CS-3372 Final Project Report

Group 1

Vi-AnonCred

The System for Issuing, Verifying, and Revoking the Anonymous Credential based on Smart Contract

Dat Tran (*), Cuong Nguyen, Dung Nguyen,

Khiem Nguyen, An Nguyen

Vi-AnonCred Repo (Full Codework)

https://github.com/CS-3372-Anonymous-Credential-Experiment/Vi-Anonymous-Credential

Dat Tran (*), Cuong Nguyen, Dung Nguyen,

Khiem Nguyen, An Nguyen

What is an Anonymous Credential System [0]

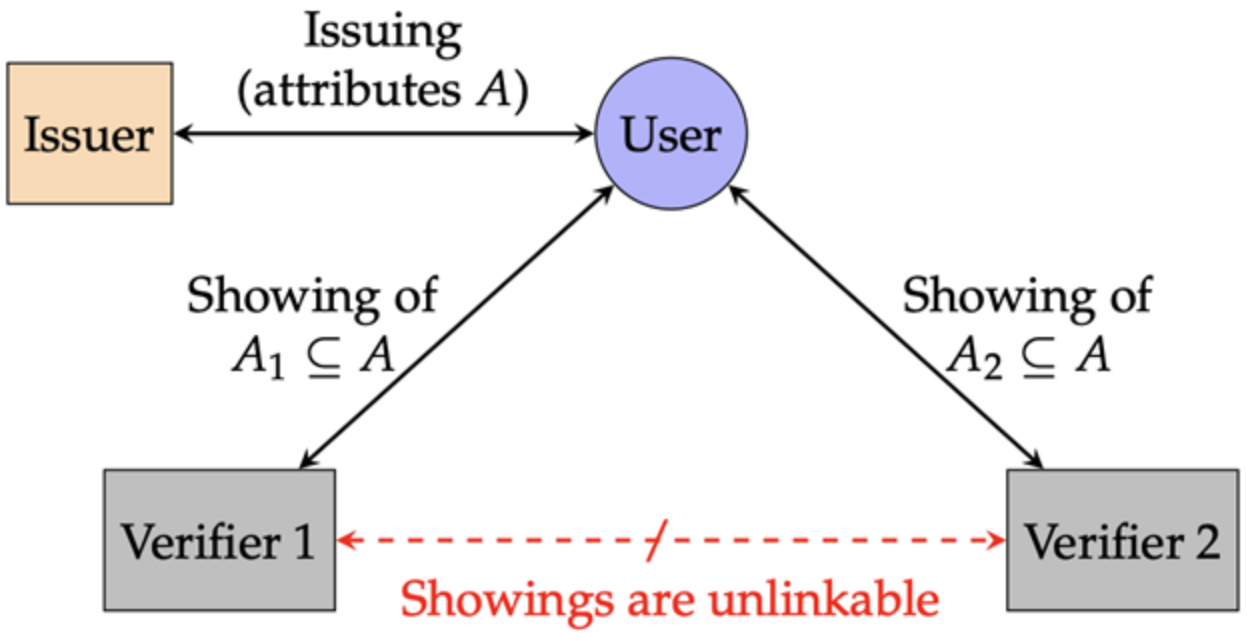


Figure 2: Abstract concept of an anonymous credential system.

What this problem focusing on

- Selective Disclosure
  - You can only know what I reveal to You,
  - But not all of my Identity,
  - My proof is Rigorously & Securely & ZK
- Credential Insurance (How Issuer …)
  - Creating my credentials?
  - Checking & updating credential Validity (with or without) reveals my identity?

What this problem focusing on

- System Construction & Security & Collusion Prevention
  - I cannot reuse, forge, or replay my proof to you
  - You cannot use my proof to map to my identity (or break it with a small chance)
- Credential Verification
  - You know that I am valid to use *your* ***service****,* but not my identity
  - How can everyone (or some people) check whether our transactions are valid without leaking our secret to them or to either of us?

Why We Need & Working on It

- Data-Minimilization [8]
  - ***(Check your validity, not your identity)***
- Cost Reduction in Data Maintenance for Application Developer

Why We Need & Working on It

- Trustless & No-Trust systems (Logistic & E-Voting system & Currency Exchange & etc.)
  - ***Verify*** your (organization) identity &
  - ***Verify*** satisfiability for ***transactions***
  - ***Without privacy invasion*** [0]

Some Terms We May Use

- Credentials: as a set of claims about its holder's identity [1]
- Verifiable Credentials (V.C): A Credential that can be verified by certain evidence [1]
- Anonymous Credential: A V.C that can be verified without leaking the holder's identity with selective disclosure [1] , [8]

Some Terms We May Use [10]

- Signature Scheme (S.S): The model of using the algorithm to create a trademark on a certain message.
- Signature: The outcome of the **Sign** algorithm using a certain S.S, which can be **Verified** later.
- Commitment: The Process of hiding data but for later proving

Some Terms We May Use [0]

- Zero-Knowledge (ZK): A special property of a proof is correct without knowing why it's correct
- Zero-Knowledge Proof (ZKP): A proof of a certain statement that is ZK
- Proof-Generation: A method or paradigm for writing a ZKP (as ZK-SNARK, STARK, Bulletproof)

Smart Contracts

['smärt 'kän-ˌtrakts]

A smart contract is a program that executes specific actions when certain conditions are met.

Investopedia

Some Terms We May Use ([0] & [11])

- Interactive Proof: Prover can interact and send a message to Verifier, and Verifier can send back
- Non-Interactive Proof: Prover only sends 1 message to Verifier only
- SNARK (zk-SNARK): Succinct Non-interactive Arguments of Knowledge
  - It's a non-interactive ZKP, but short & can be verified effectively

Abbreviations (Follow [1] & [2])

- **I: Issuers** (Who issue the credential & update the ledger & revoke a credential)
- **H: Holder** (Credential Holder & Prove generator convince the Verifier on a statement(s))
- **V: Verifier** (a Smart-Contract for checking the H's proof validity & deploying the proof on-chain)

Abbreviations (Follow [1] & [2])

- **D-Apps:**
  - Decentralized Application –
  - a type of **autonomous application**
  - not controlled by a centralized party,
  - with all assets public & accessible

- **D: Developers**
  - **(The ones who write out the smart contract)**
  - by using the data from the ledger
  - They build the D-Apps that interact with the holder.”

ZKP Proof Between H & V Include

- Proof of Being a Valid Credential or Not
  - Proof Credential being signed by issuer (Signature Verification Algorithm) [13]
  - Proof of Credential Revocability
    - **Accumulator Model (ZKP)** [4], [8]
    - **Merkel-Tree Proving (ZKP)** [5], [7]

ZKP Proof Between H & V Include

- Proof of User's Attribute
  - **Range-Proof (Bulletproof)** [6]
  - **(In)Equality Proving (out-of-scope)** [1]

Ideal Criteria of Our System [11]

- Completeness: A true statement is always convincing
- Soundness: No False Statement can be convincing
- Efficiency: You can verify my proof within a short-time & my proof is lite
- Security: You cannot use my proof on my credential to find or link my identity

CS-3372 Final Project Report

Group 1

The Revocability Protocol

Dat Tran: Accumulator-Based Design

Cuong Nguyen: Merkle Tree-Based Design

How to Handle the Revocability Problem

- Common Approaches:
  - Non-Membership Proving (Proof that your credentials have not been revoked)
  - Membership Proving (Proof that your credentials are still valid)
- In actual system deployment: Rate of Revoked Credential is low [4]

Accumulator Based Idea

- Originally suggested by [12], the Accumulator is:
  - A Collision-Resistant Hash Function accepts secret inputs, & the result of the hash returns an ***accumulated value & witness***
  - Each secret input has a **”witness”** for creating (non) membership proof of inclusion/ exclusion of the accumulator

Accumulator Enhancement

- If a Credential or Values being revoked, all witness of valid holders have to recalculate
- But Computation can be expensive & chance of revoke is small
- Make the Accumulated Values update by using revoked values (not using the valid values) [4], [13]

Accumulator Enhancement

- To simplify the calculation & accumulator construction & we work with the ***Bi-Linear Pairing Accumulator*** [14]
- We choose to work on Elliptic Curve Encryption – Instead of RSA-based (More secure & lighter accumulator size) [15]
- Why is this Secure: Strong Diffie-Hellman (q-SDH) Assumption(s) **[8]**

How the Accumulator been Constructed [4]

- Initialize the accumulated values α (no credential being revoked) using issuer's public parameter + a random exponentiation
- Initialize the set of all possible message can sign by issuer (except the secret-key): D

How the Accumulator been Constructed [4]

- new witness at time t for an values x defined as

$$\mathtt{GenWit}(\mathtt{sk}, \alpha_t, x): \quad w_{x|t} \leftarrow \alpha_t^{(x+\mathtt{sk})^{-1}}$$

- Only update the accumulated values by adding the secret of the revoked holder (x) & publish x to a public list $\boldsymbol{\delta}$

$$\mathtt{Del}(\mathtt{sk}, \alpha_t, x): \quad \alpha_{t+1} \leftarrow \alpha_t^{(x+\mathtt{sk})^{-1}}; \quad \delta \leftarrow x$$

How to creating ZKP with accumulator [4]

- Holder's shall implement the Pedersen Commitment Scheme [16] for hiding the data by doing exponentiation with randomization
- Holder would use the ***"commitment scheme"*** to hide the secret - the pair of (w,x) and use the ***commitment*** for generating ZKP

How we can generate the ZKP [15]

- ZKP was created by using the Groth16 zk-SNARK scheme
  - The Scheme was suggest for Bi-Linear Pairing & Elliptic Curve Cryptography
  - The result of the proof always constant (3 Pairing Elements) -> effective for the verification
  - The prover computation is linear-scaling with the #proving predicate & ZKP circuit (Not so ideal for prover)
  - It's Support Non-Interactive Proof (multiple Verifier can check 1 holders)

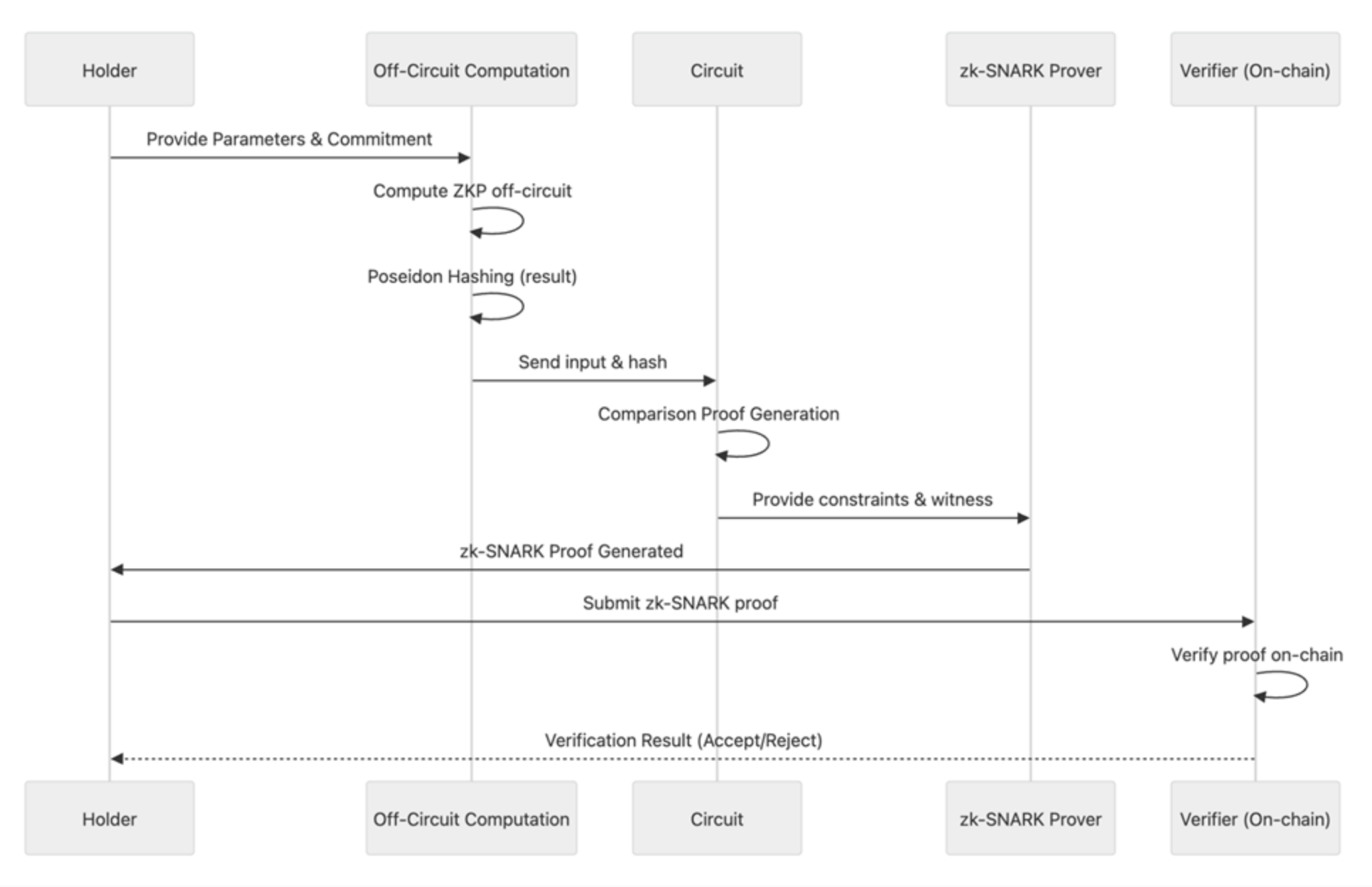
Challenge in integration to On-Chain

- The Circuit is not support Group Element Computation
- Doing Group comparison (as pairing comparison is costly) by increase
  - **#constrain & #multiplication have to do -> Raise of Gas-Cost**

