Set Accumulators for Blockchains

Matteo Loporchio
21/5/2026
matteo.loporchio@di.unipi.it

UNIVERSITÀ DI PISA

Set accumulators

Cryptographic primitives that:

1) represent a large set of elements S with a single and constant-size value (*accumulator value*);

2) can be used to efficiently generate a proof of membership (*witness*) for an element in S.

Set accumulators

S = Set of elements

Set accumulator value A(S)

Check proof

Is  in S?

Yes + proof

Prover

Verifier

Set accumulators

The **prover** knows the entire content of the set (i.e., it stores the values explicitly).

The **verifier** knows only the accumulator value (sufficient to verify proofs).

Set accumulators

Classification

- **Static**: do not support element insertion and deletion.
- **Dynamic**: also support element insertion and deletion.

- **Asymmetric**: requires proof generation before verification.
- **Symmetric**: no proof needed, the accumulated value on its own is sufficient for verification.

- **Universal** accumulators also support **non-membership** proofs (i.e., prove that an element is not in the accumulated set).

Set accumulators

Batch membership proofs: demonstrate to a verifier that all elements of a set S are included in X (i.e., S is a subset of X).

Batch non-membership proofs: demonstrate to a verifier that all elements of a set S are not included in X (i.e., S is not a subset of X).

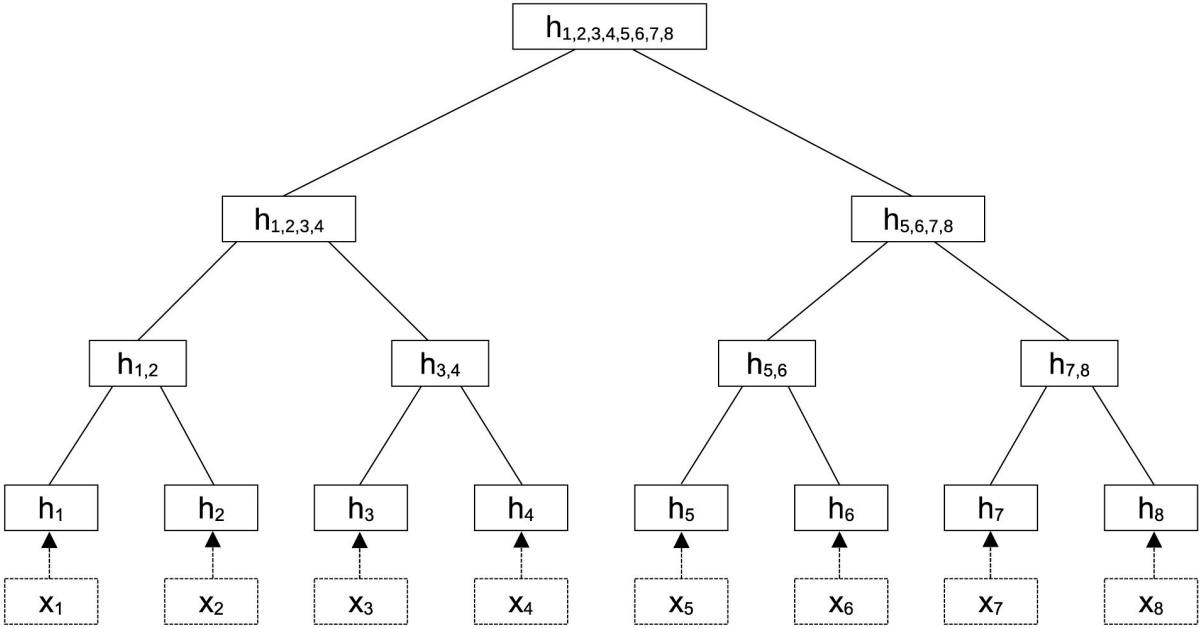
Set accumulators

- Merkle Tree
- RSA accumulator [BP97]
- Bilinear accumulator [Ngu05]
- Expressive accumulator [ZKP17]

Sources:
[BP97] N. Barić and B. Pfitzmann, “Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees,” Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 480–494, 1997.
[Ngu05] L. Nguyen, “Accumulators from Bilinear Pairings and Applications,” Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 275–292, 2005.
[ZKP17] Y. Zhang, J. Katz, and C. Papamanthou, “An Expressive (Zero-Knowledge) Set Accumulator,” 2017 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, pp. 158–173, Apr. 2017.

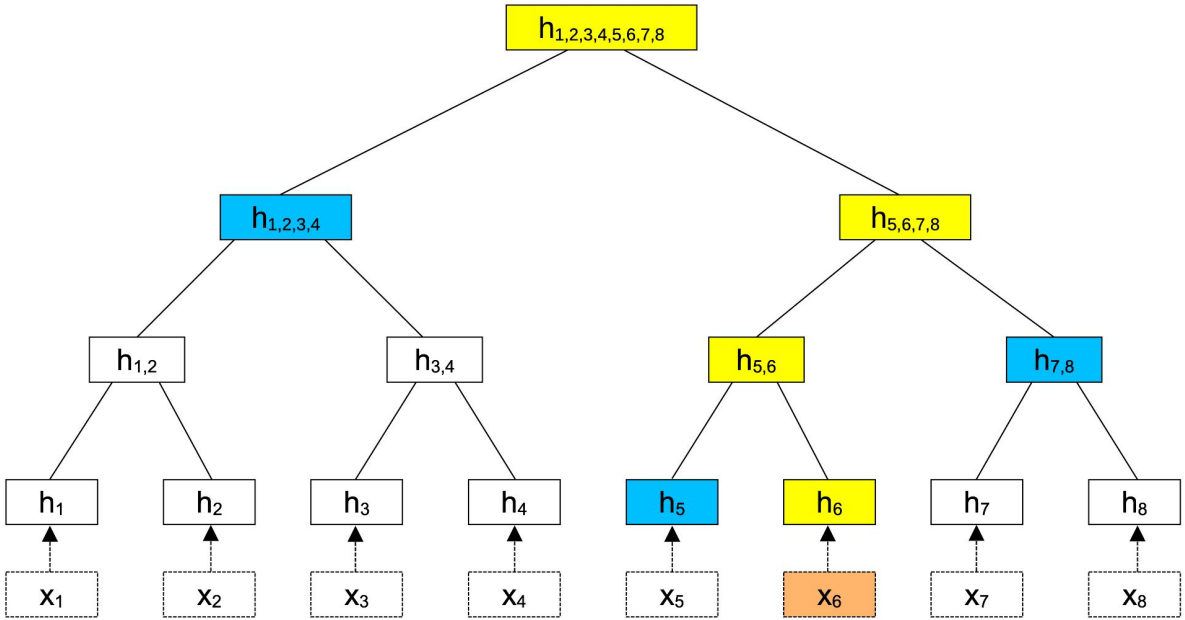
Merkle tree

Verify the integrity of a set of elements (e.g., TXs in a Bitcoin block).



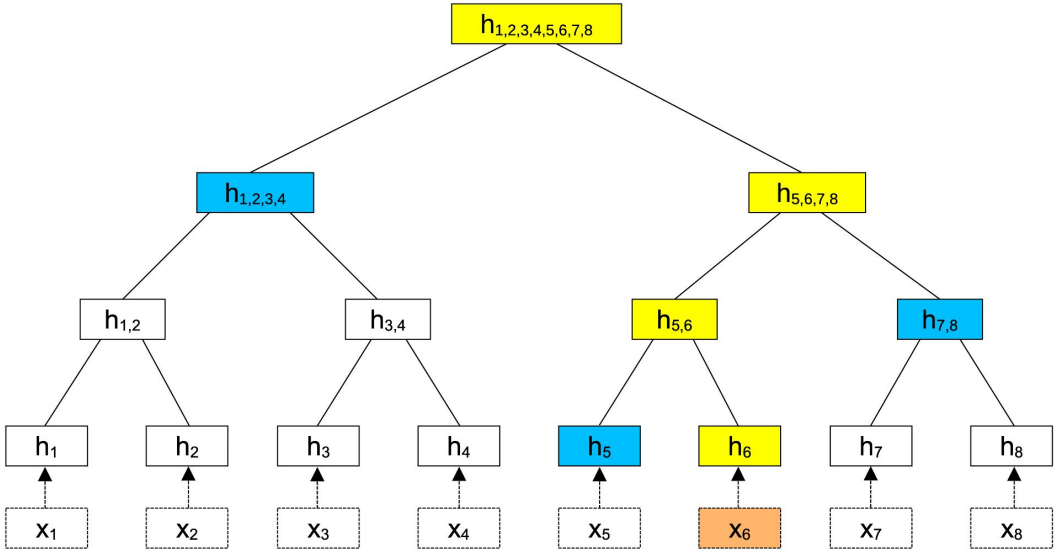
Merkle tree

Membership proof: an element x is included in a set S.



