustless 2-way peg bridges between many public blockchains. However, both chains must be capable of verifying ZKP. If generic smart contracts are implemented, this condition can be automatically met. Conversely, a trustless bridge between Ethereum and Bitcoin using ZKP hasn't been discovered yet and is said to require Witness Encryption.

About zkML:

zkML refers to systems that guarantee, through NIZK, that a model was constructed using a specific algorithm and dataset.

About AML:

The "Proof of No Crime" concept, proposed and implemented by Tornado Cash, is a mechanism that, while possessing the properties of mixing, allows one to prove that no funds were received from a specific address. It's a system that seeks a balance between privacy and AML.

Secret Voting:

Games:

zkOracle

funny category:

12.1 zkRollup

12.1.1 General

・Definition of zkRollup

zkRollup is a network that secures the bridge between Layer1 and off-chain using ZKP while retaining the security of Layer1.

・History of zkRollup

It was conceived by Vitalik Buterin in 2018, and currently, there are more than 10 networks that are its improved versions. Before Vitalik Buterin's idea, there was a proposal called Plasma snap which applied zkp to the Layer2 solution Plasma, and Barry Whitehat performed the MVP implementation. In 2019, Loopring first implemented and launched a DEX using zkRollup, and in 2020, zkSync launched the remittance network zkSync v1. In 2023, the Polygon team developed zkEVM and launched a zkRollup fully compatible with Ethereum VM, i.e., Solidity.

・Classification of zkRollup

zkRollup can be classified into the following three based on the uniqueness of the state:

Stateful zkRollup - Manages bit commitments of the state of users and smart contracts with Layer1 smart contracts. Examples: Starknet, zkSync Era, Polygon zkEVM, Scroll, Taiko, Loopring

Succinct zkRollup - Allows everyone to calculate bit commitments of the state of users and smart contracts from Layer1 data. Example: Sovereign Rollup

Stateless zkRollup - Users only compute their own bit commitment without fetching from Layer1 data. Example: Intmax

Furthermore, the most prominent zkRollup, the Stateful zkRollup, can be classified as:

Type1 Exactly equivalent to the Ethereum Network. Example: Taiko

Type2 Fully compatible with EVM bytecode but not equivalent to Ethereum Network. Example: Scroll

Type3 Mostly compatible with EVM and many Solidity codes are portable. Example: Polygon zkEVM

Type4 Compatibility with EVM is not guaranteed, and there's no guarantee that many Solidity codes are portable. Examples: StarkNet, zkSync

These classifications by zkEVM are important in the Ethereum community due to their impact on security during porting from Solidity code.

Additionally, there are zkp solutions that do not meet the requirements of zkRollup:

・zkPorter

A solution proposed by zkSync to sacrifice zkRollup security to achieve scalability. It was proposed to offset the DA cost bottleneck in zkRollup by a new token-carrying network, eliminating the need to pay Ethereum Layer1 DA costs. A DA attack by a coalition of 2/3 token holders can freeze user assets but cannot confiscate them, so there is no incentive for the attacker. Since the security assumption of a new token is included, it does not inherit the security of Ethereum Layer1.

While not classified as zkRollup, it's debatable whether it's a Layer2 or a sidechain:

・Validium

A system developed by zkSync. It is a mechanism that trusts a validator group not to carry out a DA attack, foregoing DA cost payments.

・Volition

Proposed by Starkware, it is a mechanism where individuals can choose the trustlessness level, i.e., security level, of DA storage. There's a trade-off between price and security.

-zkRollup Concerns and Solutions

11.1-2 Rollup != Bridge Argument

If a token is natively issued on Rollup (Layer2) and is not the subject of a bridge with Ethereum Layer1, there has been a claim that this token may not inherit Ethereum Layer1's security if it conducts its own forks or governance.

However, even tokens on Ethereum Layer1 that have their governance or references with projects, or lack liquidity, often do not represent Layer1's trustlessness. It was suggested that Rollup inherits not only the security of Ethereum Layer1 but also the reduced security properties by each community, affirming Rollup's role as a pure extension of Ethereum. Currently, this argument isn't seen as very valuable, so it rarely comes up.

11.1-3 Does zkRollup use Zero-Knowledge?

Most zkRollups are not confidential. Therefore, some argue that zkRollups only utilize the non-interactive nature of zkSNARKs. The validity of this claim boils down to:

"When there are many inputs that, while not a security issue if made public, are kept confidential for efficiency, is ZK being utilized?"

In other words, in Stateful zkRollups, the prover hides data not because they want to but because making it public wastes computational resources.

11.1-4 Decentralization of zkRollup's block producers

Many Stateful zkRollups have not decentralized block producers due to the high computational cost of zkp and the risk of bugs. While this doesn't pose a security risk given zkRollup's inheriting properties, it does raise concerns about service continuity.

11-1-5 Potential for Decreased Security via proto-danksharding/danksharding

The use of time-limited data, such as those provided by proto-danksharding, for periods like one month, is a very novel approach in the blockchain industry. With this, while effectively preventing block withholding attacks, rollup nodes can transfer Data Availability (DA) from Layer1. This is possible within rollups because if even one node is an honest node, it can prevent a DA attack. This is known as the 1/N security assumption, and it's considered a method to scale Layer2 trustlessly. Whether writing all data into Layer1's call data or moving the data storage to rollup nodes through proto-danksharding, both approaches operate on the 1/N assumption. However, the value of N differs significantly between them. When writing to Layer1, N might be the number of Ethereum nodes, say 5000, but when using proto-danksharding, N might be the number of candidate nodes, perhaps around 20.

For major Layer2/Rollups with high public trust and significant value concentration, maintaining a large N might be possible. However, for rollups crafted through mechanisms like "Rollup as a service," it's challenging to predict what this N number might be.

13. Security considerations

This document has no security considerations.

1. 2FA
2. AI
3. QC
4. bit
5. conjecture
6. formal method

14. Privacy considerations

The acknowledgement section contains PII of those people. Editors SHALL make sure to obtain the consent of the people to be included. Sometimes, a contributor MAY want to remain pseudonymous and just appear as an initial etc.

15. Regulatory considerations

1. How law enforcements use zkp for verification legally
2. Audit
3. Credential
4. anonymity revocation (!= proof of crime)

Legal Consideraiton

16. Informative reference

This document has no informative reference.

Appendix A – Acknowledgement

(Informative)

A.1 Editors and Co-editors

- Leona Hioki
-

A.2 Contributors

- Nat Sakimura (NAT.Consulting)