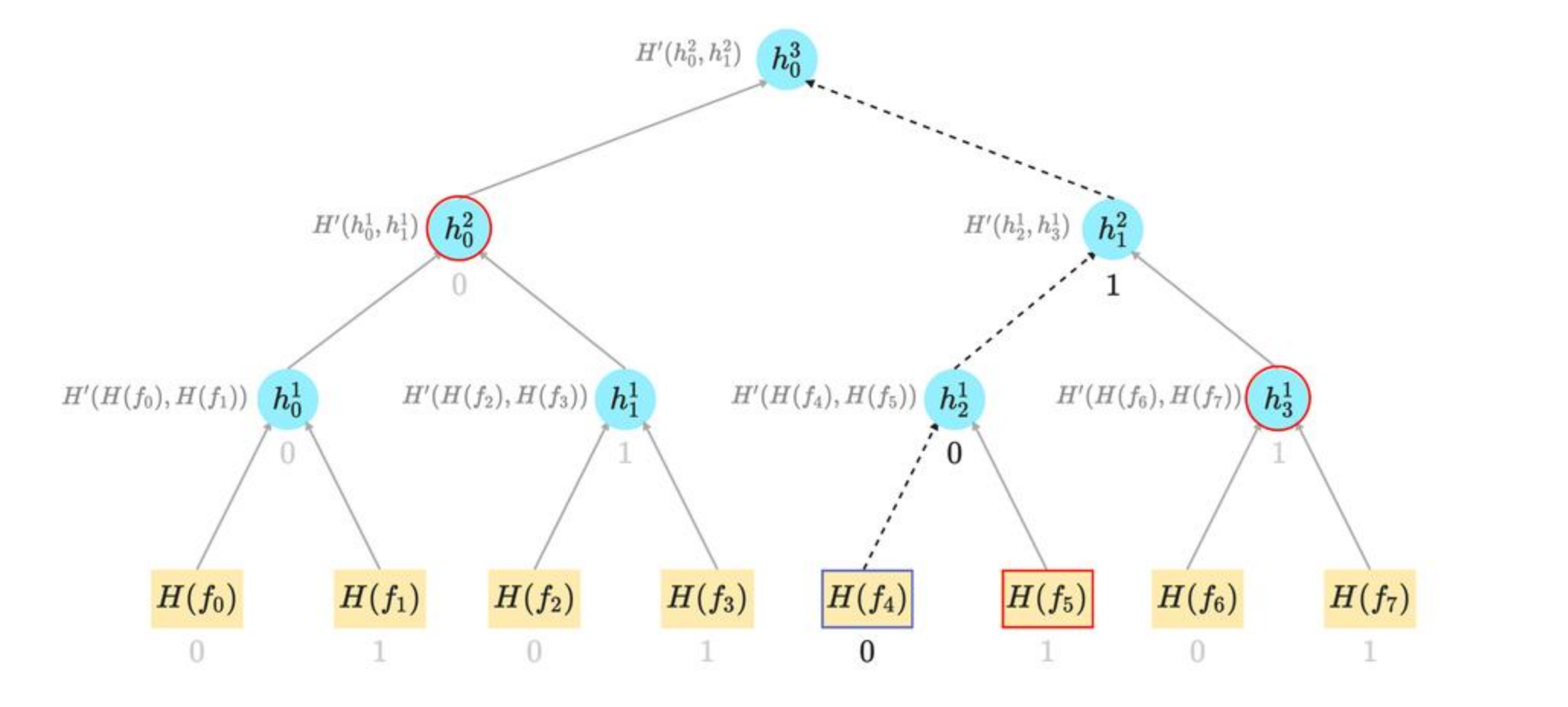
Our Solution

- Our Solution: Doing the Group-Element Computation off-chain & hashing the result by using Poseidon Hashing (or MiMC hashing) (circuit-friendly & collision resistant) [16], [17]
- If LHS of predicate == RHS of predicate (only Holder can be calculate) -> H(LHS) = H(RHS)

HOW TO BRING EVERYTHING ON CHAIN ?


Holder
Off-Circuit Computation
Circuit
zk-SNARK Prover
Verifier (On-chain)
Provide Parameters & Commitment
Compute ZKP off-circuit
Poseidon Hashing (result)
Send input & hash
Comparison Proof Generation
Provide constraints & witness
zk-SNARK Proof Generated
Submit zk-SNARK proof
Verify proof on-chain
Verification Result (Accept/Reject)
Holder
Off-Circuit Computation
Circuit
zk-SNARK Prover
Verifier (On-chain)

Index Merkle Tree-Based Design (Tuan Cuong)

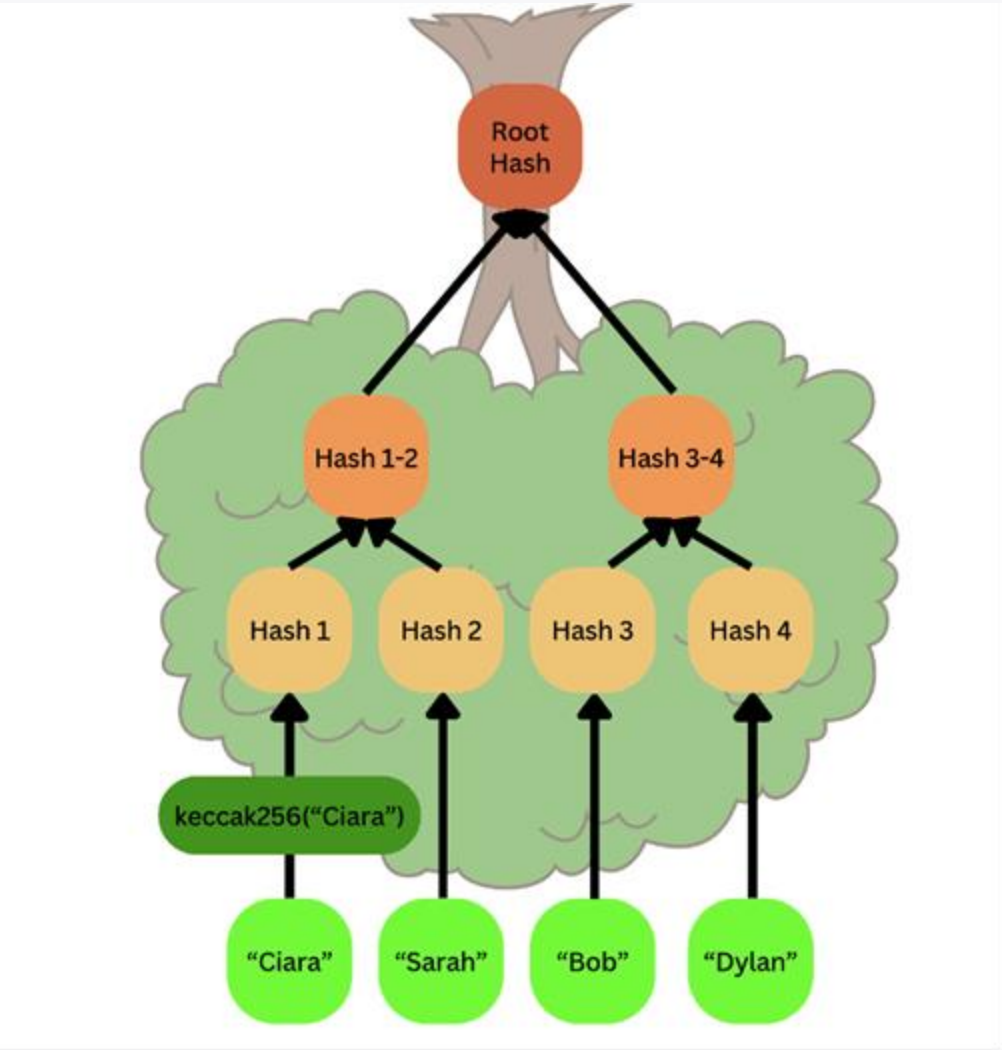


What is Merkel tree

• **Definition:** A tree structure where each **leaf node** is a cryptographic hash of its data block, and each **non-leaf node** is a hash of its direct children's concatenated hashes. [20],[22]

• **Purpose:** Authenticated data structure; ensures data integrity.

• **Root Hash:** A single cryptographic hash at the top, representing the entire dataset.



Merkle Proofs: Membership & Consistency

- **Membership Proof (Merkle Proof/Audit Path):**

  ◦ Proves a given leaf's existence in the tree.

  ◦ Consists of the leaf's sibling hash and the hashes of non-leaf nodes required to compute the root.

- **Consistency Proof (Briefly):**

  ◦ Proves a current root hash is a continuation of a previous root hash.