Proof

Merkle tree



1) Root hash represents the accumulator value.

2) Proof of membership is represented by nodes in the co-path from leaf to root.

Merkle tree

Security relies on cryptographic hash functions (collision-resistance property).

1) To modify a tree leaf while keeping the root unchanged, an adversary would need to find a collision for the hash function used.

2) Adversaries cannot forge “fake” proofs (convincing a verifier that an element is in the set while in fact it is not).

RSA accumulator

Accumulates a set of **prime** numbers $X = \{x_1, \ldots, x_n\}$.

Let $N = p \cdot q$ be an RSA **modulus**, with p and q (large) prime numbers.

RSA accumulator

Accumulates a set of **prime** numbers X = {x_1, ..., x_n}.

Let N = p · q be an RSA **modulus**, with p and q (large) prime numbers.

The **accumulator value** of X is

$$A(X) = g^{(x_1 \cdot \ldots \cdot x_n)} \bmod N$$

where g is an exponentiation **base** in $\mathbb{Z}_N = \{0, 1, \ldots, N\}$.

=> N, g and A(X) are **public**. p and q should be kept **secret**.

RSA accumulator

If data are not prime numbers we can use a prime-generating hash function (“hash-to-prime”).

- Example: SHA-256 hash function + Miller-Rabin primality test.

Security: RSA modulus size determines accumulator security.

- Example: a 3072-bit RSA modulus gives ~ 128 bits of security.

A simple hash-to-prime function with 256-bit output

```python
from Crypto.Hash import SHA256
from Crypto.Util.number import isPrime

def hash_to_prime(data: bytes) -> tuple[int, int]:
    nonce = 0
    while True:
        h = SHA256.new()
        h.update(data)
        h.update(nonce.to_bytes(8, "big"))
        x = int.from_bytes(h.digest(), "big")
        x |= (1 << 255)  # Ensures full 256-bit size
        x |= 1 # Forces result oddness
        if isPrime(x):
            return x, nonce
        nonce += 1

if __name__ == '__main__':
    message = "Hello, world!"
    prime, counter = hash_to_prime(message.encode("utf-8"))
    print(prime, counter)

```

RSA accumulator

Example:

We fix two primes p = 7, q = 11 and compute the RSA modulus N = p · q = 77.

Let X = {13, 31, 37, 59} and g = 2 be an exponentiation base in $\mathbb{Z}_{77}$.

Accumulator value: A(X) = 2^(13 · 31 · 37 ·59) = 39 mod 77

RSA accumulator

X = {13, 31, 37, 59}. A(X) = 39. g = 2.

Membership: A prover P wants to demonstrate to a verifier V that 37 ∈ X.

1) P computes the product of all elements in X - {37}, namely t = 13 · 31 ·59 = 23777.

2) P computes a witness w = g^t = 2^(23777) = 18 mod 77 and sends w to V.

3) V computes w^37 = 18^37 = 39 mod 77. Since A(X) = 39 mod 77 the proof is accepted as valid.

RSA accumulator

Non-membership protocol is made possible by Bezout's lemma:

"For any two integers x and y, the gcd(x, y) can be expressed as a linear combination of x and y".

Bezout cofactors (i.e., a and b s.t. ax + by = gcd(x, y)) can be computed with the Extended Euclidean Algorithm.

RSA accumulator

$X = \{13, 31, 37, 59\}$. $A(X) = 39$. $g=2$.

Non-membership: P wants to convince V that $61 \notin X$.

1) P computes $f = 13 \cdot 31 \cdot 37 \cdot 59 = 879749$.

2) P finds a and b such that $a \cdot f + b \cdot 61 = \gcd(f, 61) = \gcd(879749, 61) = 1$.

3) P sends the witness $w' = (a, b) = (-26, 374975)$ to V.

4) V verifies w' by checking if $A(X)^a \cdot (g^{(61)})^b = g \bmod 77$.
Indeed it is $39^{(-26)} \cdot 2^{(61 \cdot 374975)} = 53 \cdot 32 = 2 \bmod 77$.

RSA accumulator

Advantages:

- Supports batch proofs.
- Proof size is O(1) and independent from set size.

Limitations:

- **Trusted setup**: prover and verifiers have to agree on the RSA modulus N (but without disclosing p and q).
- Modular exponentiations are more expensive than hashing (bit operations).

RSA accumulator

Definition 3.3 (*Strong RSA Assumption*). When the RSA modulus N is sufficiently large and randomly generated, given a value $x \in \mathbb{Z}_N^*$ it is computationally infeasible to find $a, b \in \mathbb{Z}$, with $a \neq \pm 1$, such that $x \equiv b^a \mod N$.

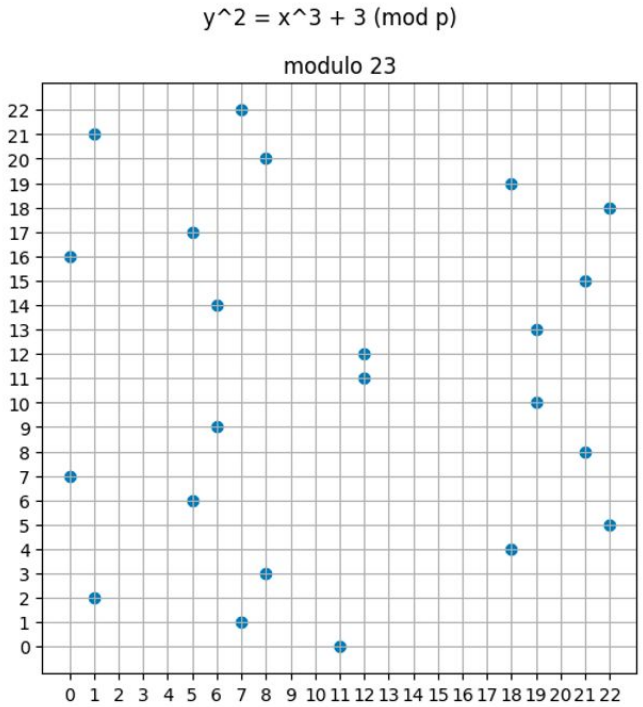
The RSA accumulator is **collision-free** if it is computationally infeasible for an adversary to forge a valid membership (resp. non-membership) proof for an element that is not part (resp. is part) of the accumulated set.

True as long as p and q (factors of N) are kept **secret**.

Bilinear accumulator

Based on efficient **bilinear pairings**, i.e., functions constructed on the points of elliptic curves defined over a finite field $\mathbb{F}_q$ (e.g., Weil and Tate pairings).

Supports both membership and non-membership proofs.



Source: [Ngu05] L. Nguyen. “Accumulators from Bilinear Pairings and Applications”. In: Topics in Cryptology - CT-RSA 2005, The Cryptographers’ Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings. Vol. 3376. Lecture Notes in Computer Science. Springer, 2005, pp. 275–292.

Bilinear accumulator

Definition 3.7 (*q-SBDH Assumption*). Let p be a prime number and let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a symmetric bilinear pairing, where $\mathbb{G}$ and $\mathbb{G}_T$ are two cyclic groups. Let also g be a generator of $\mathbb{G}$ and s a random secret chosen in $\mathbb{Z}_p$. The q-strong bilinear Diffie–Hellman (q-SBDH) assumption holds if and only if for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ outputs a pair $(e(g, g)^{1/(s+x)}, x)$ with $x \in \mathbb{Z}_p^*$ given a tuple $(g, g^s, g^{s^2}, \ldots, g^{s^q})$ as input is negligible.[2]

Trusted setup phase required.

- Trusted party generates secret value s (called trapdoor) and forgets it.
- Knowledge of s would allow a party to forge fake proofs.
- Public parameters: {g^(s^i) | 0 ≤ i ≤ q} with q fixed in advance.

Accumulated sets can have a maximum cardinality equal to q.

Bilinear accumulator

A Java implementation is available in the Cryptimeleon library (https://cryptimeleon.github.io).

Usage example:

https://github.com/mloporchio/Tutorial-ICBC2025/blob/main/bilinear/BilinearAccumulatorExample.java

Source: Jan Bobolz et al. "Cryptimeleon: A Library for Fast Prototyping of Privacy-Preserving Cryptographic Schemes," in IACR Cryptol. ePrint Arch., vol. 2021, pp. 961, 2021.

Expressive accumulator

Still based on bilinear pairings.