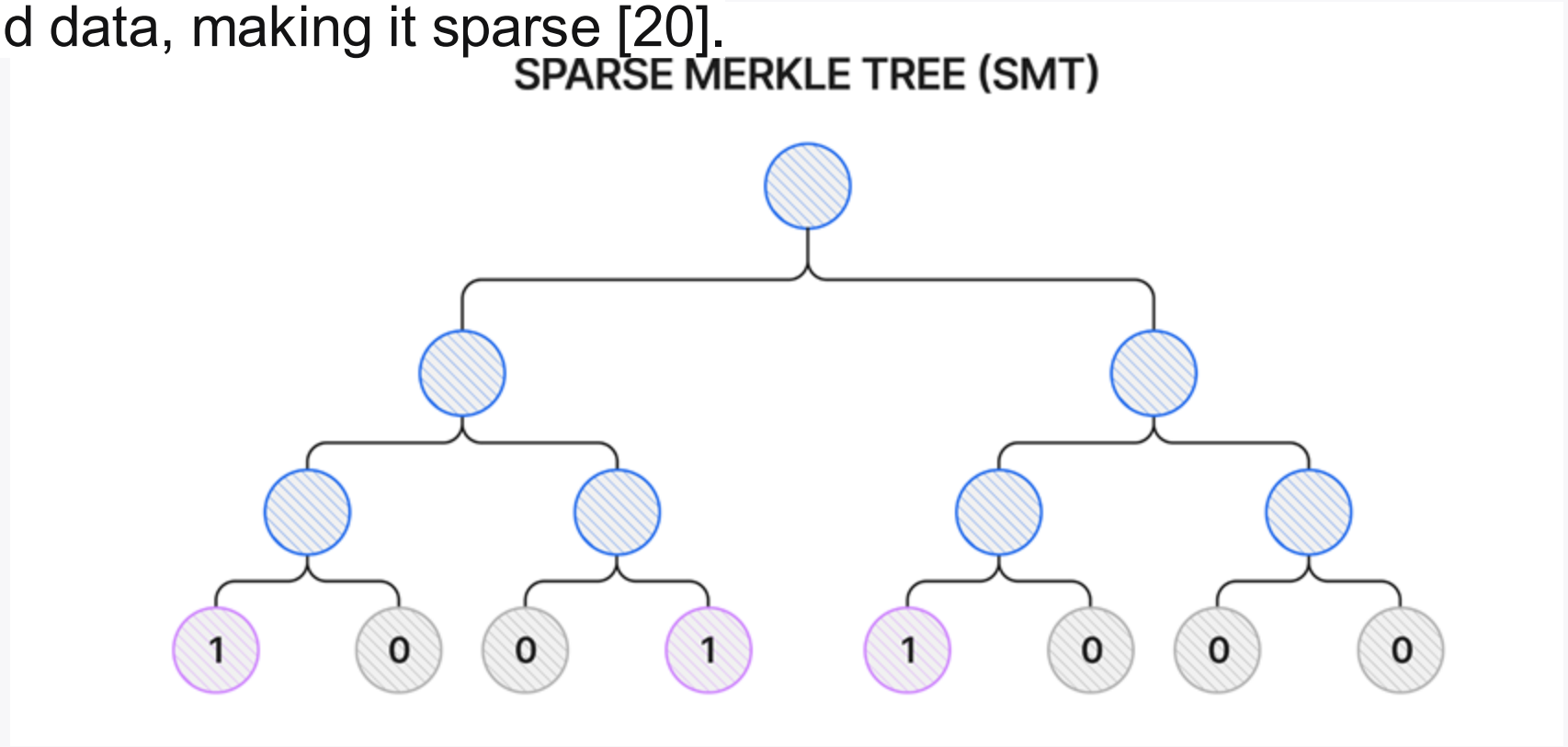
  ◦ Used to verify append-only logs.

Merkle Proofs: Membership & Consistency

- **Verification:** Recompute root hash from audit path; if it matches the known root, membership is proven. [20],[22]

Sparse Merkle Trees (SMTs) as Nullifier Trees

• **Concept:** A Merkle tree of "intractable size," where only a few leaf nodes have valid data, making it sparse [20].



Non-Membership Proof with SMT

- **Non-Membership Proof:** Retrieve Merkle path to relevant leaf; if value is 0, non-membership is proven [20, 24].
- **SMT Advantages [24]:**
  - Easy non-membership proofs (same as membership).
  - Constant proof size due to fixed depth.
  - Simple structure and append mechanism.

Non-Membership Proof with SMT

- **SMT Drawbacks:**
  - **Fixed depth leads to complex ZK circuits:** e.g., 254 hashes for 256-bit nullifiers [21, 24].
  - **High computation cost:**

Introducing Indexed Merkle Trees (IMTs)

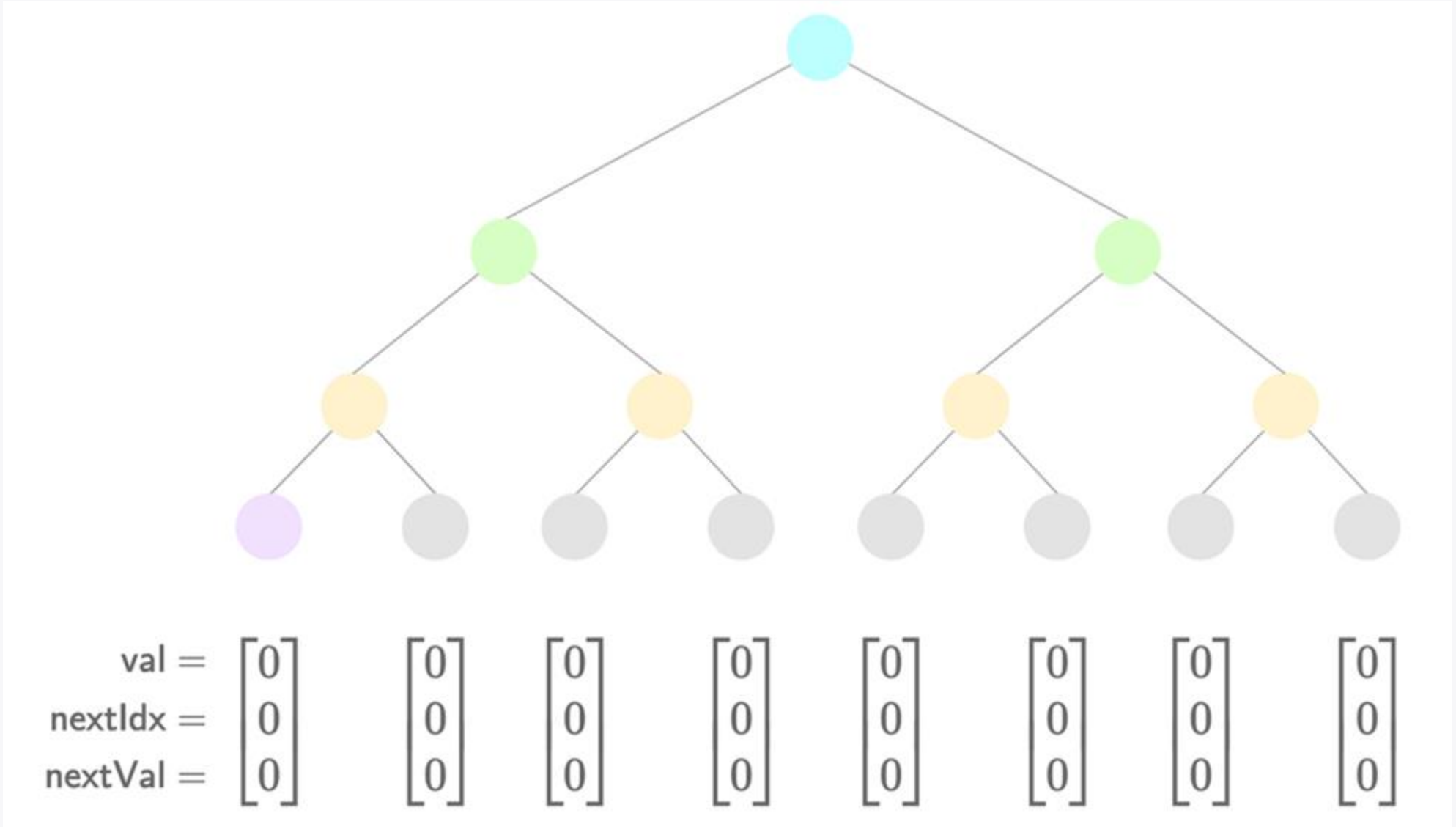
- **Concept:** A type of Merkle tree where **leaf nodes maintain additional pointers**, creating an ordered structure similar to a linked list [21, 24].

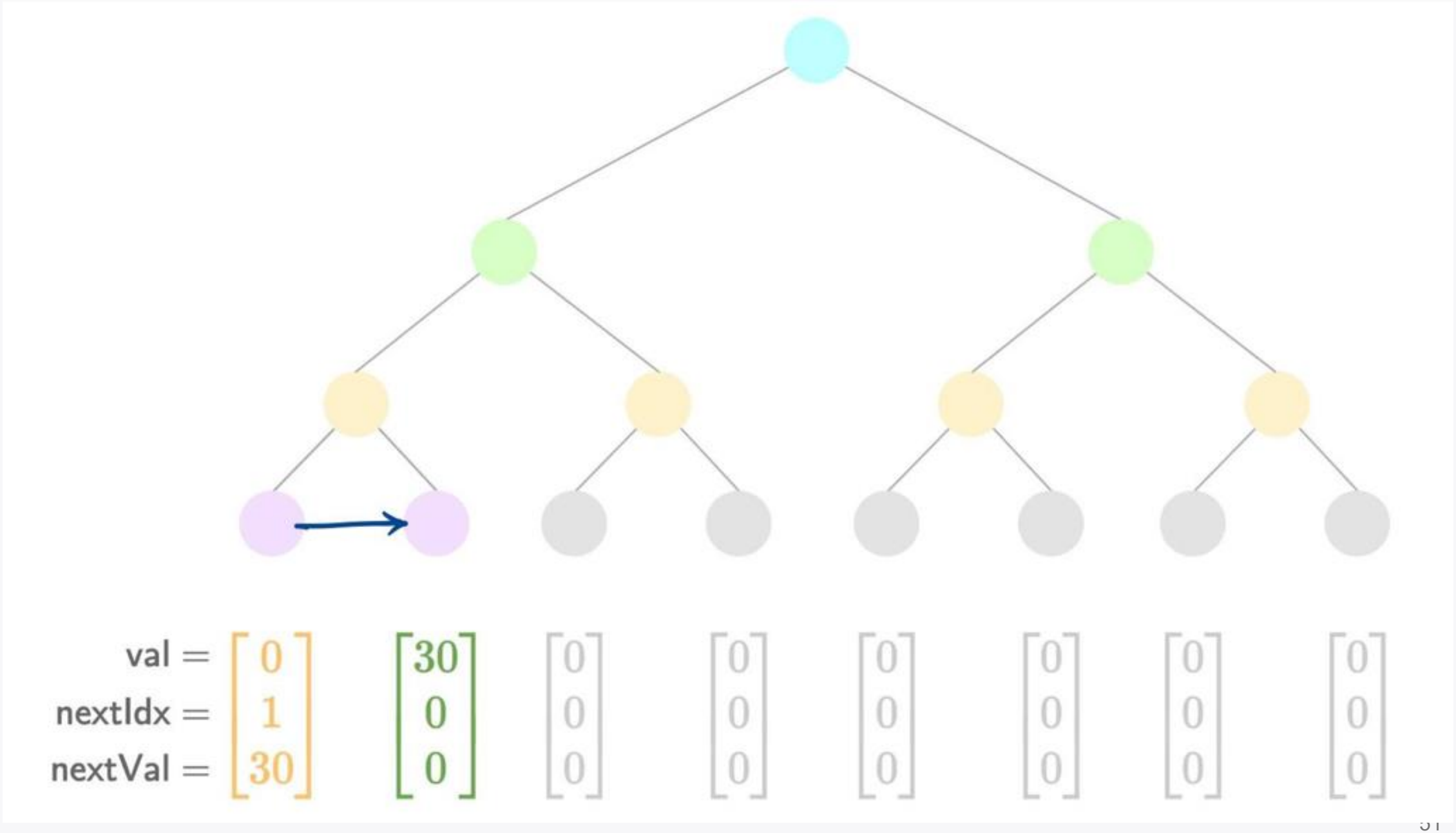
- **Key Feature:** Each leaf includes a pointer to the **next-highest value leaf**.

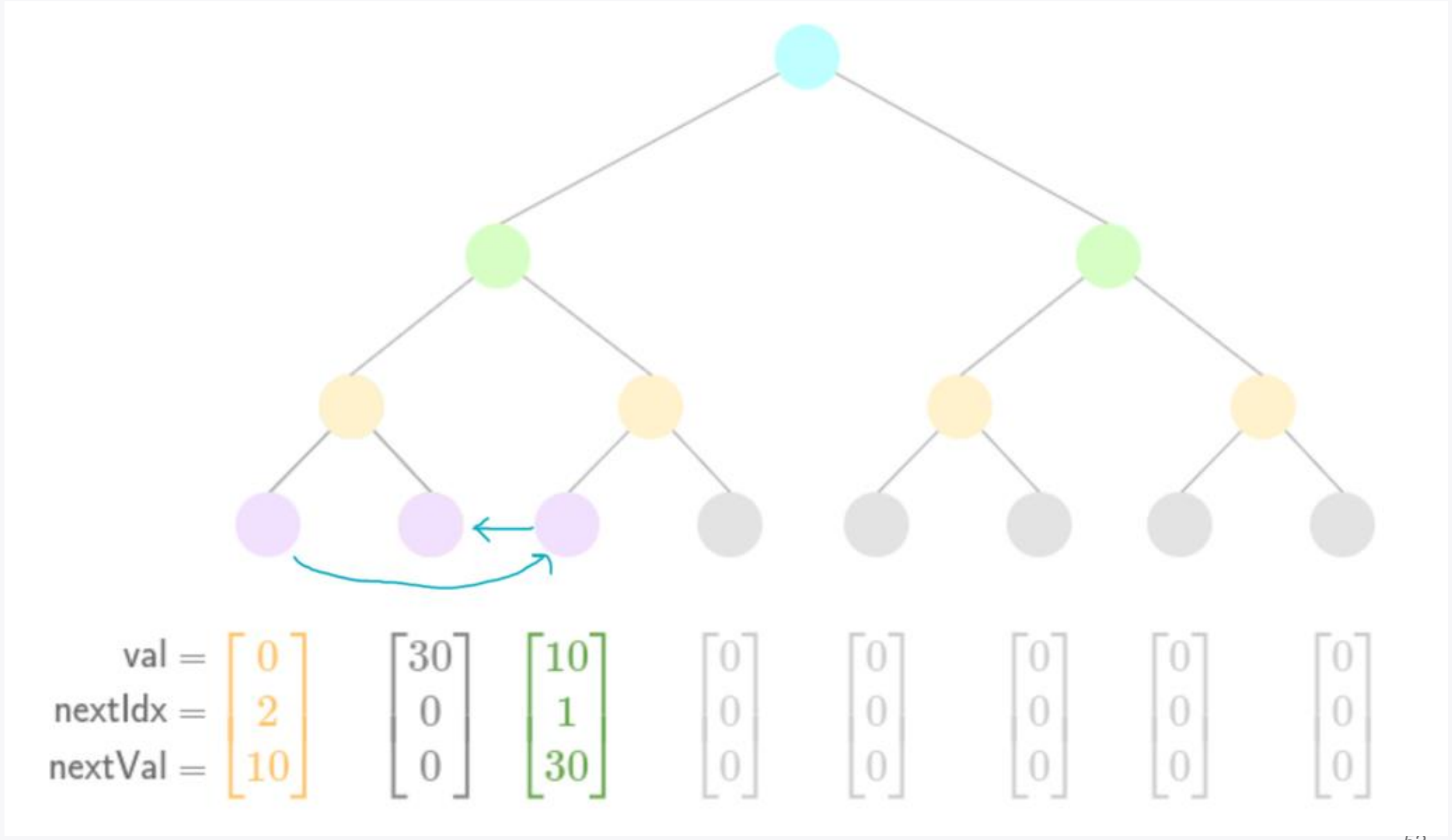
Benefit of IMTs

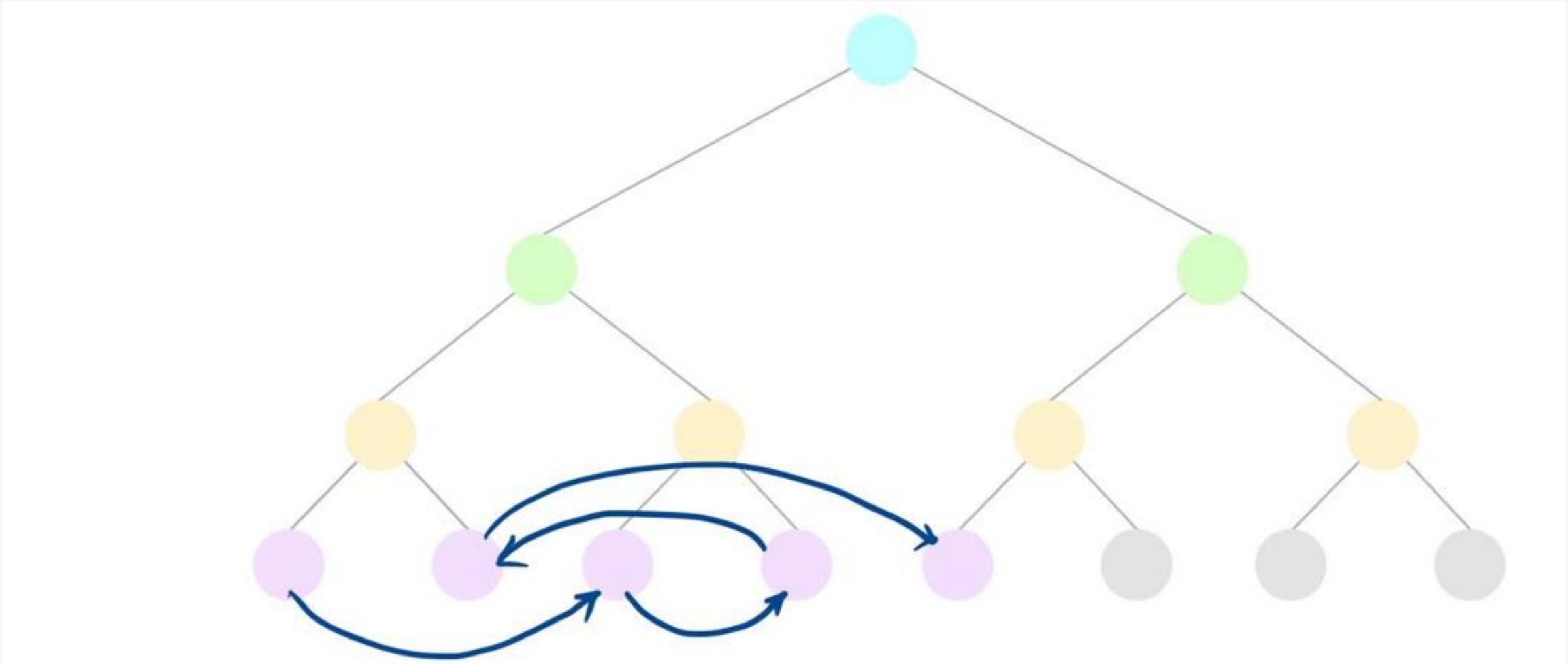
- **Benefits:**
  - Efficient **range and adjacency checks**.
  - Support for **batch inclusions**.
  - Optimized **non-membership proofs**.
  - **Dynamic growth**, unlike SMTs' fixed size.

How IMTs Work: Structure & Instruction




val = [0] [30] [0] [0] [0] [0] [0] [0]
nextIdx = [1] [0] [0] [0] [0] [0] [0] [0]
nextVal = [30] [0] [0] [0] [0] [0] [0] [0]


val = [0] [30] [10] [0] [0] [0] [0] [0]
nextIdx = [2] [0] [1] [0] [0] [0] [0] [0]
nextVal = [10] [0] [30] [0] [0] [0] [0] [0]



$$\text{val} = \begin{bmatrix} 0 \\ 2 \\ 10 \end{bmatrix} \quad \begin{bmatrix} 30 \\ 4 \\ 50 \end{bmatrix} \quad \begin{bmatrix} 10 \\ 3 \\ 20 \end{bmatrix} \quad \begin{bmatrix} 20 \\ 1 \\ 30 \end{bmatrix} \quad \begin{bmatrix} 50 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\text{nextIdx} =$$

$$\text{nextVal} =$$

IMT Non-membership Proofs

• **Core Idea:** To prove non-membership of a value `X`, reveal the Merkle path to the **'low nullifier'** (the predecessor element) [21, 24].

• **Low Nullifier:** The leaf whose value is less than `X`, but whose `next_value` is greater than `X` (or 0 if `X` is the largest).

IMT Non-membership Proofs (Cont)

Verification:

1. **Hash the low nullifier** (its value, next_index, next_value).
2. **Prove low nullifier membership** in the tree (using its Merkle path to the root).
3. **Perform Range Checks:**
    - Assert `X > low_nullifier.value`.
    - Assert `X < low_nullifier.next_value` OR `low_nullifier.next_value == 0` (for max value).

IMT Non-membership Proofs (Simplified)

- **Performance Improvement:**
  - **Smaller Tree** (no more index, so it’s shallow - 32 levels is fine)
  - **Few Hash** (from 32 hash to 254 hash; faster in computation)
  - **Faster Insertion** (8 times quicker than SMT)

We can do better : Batch Insertion in IMTs

• **Concept:** Leverage adjacency pointers to insert multiple nullifiers as a subtree in a grouped operation [21, 23, 24].

• **Benefit:** Significant efficiency increase, especially beneficial for rollup circuits.

IMT Advantages

- **Advantages:**
  - **Compact & Dynamic:** Grows with data, avoiding vast, sparse trees [24].
  - **Efficient Proofs:** Smaller tree depths mean fewer hashes per proof (e.g., 32 vs 254) [23,24].
  - **Better for Smaller Datasets:** More computationally efficient when the number of nullifiers is not at maximum capacity [24].

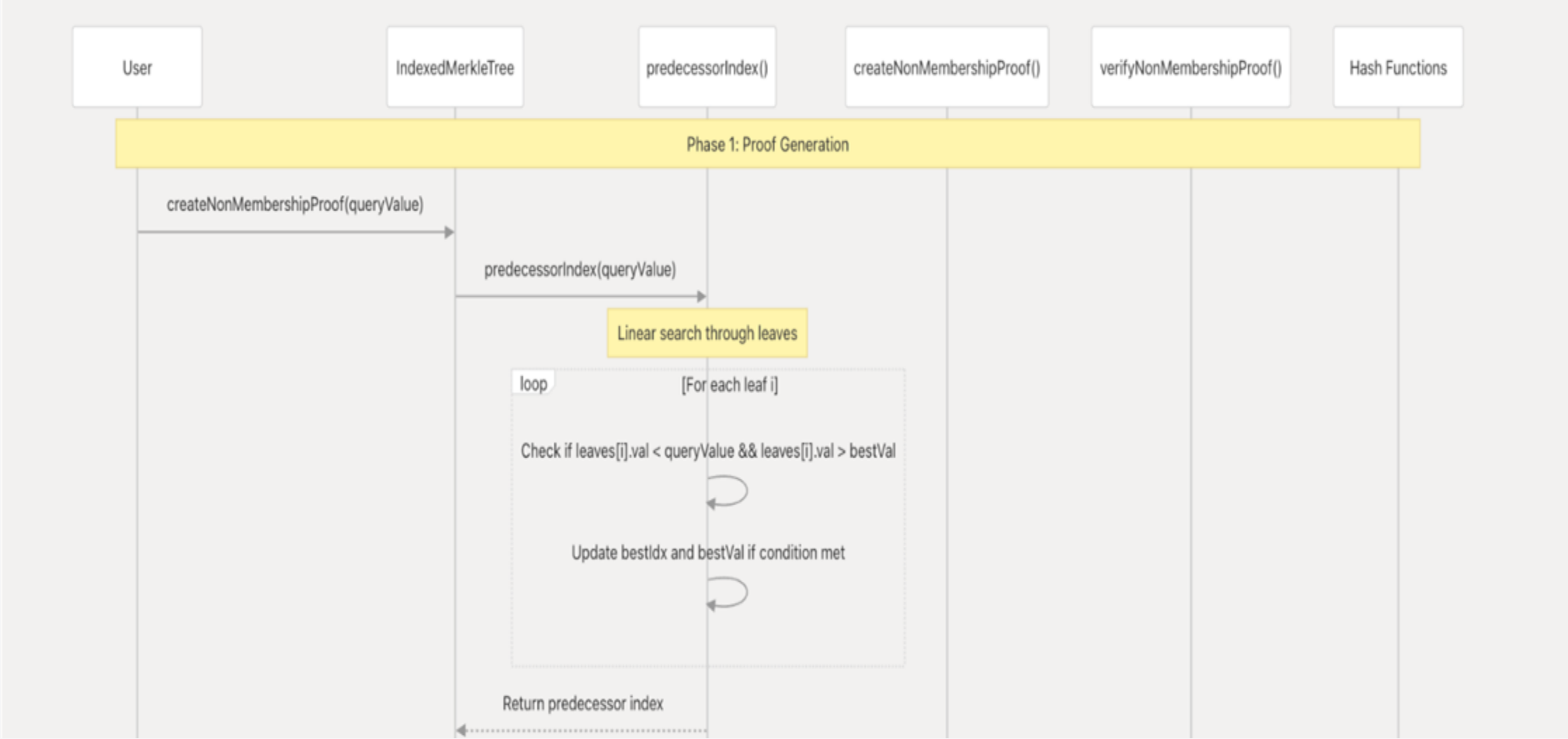
IMT Advantages (Cont)