Supports **verifiable operations** on accumulated sets.

Examples: Given a set X = {x1, … , xN}:

- SUM: prover can demonstrate that s = x1 + … + xN is actually the sum of the elements.

- MIN/MAX: prover can show that m (resp. M) is min(X) (resp. max(X)).

Source: [ZKP17] Y. Zhang, J. Katz, and C. Papamanthou. “An Expressive (Zero-Knowledge) Set Accumulator”. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017. IEEE, 2017, pp. 158–173.

Set accumulator comparison

Table 1
A comparison of different set accumulators for a set of n elements. For each construction, we detail: whether it needs a trusted setup phase and support batch proofs; the size of the public parameters (PK); the type of primitive operation it is built upon; the size of membership and non-membership proofs and the asymptotic complexity of generating and verifying them.

	Setup	Batch	PK	Operation	Membership			Non-Membership		
					Size	Gen.	Ver.	Size	Gen.	Ver.
RSA	Yes	Yes	$O(1)$	Modular Exp.	$O(1)$	$O(n)$	$O(1)$	$O(1)$	Ext. Euclid	$O(1)$
Bilinear	Yes	Yes	$O(q)$	EC mult.	$O(1)$	$O(n \log n)$	$O(1)$	$O(1)$	Ext. Euclid	$O(1)$
ESA	Yes	Yes	$O(\lvert\mathcal{U}\rvert^2)$	EC mult.	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(1)$
Merkle	No	No	$O(1)$	Hashing	$O(\log n)$	$O(\log n)$	$O(\log n)$	n/a	n/a	n/a

Source: [LBDR23] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa and L. Ricci. "A survey of set accumulators for blockchain systems." Computer Science Review 49 (2023): 100570.

Set accumulator comparison

Table 1
A comparison of different set accumulators for a set of n elements. For each construction, we detail: whether it needs a trusted setup phase and support batch proofs; the size of the public parameters (PK); the type of primitive operation it is built upon; the size of membership and non-membership proofs and the asymptotic complexity of generating and verifying them.

	Setup	Batch	PK	Operation	Membership			Non-Membership		
					Size	Gen.	Ver.	Size	Gen.	Ver.
RSA	Yes	Yes	$O(1)$	Modular Exp.	$O(1)$	$O(n)$	$O(1)$	$O(1)$	Ext. Euclid	$O(1)$
Bilinear	Yes	Yes	$O(q)$	EC mult.	$O(1)$	$O(n \log n)$	$O(1)$	$O(1)$	Ext. Euclid	$O(1)$
ESA	Yes	Yes	$O(\lvert\mathcal{U}\rvert^2)$	EC mult.	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(1)$
Merkle	No	No	$O(1)$	Hashing	$O(\log n)$	$O(\log n)$	$O(\log n)$	n/a	n/a	n/a

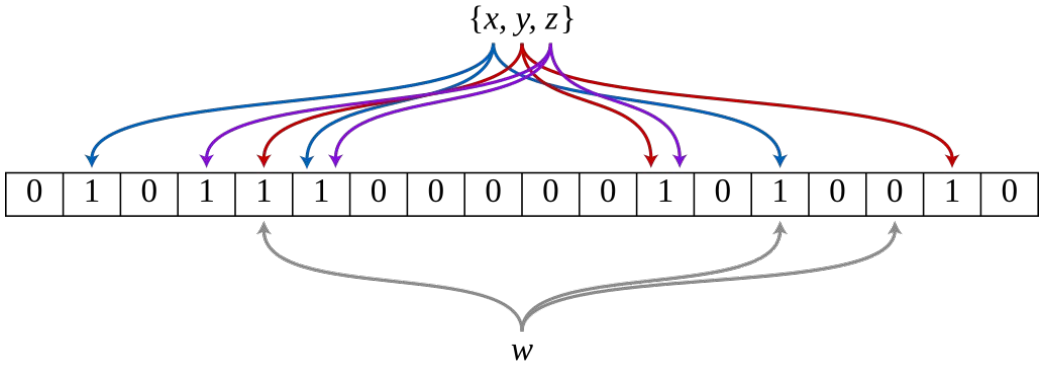
=> Hashing is less expensive than modular exponentiations and EC multiplications. No trusted setup, but proof size is O(log n).

Source: [LBDR23] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa and L. Ricci. "A survey of set accumulators for blockchain systems." Computer Science Review 49 (2023): 100570.

Secure Bloom Filters

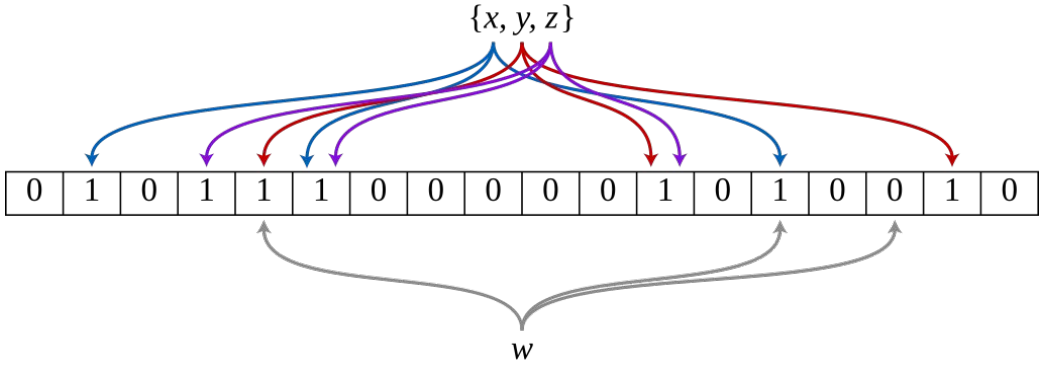
Employ cryptographic hash functions to insert and check the membership of elements.

- k cryptographic hash functions {H[0], ..., H[k-1]}
- m bits
- n elements to be inserted



Secure Bloom Filters

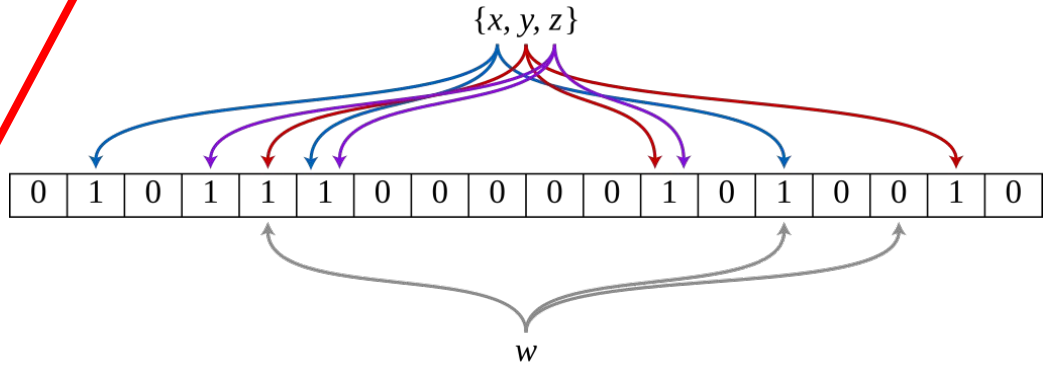
```
1  def initialize(B):
2      for i in range(0, m):
3          B[i] = 0
4
5  def insert(B, x):
6      for i in range(0, k):
7          h = H[i]
8          B[h(x)] = 1
9
10 def check(B, x):
11     for i in range(0, k):
12         h = H[i]
13         if B[h(x)] == 0:
14             return False
15     return True
16
```



Secure Bloom Filters

```
1  def initialize(B):
2      for i in range(0, m):
3          B[i] = 0
4
5  def insert(B, x):
6      for i in range(0, k):
7          h = H[i]
8          B[h(x)] = 1
9
10 def check(B, x):
11     for i in range(0, k):
12         h = H[i]
13         if B[h(x)] == 0:
14             return False
15     return True
16
```