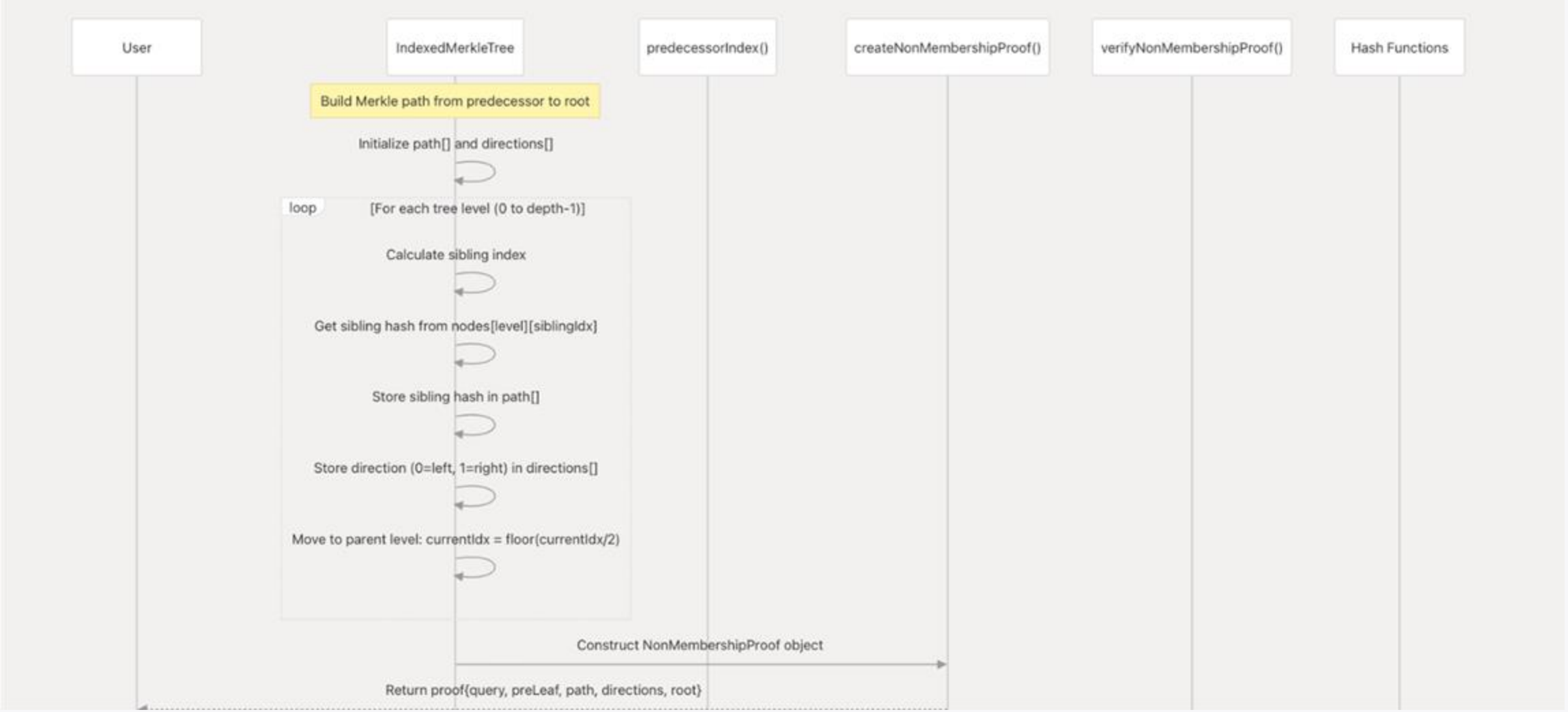
- **Batch Insertion Efficiency:** Amortized costs (Average cost/operation sequences) for multiple insertions is low [21].
- **"Engineering Fix":** Shifts computation outside the ZK circuit where possible [23].

IMT Disadvantages (Simplified)

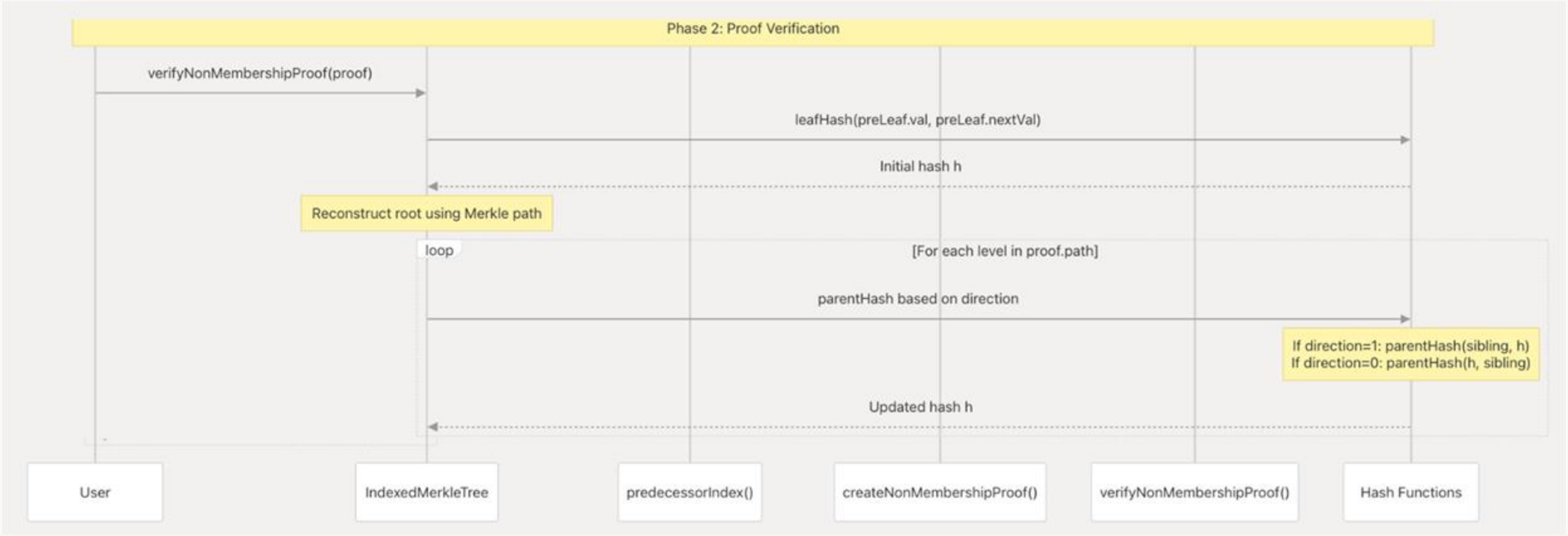
- **Disadvantages [21.24]:**
  - Implementation Complexity when scaling
  - ZK Circuit Complexity dues to adding sub-tree
  - Off-Circuit Sorting
  - Increased Storage per Node

HOW A ZKP BEING GENERATED WITH MERKLE TREE


User
IndexedMerkleTree
predecessorIndex()
createNonMembershipProof()
verifyNonMembershipProof()
Hash Functions
Phase 1: Proof Generation
createNonMembershipProof(queryValue)
predecessorIndex(queryValue)
Linear search through leaves
loop
[For each leaf i]
Check if leaves[i].val < queryValue && leaves[i].val > bestVal
Update bestIdx and bestVal if condition met
Return predecessor index


User
IndexedMerkleTree
predecessorIndex()
createNonMembershipProof()
verifyNonMembershipProof()
Hash Functions
Build Merkle path from predecessor to root
Initialize path[] and directions[]
loop
[For each tree level (0 to depth-1)]
Calculate sibling index
Get sibling hash from nodes[level][siblingIdx]
Store sibling hash in path[]
Store direction (0=left, 1=right) in directions[]
Move to parent level: currentIdx = floor(currentIdx/2)
Construct NonMembershipProof object
Return proof{query, preLeaf, path, directions, root}

HOW A ZKP BEING VERIFIED WITH MERKLE TREE


Phase 2: Proof Verification
verifyNonMembershipProof(proof)
leafHash(preLeaf.val, preLeaf.nextVal)
Initial hash h
Reconstruct root using Merkle path
loop
[For each level in proof.path]
parentHash based on direction
If direction=1: parentHash(sibling, h)
If direction=0: parentHash(h, sibling)
Updated hash h
User
IndexedMerkleTree
predecessorIndex()
createNonMembershipProof()
verifyNonMembershipProof()
Hash Functions


Perform three validation checks
Check 1: computed_root === proof.root
Check 2: preLeaf.val < query
Check 3: query < preLeaf.nextVal OR preLeaf.nextVal === 0n
alt
[All checks pass]
Return true (proof valid)
[Any check fails]
Return false (proof invalid)
User
IndexedMerkleTree
predecessorIndex()
createNonMembershipProof()
verifyNonMembershipProof()
Hash Functions

CS-3372 Final Project Report

Group 1

The Attribute Proving Protocol

An Nguyen: Bulletproof-Based Protocol Off-Chain

Dat Tran: On-Chain Deployment of Bulletproof

Why We Choose Bulletproof for Attribute Proving ?

- Blockchain transactions lack confidentiality
- Range proofs needed to ensure amounts are valid
- Earlier solutions: large proof size or trusted setup [6]
- In this Experiment
  - Bulletproofs is used for efficient range proof,
  - Smaller/ compact proof size

Limitations of Earlier Range Proofs [6]

- Linear-size range proofs: proof grows with bit length of data
- Trusted setup requirements in SNARKs
- Practical inefficiency in Bitcoin confidential transactions or any on-chain deployment

Bulletproofs Introduction

- Non-interactive zero-knowledge proof system [6]
- No trusted setup, relies only on discrete log assumption
- Proof size logarithmic-scaling in range size: $2 \log_2(n) + 9$ elements

Range Proofs in Bulletproofs

- Prove committed (secret) value lies within a range - from [0, 2^n - 1]
- Compact proofs, suitable for confidential transactions
- Support batch verification for efficiency [6]

The Idea of doing bullet-range proof (Brief)

- Use Pedersen Commitment with randomization to prove a secret value v.
- Show that v satisfies a dot-product relation on its binary representation without revealing it.
- Reject if v is out of range or has no valid binary representation.

Aggregation & Applications

- Aggregate m range proofs with only O(log m) overhead (extra-work) [6]
- Able to do multiple attribute verification
- Applications:
    1. Confidential transactions
    2. Solvency proofs (Provisions)
    3. ***Smart Contracts Deployment (Our-case)***

Using MiMC to Make Bulletproof on-chain

- Bulletproof is not able to bring on-chain as Ethereum network yet
- Solution: Doing the Hash the proof & check it's result on-chain
- Challenge:
  - Secure ZKPs need efficient primitives for Circuit construction
  - Linear operation are cheap, multiplications as (bit-shift , XOR) are expensive [17]
  - Goal: reduce multiplicative complexity in the Hashing Process
  - Hashing Parameters set-up need to be simple (Poseidon setup is complicated)

MiMC Design

- The MiMC hashing based on the Sponge-Construction in [18]
  - To map one input bit-string chunks (each chunk size r) to a fix-length string (n)
  - The Hashing-Function (or Sponge) have an internal state (state-machine) able to update after each hashing iteration
  - The State is a Binary-String (can be manipulated)

MiMC Hashing Algorithm (Short Version)

- **The MiMC Hashing Include 3 keys steps**
  - **Initialization:** Set up the starting state with input and key.
  - **Absorption:** Mix the input into the state through nonlinear rounds.
  - **Compression:** Reduce the final state to produce the hash output.
- The Absorption & Compression depend on the Data-Mixing Process using Special Permutation Function (P) -> need to be engineer carefully for security

MiMC Permutation Function Design

- **Permutation Function P** is Essential for Security
- How to Build P:
  - Build the Rounding function F(x) = x^3 mod P (P is large)
  - x being argument with k (the key) & a constant c by XOR
  - each times doing the hashing, use a unique c (rounding constant) & accepting last iteration rounding input

MiMC The Permutation Function F [17]

$$F(x) = x^3 \pmod{p}$$
$$F_i(x) = F(x \oplus k \oplus c_i)$$
$$H(x) = (F_{r-1} \circ F_{r-2} \circ \cdots \circ F_0)(x) \oplus k$$

- The first & last round const c = 0
- Choosing c values from [1 to r-2] is matter for hashing security

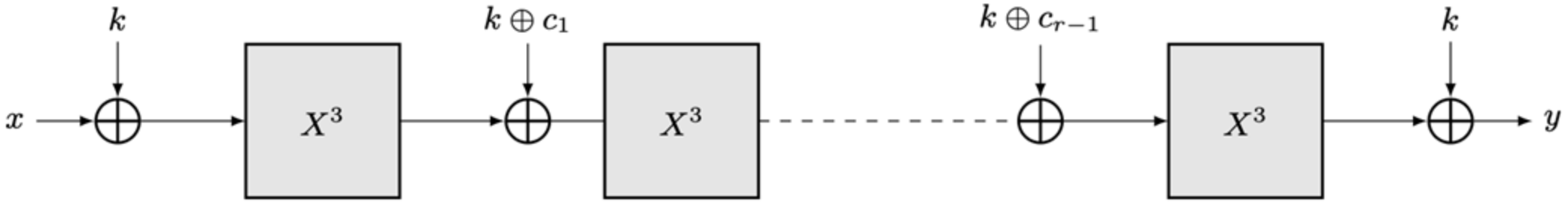


Fig. 1: r rounds of MiMC-n/n

MiMC as a Hash Function Application

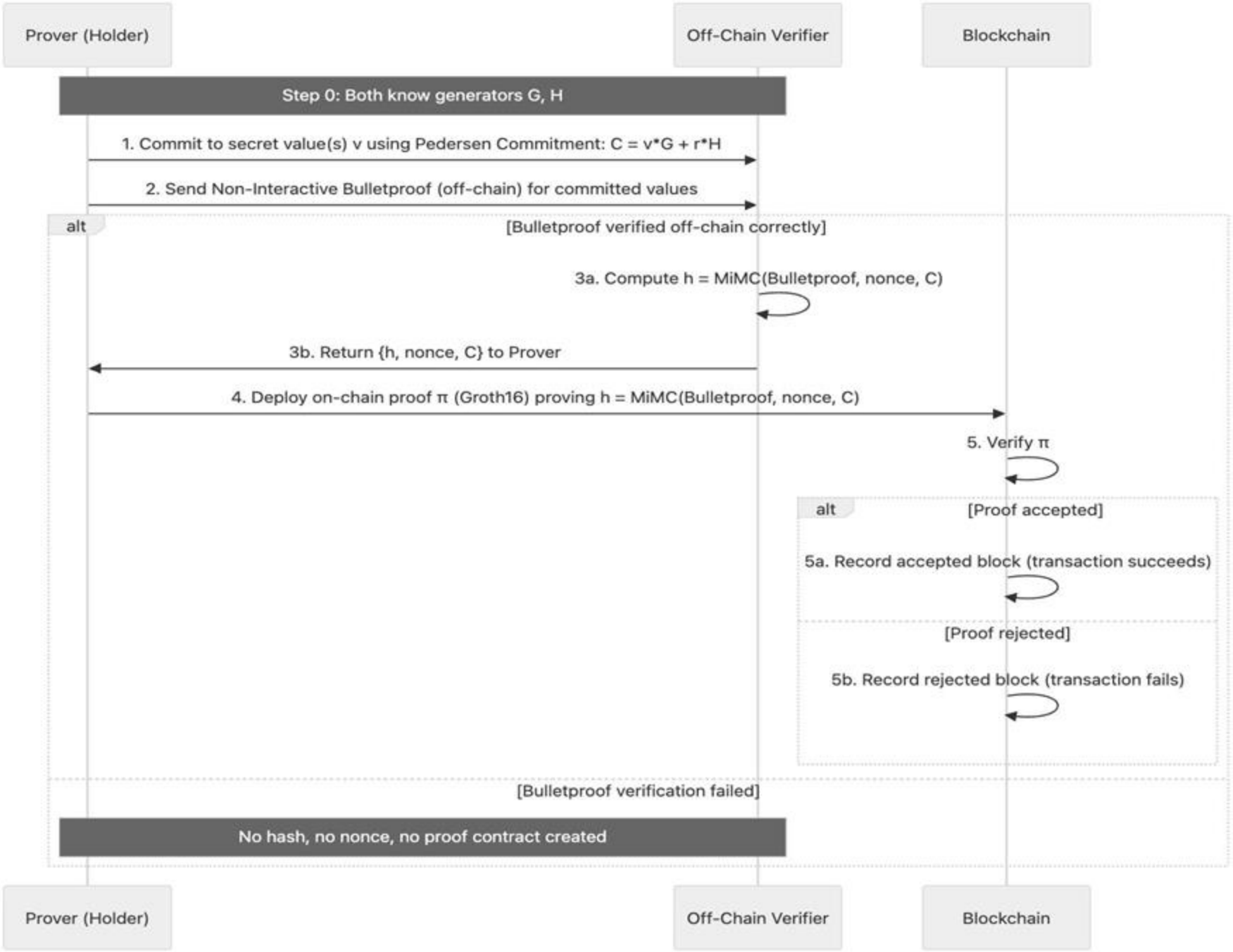
- Applications: zk-SNARKs, verifiable computing
- Outperforms SHA-256 in SNARK contexts [17]
  - Why: Only doing the addition on bits & less on multiplication

Integration with zk-SNARK

- MiMC provides efficient in-circuit hashing (for bring the bulletproof to on-chain verification)
- Bulletproofs ensure short range proofs with no trust and compact-size ~ log(#attribute or #asked predicate)
- Together: efficient attribute proving protocol

Attribute Proving Flow

- Commit to attributes with Pedersen commitments
- Off-Chain Validity Proof with Bulletproofs
- Off-Chain Hashing with MiMC
- Generate On-Chain Proof with zk-SNARK
- On-Chain Verification


Prover (Holder)
Off-Chain Verifier
Blockchain
Step 0: Both know generators G, H
1. Commit to secret value(s) v using Pedersen Commitment: C = v*G + r*H
2. Send Non-Interactive Bulletproof (off-chain) for committed values
alt
[Bulletproof verified off-chain correctly]
3a. Compute h = MiMC(Bulletproof, nonce, C)
3b. Return {h, nonce, C} to Prover
4. Deploy on-chain proof π (Groth16) proving h = MiMC(Bulletproof, nonce, C)
5. Verify π
alt
[Proof accepted]
5a. Record accepted block (transaction succeeds)
[Proof rejected]
5b. Record rejected block (transaction fails)
[Bulletproof verification failed]
No hash, no nonce, no proof contract created
Prover (Holder)
Off-Chain Verifier
Blockchain

The System Designing

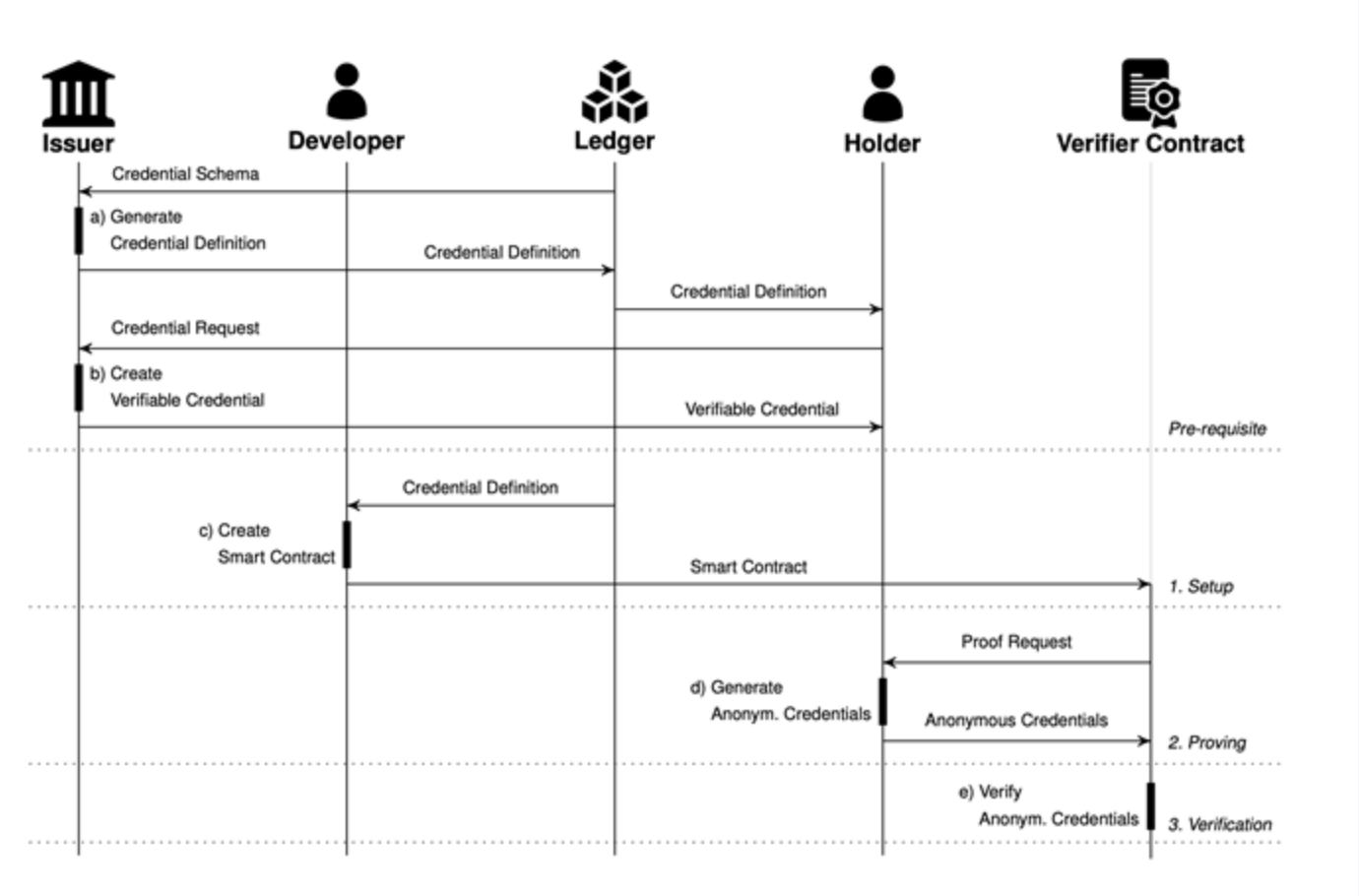
Khiem Nguyen: Off-Chain Designing

Dung Nguyen: On-Chain Designing

But Why Have to be on Blockchain ? [1]

- Decentralized System & Trustless environment between parties
  - Removing the trust (or Less Trust)
  - Reducing Collusion opportunity
  - Better Security
  - Prevent Forgery & Easy to Recheck later