True = x **might** be in the set
False = x **is surely not** in the set



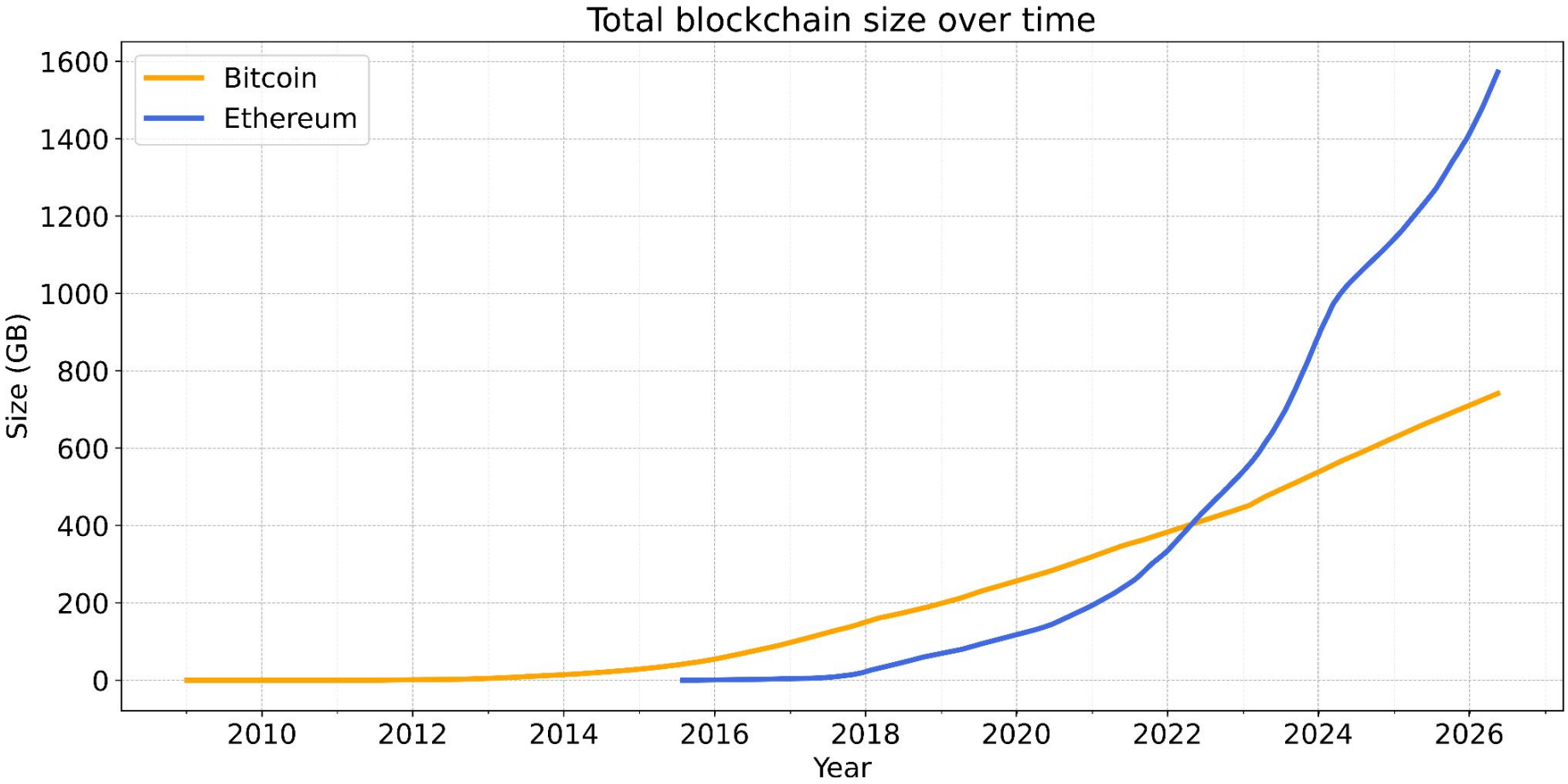
Secure Bloom Filters

Symmetric accumulators (i.e., no need to compute a witness to check membership).

Optimal size also depends on the number of elements in the set
=> Bloom filters are less space-efficient w.r.t. accumulators.

```
n = ceil(m / (-k / log(1 - exp(log(p) / k))))
p = pow(1 - exp(-k / (m / n)), k)
m = ceil((n * log(p)) / log(1 / pow(2, log(2))));
k = round((m / n) * log(2));
```

Query authentication



Sources: https://blockchair.com/bitcoin/charts/blockchain-size and https://blockchair.com/ethereum/charts/blockchain-size

Query authentication

To retrieve on-chain data, resource-constrained light nodes need to query untrusted full nodes.

However, full nodes may tamper with the results.

Challenges:

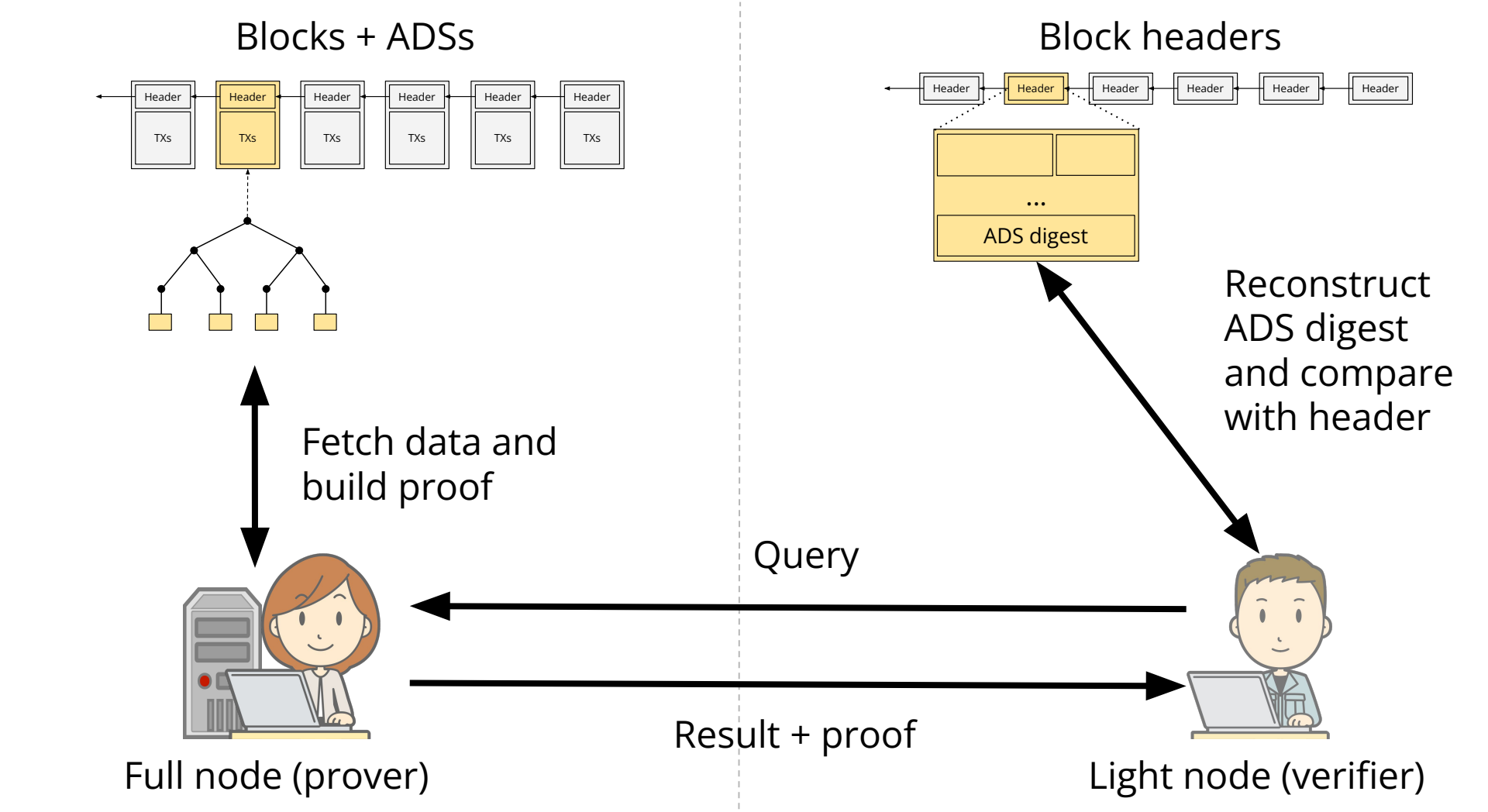
1) How to guarantee correctness of the results?
2) How to retrieve such data efficiently?

Query authentication

An **Authenticated Data Structure (ADS)** on a collection of data objects C incorporates cryptographic information so that:

1) an untrusted **prover** can carry out operations on C;

2) a **verifier** can efficiently check the **correctness** of the result of such operations.


Blocks + ADSs
Header
TXs
Block headers
...
ADS digest
Reconstruct ADS digest and compare with header
Fetch data and build proof
Query
Result + proof
Full node (prover)
Light node (verifier)

Query authentication

The cryptographic proof of correctness is also called **Verification Object (VO)**.

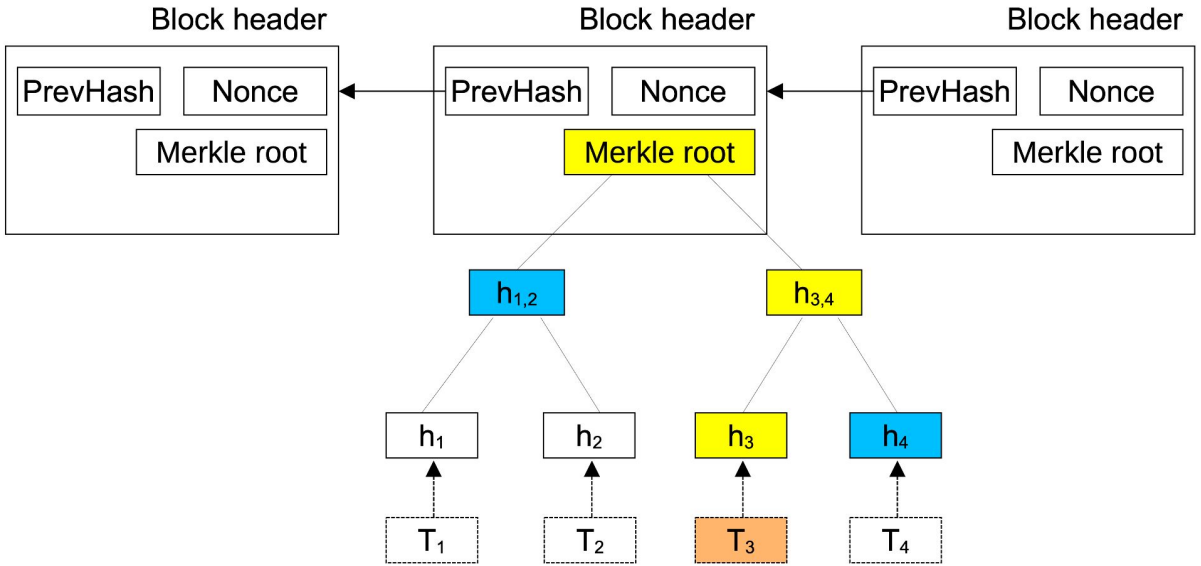
Notice that:

1) The verifier cannot access the data collection (only known to the prover).

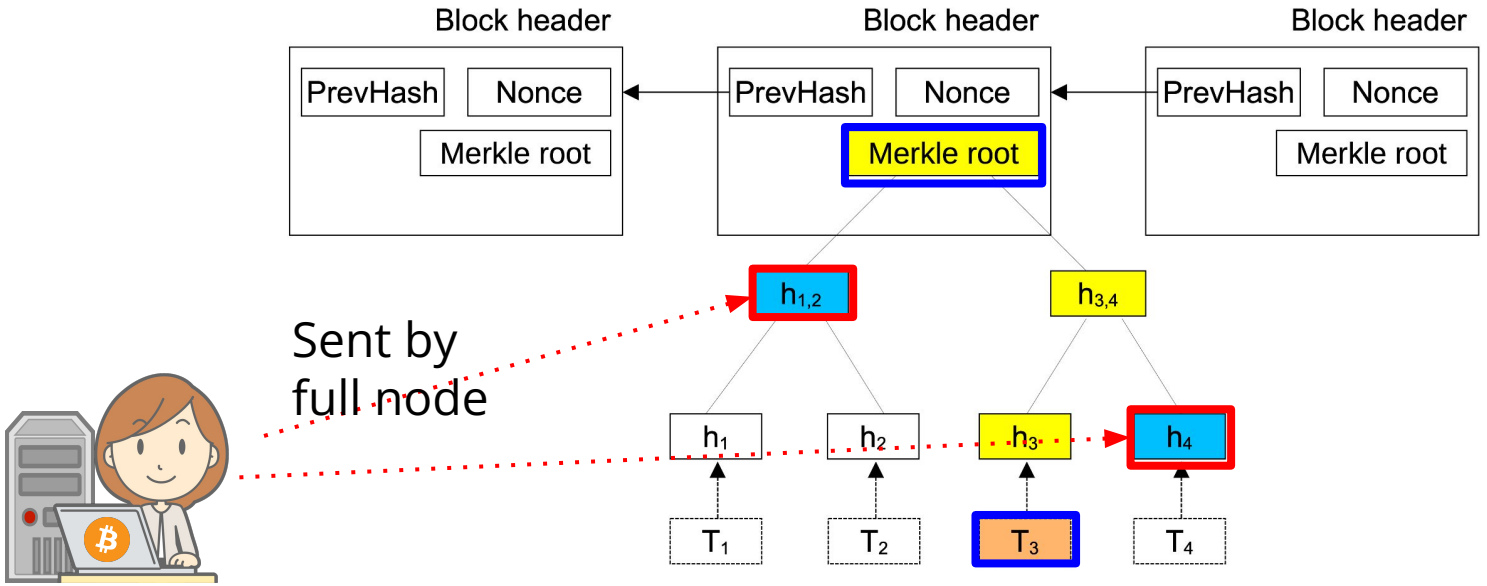
2) Instead, it maintains only a small piece of authentication information summarizing the current collection state.

Bitcoin Simplified Payment Verification

Query: is a transaction confirmed (i.e., included in a block)?



Bitcoin Simplified Payment Verification



Bitcoin Simplified Payment Verification

Block header

PrevHash | Nonce | Merkle root

Block header

PrevHash | Nonce | Merkle root

Block header

PrevHash | Nonce | Merkle root

$h_{1,2}$ $h_{3,4}$

h_1 h_2 h_3 h_4

T_1 T_2 T_3 T_4

Already known by light node

Sent by full node

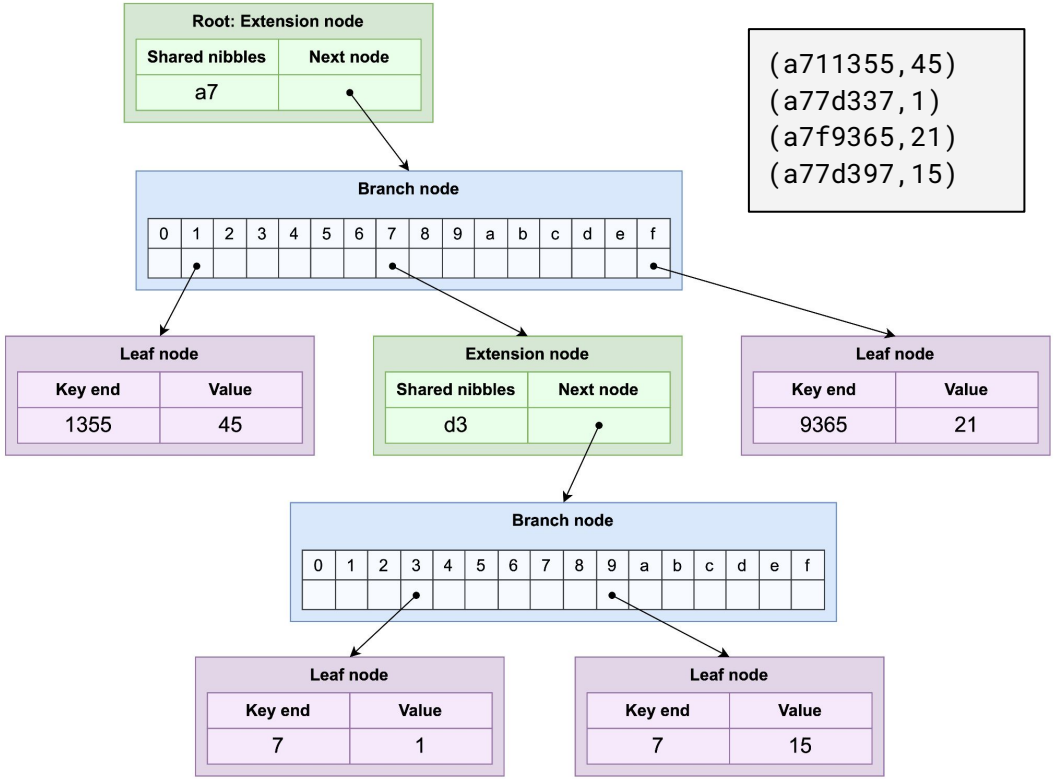
Ethereum Merkle Patricia Trie

Manages Ethereum state, i.e., a set of (account, balance) pairs.

Key-value store:

- Key determined by path.
- Value included in a leaf node.