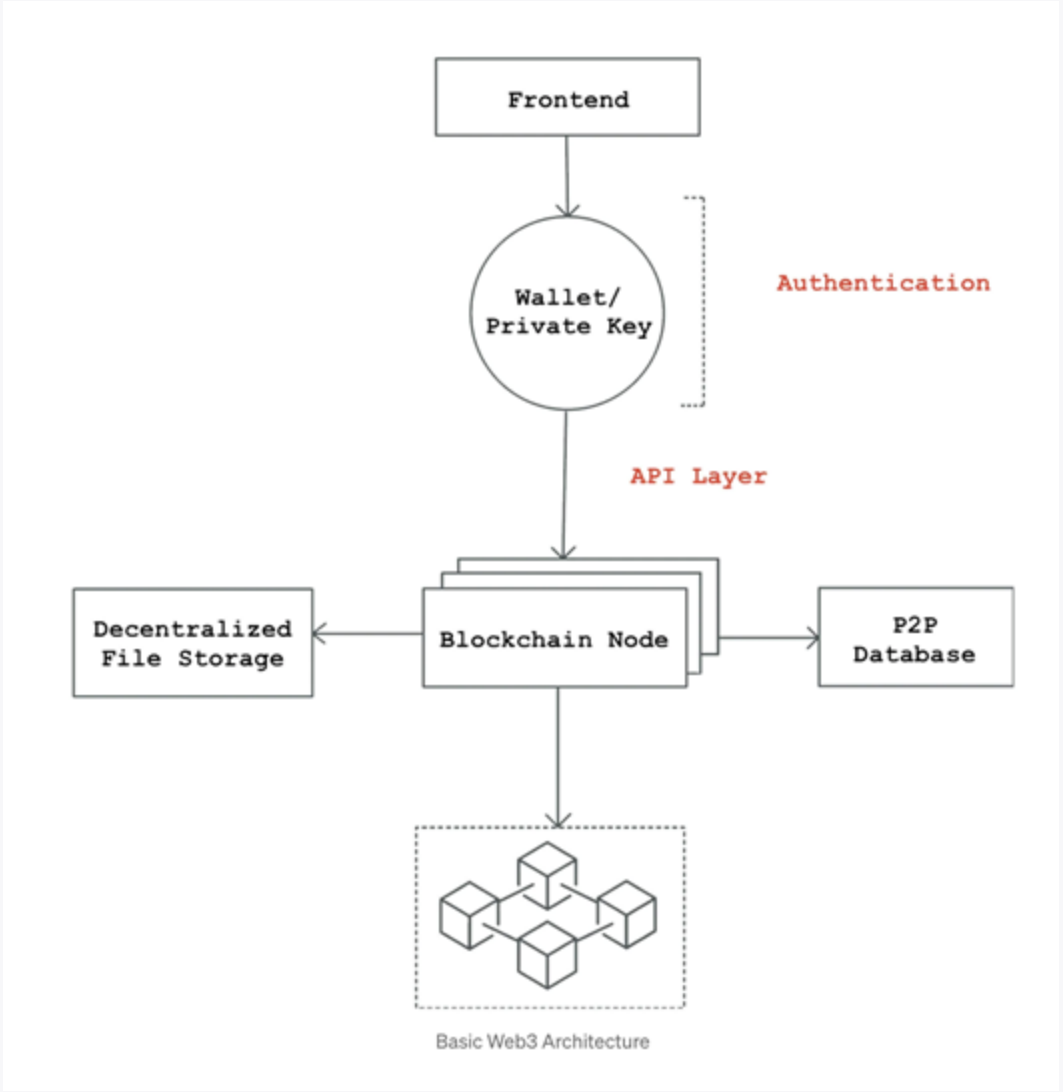
The Whole Pipeline – Muth et al., 2023



- The ledger is a public asset that all parties can access.
- Verifier (usually defined by the Developer) is the smart contract
- The issuer would predefine (update) the *credential definition* by pushing the new block

Between I & H & V is

a Web-3 System [3]



Hybrid Options (Off-Chain & On-Chain) [9]

- I create & issue the credential to H off-chain
- I published a certain number of claims that V (or D) can ask H
- V (or D) can create a smart contract to check H's proof validity
- Smart contract runs and deploys a block on the chain as a transaction

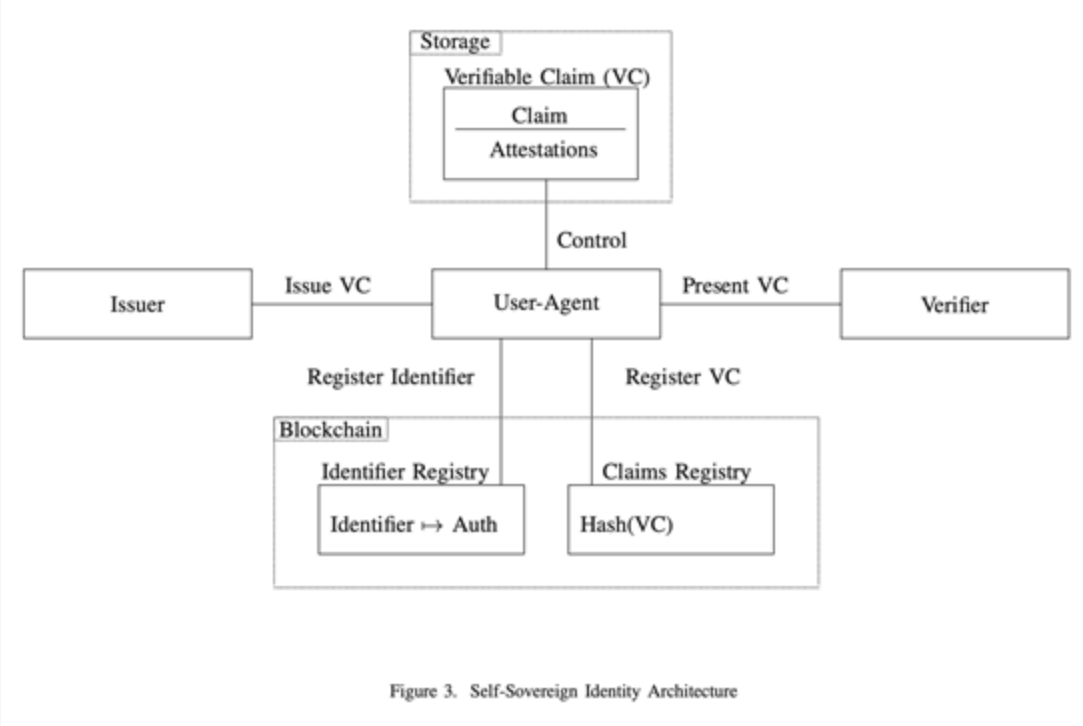
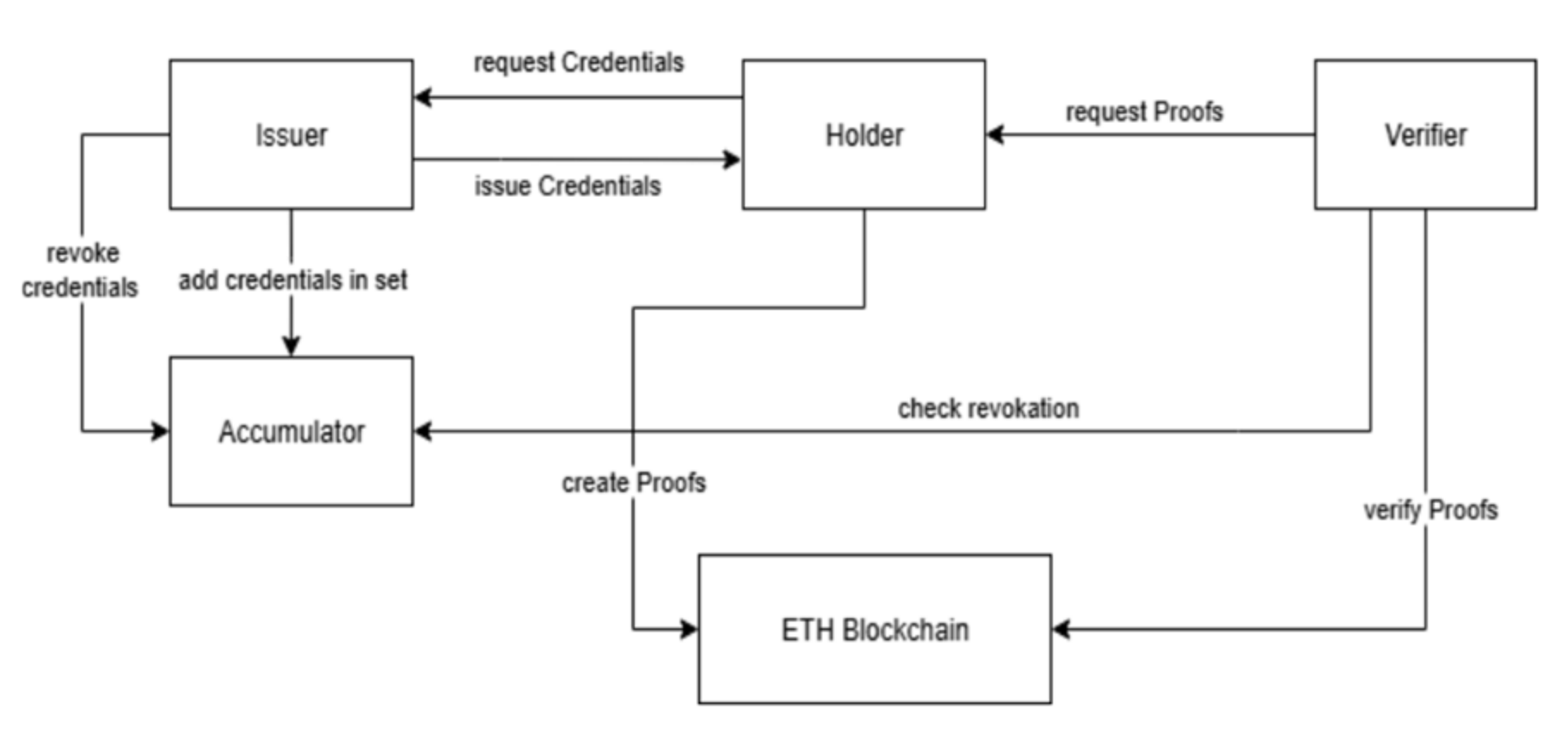


Figure 3. Self-Sovereign Identity Architecture

New pipeline with Accumulator



- Accumulator:

  this is our extra-module/system in additional to the old one(Muth) with additional of Acc for revocation check of credential

Mathematic foundation

$$\text{Credential: } \sigma = \text{Sign}_{sk_I}(H(\mathbf{attr} \| \text{nonce}))$$

$$\text{Commitment: } C = g^{\mathbf{attr}} h^r \text{ where } r \xleftarrow{\$} \mathbb{Z}_p$$

$$\text{Witness: } w = \text{acc}_{current}^{1/(C+\delta)}$$

$$\text{Proof: } \pi = \mathsf{ZKProve}(\text{stmt}, \text{wit})$$

$$\text{Verification: valid} = \mathsf{ZKVerify}(vk, \pi, \mathbf{x}) \wedge \text{AccVerify}(\text{acc}, w, C)$$

The Empirical Study

Khiem Nguyen, Dung Nguyen: Experiment Design

Developing Environment

Techstack

- **Geth (Ethereum blockchain simulation)**
- **Solidity**
- **Rust**
- **Circom (ZK_Snarks)**
- **Nodejs 18 (API)**

Algorithm

- Signature (BLS_12_381)
- ZKP (ZK_Snarks)

IELTS Certification Verification

IELTS™

Test Report Form

GENERAL TRAINING

NOTE Admissions to undergraduate and post graduate courses should be based on the ACADEMIC Reading and Writing Modules. GENERAL TRAINING Reading and Writing Modules are **not** designed to test the full range of language skills required for academic purposes. It is recommended that the candidate's language ability as indicated in this Test Report Form be re-assessed **after two years** from the date of the test.