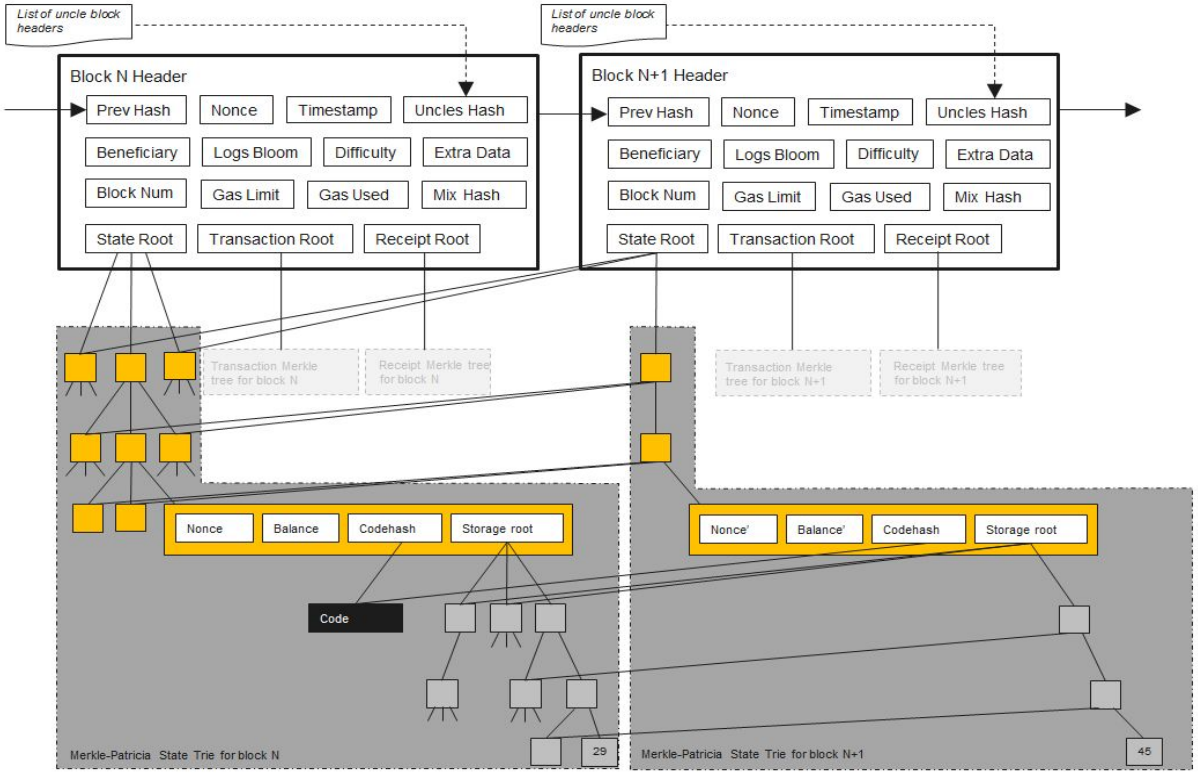
Supports both membership and non-membership proofs.



Ethereum Merkle Patricia Trie

The MPT root hash is embedded in each block header (*StateRoot* field).

Represents a commitment to current Ethereum state.



Ethereum Merkle Patricia Trie

```python
from trie import HexaryTrie

# Create a new Ethereum hexary MPT.
trie = HexaryTrie(db={})

# Add several key-value pairs to the MPT.
pairs = {
    b"cat": b"feline",
    b"dog": b"puppy",
    b"do": b"verb",
    b"doge": b"coin"
}
for key, value in pairs.items():
    trie[key] = value

# Generate a membership proof for a given key.
target_key = b"cat"
proof_nodes = trie.get_proof(target_key)

# Proof verification will return the associated value
# if the process succeeds.
value = HexaryTrie.get_from_proof(
    trie.root_hash,
    target_key,
    proof_nodes
)
print("Proof verified. Value:", value)

# For non-membership, it is sufficient to ask the MPT
# to generate a proof for a missing key.
target_missing_key = b"crocodile"
proof_missing = trie.get_proof(target_missing_key)

# Verification of the obtained proof will return no value.
value = HexaryTrie.get_from_proof(
    trie.root_hash,
    target_missing_key,
    proof_missing
)
print("Proof verified. Value:", value)

```

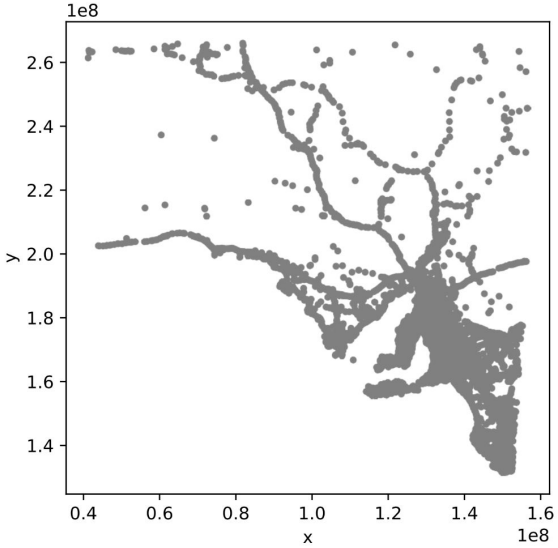
“Merkleization” of data structures

Many authenticated data structures are obtained by “Merkleizing” existing data structures (i.e., by enriching them with cryptographic information).

- Merkle R-tree (Merkle Tree + R-tree)
- Merkle B-tree (Merkle Tree + B-tree)
- Merkle kd-tree (Merkle Tree + kd-tree)
- Merkle Patricia Trie (Merkle Tree + PATRICIA trie)
- …

Merkle R-tree

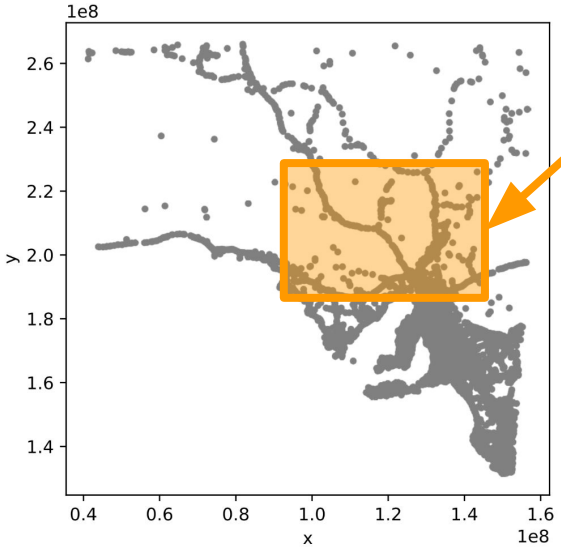
An ADS that combines Merkle trees and R-trees for range query authentication on **multidimensional** data (e.g., *spatial* data).

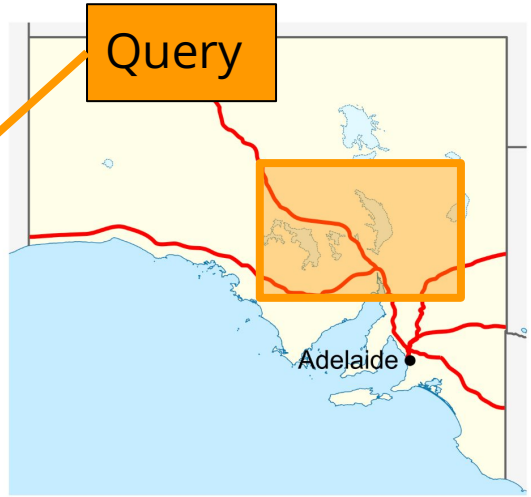


Source: [YPPK09] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, “Authenticated indexing for outsourced spatial databases,” VLDB J., vol. 18, no. 3, pp. 631–648, Jun. 2009.

Merkle R-tree

An ADS that combines Merkle trees and R-trees for range query authentication on **multidimensional** data (e.g., *spatial* data).





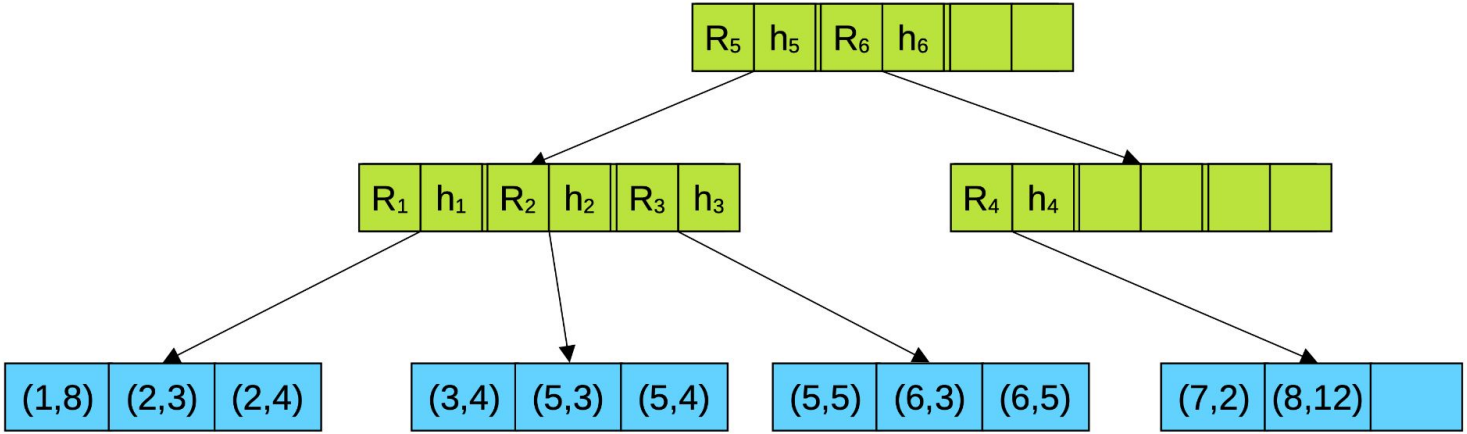
Range query (2D): return all points within a given (axis-aligned) rectangle.

$Q = (lx, ux, ly, uy) = \{(x, y) \in \mathbb{R}^2 : lx \leq x \leq ux \wedge ly \leq y \leq uy\}$

Source: [YPPK09] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, “Authenticated indexing for outsourced spatial databases,” VLDB J., vol. 18, no. 3, pp. 631–648, Jun. 2009.

Merkle R-tree

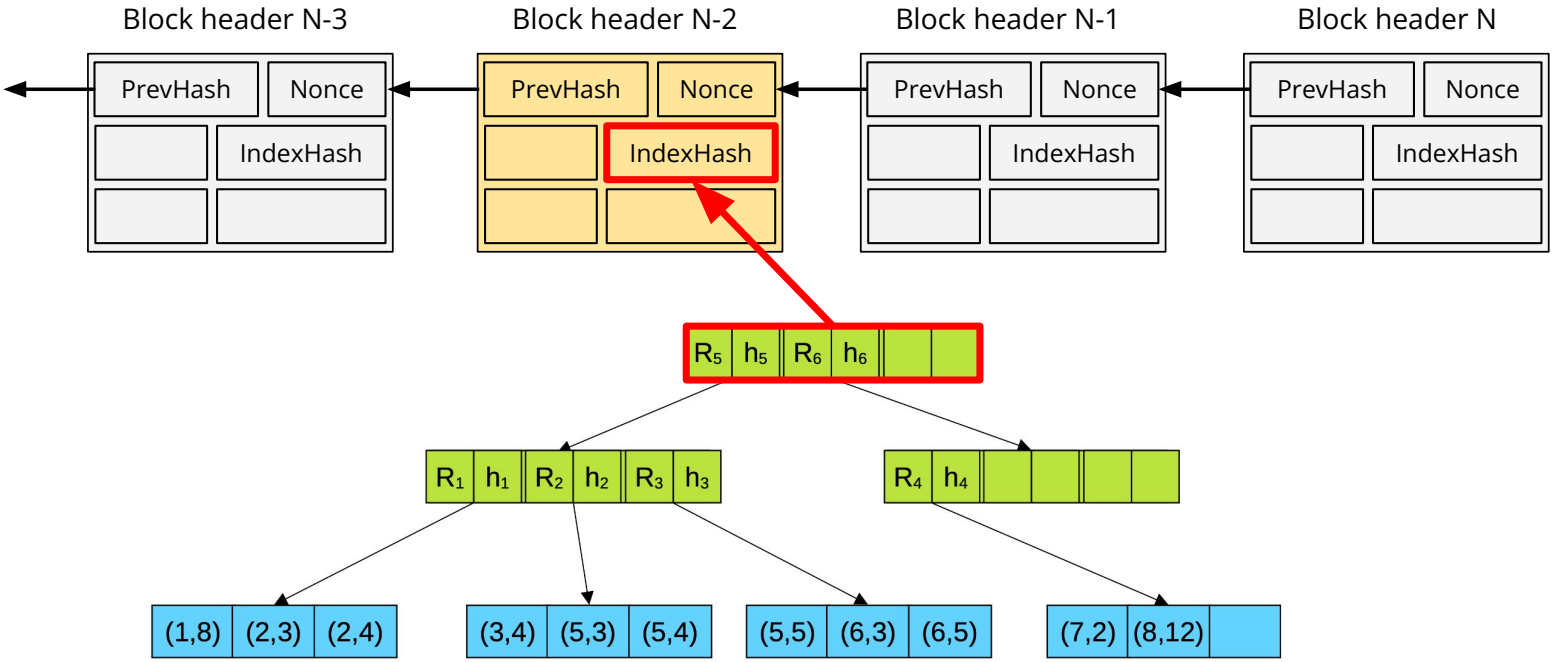
- c = **page capacity**
- **Leaf nodes:** at most c data points.
- **Internal nodes:** at most c entries (i.e., minimum bounding rectangle of subtree and cryptographic digest).



Source: [YPPK09] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, “Authenticated indexing for outsourced spatial databases,” VLDB J., vol. 18, no. 3, pp. 631–648, Jun. 2009.

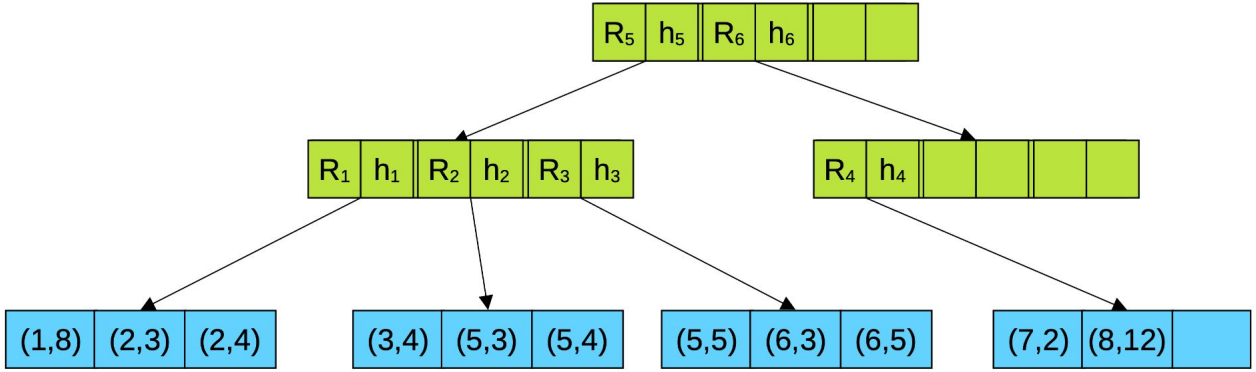
Merkle R-tree

Embedding root hash in block header enables authentication.



Source: [LBDR21] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa and L. Ricci. “Authenticating spatial queries on blockchain systems.” IEEE Access 9 (2021): 163363-163378.

Merkle R-tree



Query: recursively explore all subtrees whose rectangle overlaps with the query.

All records in a visited leaf are added to the VO, as well as rectangles and hashes of unexplored subtrees.

Verification: reconstruct the tree root starting from the received VO.

The reconstructed root hash is then compared with the value in the block header.

Merkle R-tree

The protocol guarantees that results are **correct**, i.e., that the following properties hold:

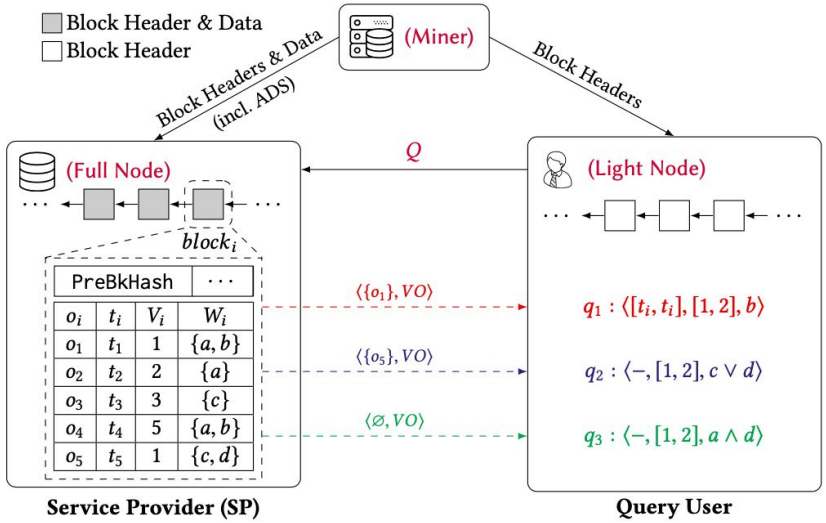
- **Authenticity:** results are not modified w.r.t. on-chain data.
- **Completeness:** no matching element is omitted from the results.