Centre Number: GB555 | Test Date: 04/JUL/2024 | Candidate Number: 000203

Candidate Details

Family Name: ABCDEFGHIJKLMNOP

First Name: ABCDEFGHIJKLMNOP

Candidate ID: 123456789

Date of Birth: 12/12/1959 | Sex: M | Scheme Code: Private Candidate

Country or Region of Origin:

Country of Nationality: ABCDEFGHIJKLMNOP

First Language: ABCDEFGHIJKLMNOP

Test Results

Listening	Reading	Writing	Speaking	Overall Band Score
9.0 C2	9.0 C2	9.0 C2	9.0 C2	9.0

Administrator Comments

Centre Stamp

Validation Stamp

Administrator's Signature

Issue Date: 22/08/2024

(UKVI CEFR Threshold Met)

B1	B2	C1	C2
✓	✓	✓	✓

UKVI Unique Reference Number: IELI04072024/GB555/000203

Test Report Form Number: 24NZ000203ABCA013G

The validity of this IELTS Test Report Form can be verified online by recognising organisations at ielts.org/verify

Credential:

```
==========================
name: Alice Johnson
listening: 8.0
reading: 7.5
writing: 7.0
speaking: 8.5
==========================
```

IELTS Certification Verification

Signature:

```
==========
secret: abc123
salt: def456
amount: 1000
==========
```

Credential:

```
==========
name: Alice Johnson
listening: 8.0
reading: 7.5
writing: 7.0
speaking: 8.5
==========
```

IELTS Certification Verification

Signature:

```
========================
commitment: 0xd95d…
nullifier: 0x6ca13…
========================
```

Credential:

```
========================
name: Alice Johnson
listening: 8.0
reading: 7.5
writing: 7.0
speaking: 8.5
========================
```

IELTS Certification Verification

Zero-Knowledge Proof:

```
========================
a: [107…,339…, 1]
b: [151…,205…,157…]
c: [123…,322…,233…]
========================
```

IELTS Certification Verification

Zero-Knowledge Proof:

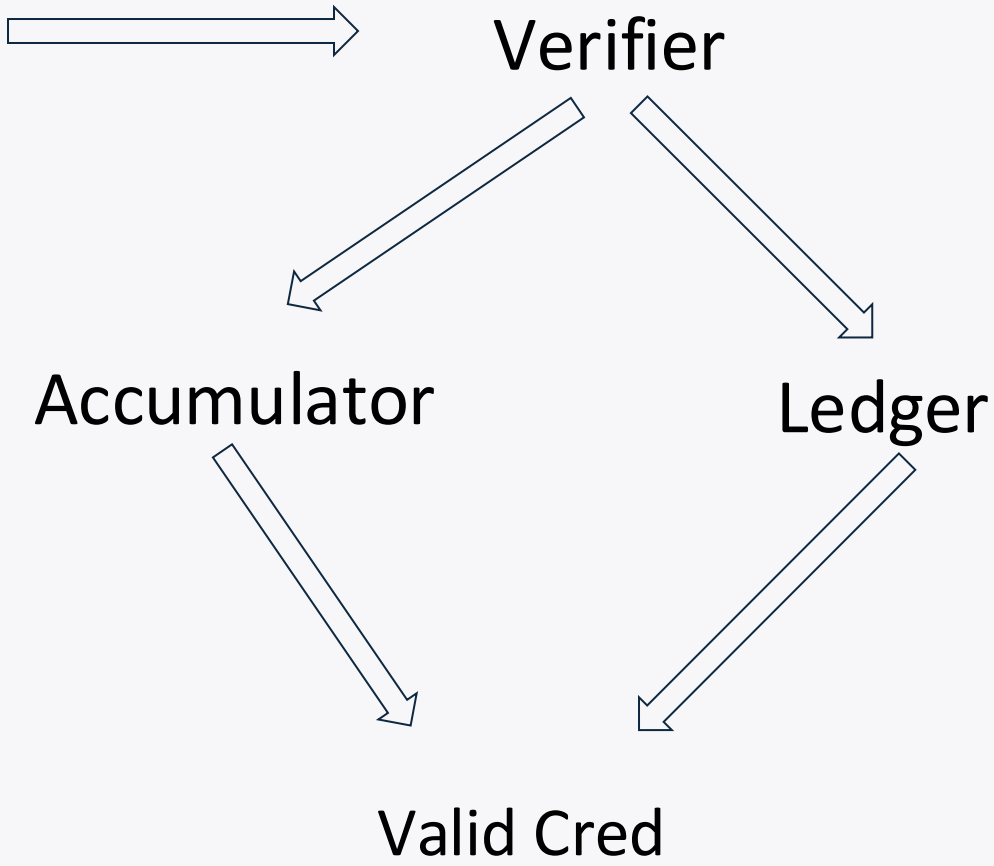
Verifier

```
========================
a: [107…,339…, 1]
b: [151…,205…,157…]
c: [123…,322…,233…]
========================
```

IELTS Certification Verification

Zero-Knowledge Proof:

```
==========================
a: [107…,339…, 1]
b: [151…,205…,157…]
c: [123…,322…,233…]
==========================
```



Cost Evaluation

Table 4: Comprehensive Performance Comparison of Revocation Mechanisms

Metric	EC Accumulator	Merkle Tree	Sorted List
Gas Consumption (Ethereum)			
Verification ($n = 10^3$)	190,000	45,000	38,000
Verification ($n = 10^4$)	205,000	62,000	51,000
Verification ($n = 10^5$)	215,000	78,000	67,000
Single Revocation	420,000	85,000	42,000
Batch Revocation (64)	415,000	15,000	8,500
Zero-Knowledge Proof Characteristics			
Circuit Constraints	12,000	8,000-15,000	5,000
Proof Size (bytes)	128,000	180,000	95,000
Prover Time (seconds)	2.7	6.2	1.8
Scalability Properties			
Witness Size	$O(1)$	$O(\log n)$	$O(n)$
Update Complexity	$O(n)$	$O(k \log n)$	$O(n)$
Verification Time	$O(1)$	$O(\log n)$	$O(n)$
Implementation Characteristics			
Cryptographic Assumptions	q-SDH	Hash security	Hash security
Implementation Complexity	High	Medium	Low
Transparency	None	Full	Full

Conclusion

Research Contributions

- The development of the comprehensive system model specifically designed for anonymous credentials in gas-metered execution environments.

Research Contributions

- Providing:
  - a comparative analysis of revocation mechanisms,
  - showing trade-offs between elliptic curve accumulators (efficient verification but costly updates)
  - Merkle trees (scalable updates but less efficient verification), o
  - ffering quantitative guidance for system design.

Research Limitations and Future Work

- Cryptographic Assumptions:
  - Reliance on **the q-Strong Diffie–Hellman assumption** for elliptic-curve accumulators introduces a potential single point of failure.
  - **Trusted setup requirements** in some zk-SNARKs pose similar risks.
  - Future work should:
    - Investigate **more conservative cryptographic assumptions**.
    - Explore **transparent setup alternatives** to avoid trust bottlenecks.

Research Limitations and Future Work

- Scalability issues with EC accumulators, which fail at very large set sizes.
- Merkle trees have call-data scaling problems.
- A hybrid approach or advanced cryptographic methods may solve this issue.

Research Limitations and Future Work

- User experience challenges:
  - proof generation is too slow (2.7–6.2s),
  - witness management is complex for non-technical users.
  - Future research should focus on:
    - hardware acceleration, improved circuit architectures, and user interface designs.
- Economic sustainability depends on evolving blockchain fee markets and layer-2 scaling solutions.

Reference

Materials that you can use to fact-check us !

Reference List

[0]: Slamanig, D. (2025). *Privacy-Preserving Authentication: Theory vs. Practice*. ArXiv.org. https://arxiv.org/abs/2501.07209v1

[1]: Muth, R., Galal, T., Heiss, J., & Tschorsch, F. (2022). *Towards Smart Contract-based Verification of Anonymous Credentials*. Cryptology EPrint Archive. https://eprint.iacr.org/2022/492

Reference List

[2]: FinCEN - Financial Crimes Enforcement Network. (2019). *Application of FinCEN's Regulations to Certain Business Models Involving Convertible Virtual Currencies* (p. 18). https://www.fincen.gov/sites/default/files/2019-05/FinCEN%20CVC%20Guidance%20FINAL.pdf

[3]: Agrawal, Y. (2022, May 30). *A Beginners' guide the basic Web3 Architecture and Tech Stack*. Web3Auth. https://medium.com/toruslabs/a-beginners-guide-the-basic-web3-architecture-and-tech-stack-81f2061d263c

[4]: Flamini, A., Silvio Ranise, Sciarretta, G., Scuro, M., Smaniotto, N., & Tomasi, A. (2025). *Public Key Accumulators for Revocation of Non-Anonymous Credentials*. Cryptology EPrint Archive. https://eprint.iacr.org/2025/549

Reference List

[5]:Laurie, B., & Kasper, E. (n.d.). *Revocation Transparency* (pp. 1–3). Retrieved August 16, 2025, from https://www.links.org/files/RevocationTransparency.pdf

[6]: Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Pieter Wuille, & Maxwell, G. (2017). *Bulletproofs: Short Proofs for Confidential Transactions and More*. Cryptology EPrint Archive. https://eprint.iacr.org/2017/1066

[7]: kd14. (2025, February 18). *Sparse Merkle Tree vs Indexed Merkle Tree*. Vac. https://forum.vac.dev/t/sparse-merkle-tree-vs-indexed-merkle-tree/438

[8]: Camenisch, J., Kohlweiss, M., & Soriente, C. (2008). *An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials*. Cryptology EPrint Archive (Eprint.iacr.org). https://eprint.iacr.org/2008/539

Reference List

[9]: Mühle, A., Grüner, A., Gayvoronskaya, T., & Meinel, C. (2018). A survey on essential components of a self-sovereign identity. *Computer Science Review*, *30*, 80–86. https://doi.org/10.1016/j.cosrev.2018.10.002

[10]: Stinson, D. R. (2019). *Cryptography: Theory and Practice*. (Original work published 1995)

[11]: Thaler, J. (2022). *Proofs, Arguments, and Zero-Knowledge 1*. https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf

[12]: Investopedia. (2025, February 9). *Smart Contracts*. Investopedia. https://www.investopedia.com/terms/s/smart-contracts.asp

Reference List

[13]:Tessaro, S., & Zhu, C. (2023). Revisiting BBS signatures. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*.

https://eprint.iacr.org/2023/275

[14]: Nguyen, L. (2005, April 28). *Accumulators from Bilinear Pairings and Applications to ID-based Ring Signatures and Group Membership Revocation*. Cryptology EPrint Archive.

https://eprint.iacr.org/2005/123

Reference List

[15]: Groth, J. (2016). *On the Size of Pairing-based Non-interactive Arguments*. EPrint IACR. https://eprint.iacr.org/2016/260

[16]: Grassi, L., Dmitry Khovratovich, Rechberger, C., Roy, A., & Schofnegger, M. (2019). *Poseidon: A New Hash Function for Zero-Knowledge Proof Systems*. Cryptology EPrint Archive. https://eprint.iacr.org/2019/458

Reference List

[17]: Albrecht, M., Grassi, L., Rechberger, C., Roy, A., & Tiessen, T. (2016). *MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity*. Cryptology EPrint Archive. https://eprint.iacr.org/2016/492

[18]: Bertoni, G., Daemen, J., Peeters, M., & Assche, G. (2008). *On the Indifferentiability of the Sponge Construction*. https://www.iacr.org/archive/eurocrypt2008/49650180/49650180.pdf

[19]:Lefevre, C. (2023). *Indifferentiability of the Sponge Construction with a Restricted Number of Message Blocks*. Cryptology EPrint Archive. https://eprint.iacr.org/2023/217

Reference List

[20]: Haider, F. (2018). Compact Sparse Merkle Trees.

[21]: Indexed Merkle Tree (Nullifier Tree) | Privacy-first zkRollup | Aztec Documentation. (2025).

Retrieved August 17, 2025, from Aztec.network website:

https://docs.aztec.network/aztec/concepts/advanced/storage/indexed_merkle_tree

[22]: Merkle Tree | Pangea. (n.d.). Retrieved from pangea.cloud website:

https://pangea.cloud/docs/audit/merkle-trees

Reference List

[23]: Privacy Stewards of Ethereum. (2023, August 17). Indexed Merkle Trees - Lasse/Aztec Labs. Retrieved August 17, 2025, from YouTube website: https://www.youtube.com/watch?v=x_0ZhUKtWSs

[24]: Nomos Team. (2025, March 17). Designing Nullifier Sets for Nomos Zones: Sparse vs Indexed Merkle Trees. Retrieved August 17, 2025, from Nomos Blog website: https://blog.nomos.tech/designing-nullifier-sets-for-nomos-zones-sparse-vs-indexed-merkle-trees/

Reference List

[25]: Notes (UTXOs) | Privacy-first zkRollup | Aztec Documentation. (2025). Retrieved August 17, 2025, from Aztec.network website: https://docs.aztec.network/aztec/concepts/storage/notes

[26]: Notes, Nullifiers, and Trees. (2025). Retrieved from Penumbra.zone website: https://protocol.penumbra.zone/main/concepts/notes_nullifiers_trees.html