=> To break such properties and forge a fake proof, an attacker should find a collision for the cryptographic hash algorithm.

vChain

Framework for **verifiable Boolean range queries** over blockchain transaction records.

Supports both **intra-block** and **inter-block** queries.



o_i	t_i	V_i	W_i
o_1	t_1	1	$\{a, b\}$
o_2	t_2	2	$\{a\}$
o_3	t_3	3	$\{c\}$
o_4	t_4	5	$\{a, b\}$
o_5	t_5	1	$\{c, d\}$

Uses ADSs based on **bilinear accumulators** (or ESA-based construction).

Source: [XZX19] C. Xu, C. Zhang, and J. Xu, “vChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases,” Proceedings of the 2019 International Conference on Management of Data. ACM, Jun. 25, 2019.

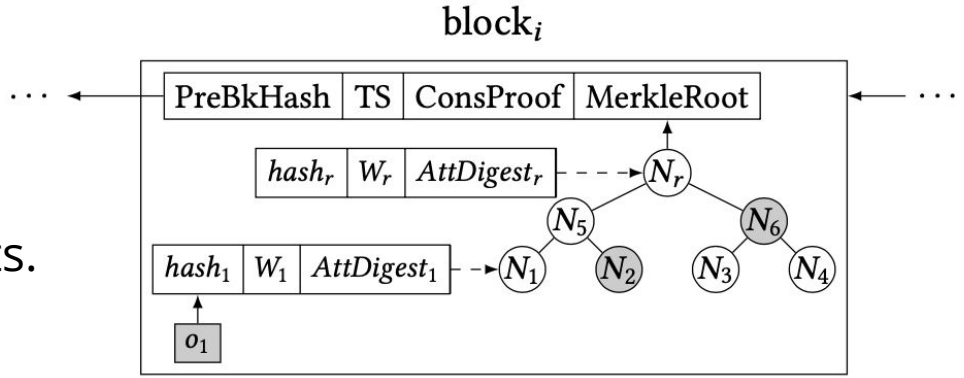
vChain

Each block stores a set of data objects.

Intra-block index

- Leaf node
  a. hash of data object;
  b. multiset with object attributes;
  c. accumulator.

- Internal node
  a. hash of child nodes;
  b. union of children multisets;
  c. accumulator.



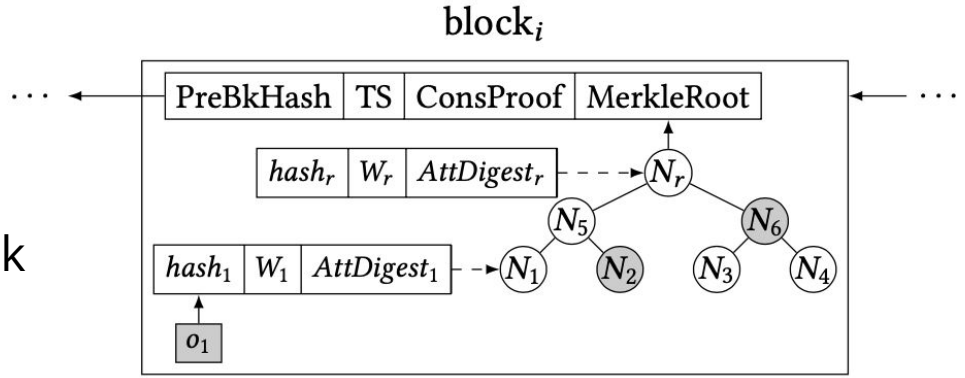
Node	Object	Set Attributes
N_1	o_1	W_1 = {"Sedan", "Benz"}
N_2	o_2	W_2 = {"Sedan", "Audi"}
N_3	o_3	W_3 = {"Van", "Benz"}
N_4	o_4	W_4 = {"Van", "BMW"}

Source: [XZX19] C. Xu, C. Zhang, and J. Xu, "vChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases," Proceedings of the 2019 International Conference on Management of Data. ACM, Jun. 25, 2019.

vChain

Validators build an ADS for each block and embed root digest into header.

Full node receives query and traverses the ADS in a top-down fashion to retrieve results and build VO.



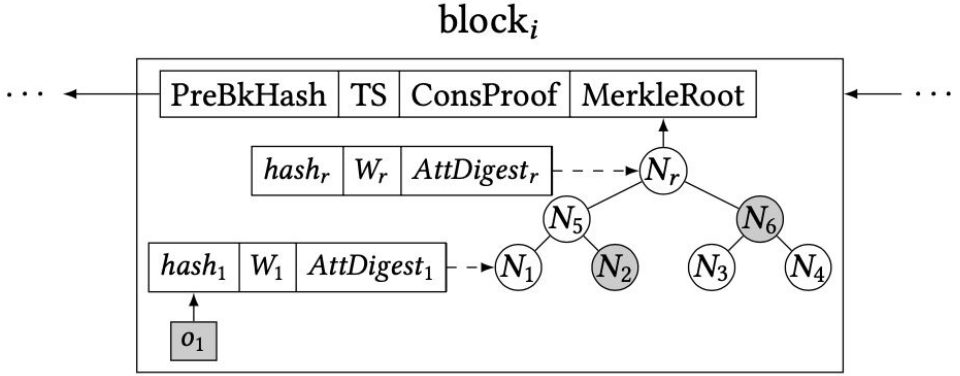
Node	Object	Set Attributes
N_1	o_1	W_1 = {“Sedan”, “Benz”}
N_2	o_2	W_2 = {“Sedan”, “Audi”}
N_3	o_3	W_3 = {“Van”, “Benz”}
N_4	o_4	W_4 = {“Van”, “BMW”}

Source: [XZX19] C. Xu, C. Zhang, and J. Xu, “vChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases,” Proceedings of the 2019 International Conference on Management of Data. ACM, Jun. 25, 2019.

vChain

When an internal node does not match the query, a non-membership proof is added to the VO.

=> All elements in the corresponding subtree do not match the query.



Node	Object	Set Attributes
N_1	o_1	W_1 = {“Sedan”, “Benz”}
N_2	o_2	W_2 = {“Sedan”, “Audi”}
N_3	o_3	W_3 = {“Van”, “Benz”}
N_4	o_4	W_4 = {“Van”, “BMW”}

Source: [XZX19] C. Xu, C. Zhang, and J. Xu, “vChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases,” Proceedings of the 2019 International Conference on Management of Data. ACM, Jun. 25, 2019.

Skip index

A data structure for efficient **inter-block** queries (and authentication).

Use case: searching for events along the Ethereum blockchain.

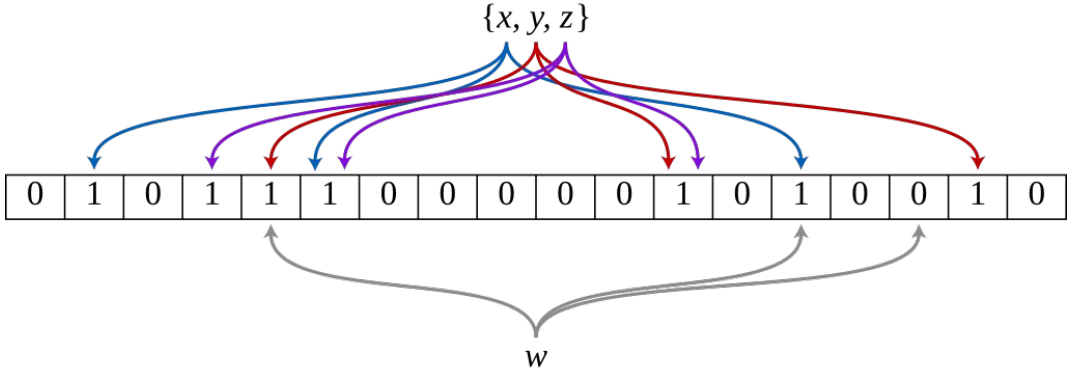
Membership queries: is a given event included in a certain block?

Source: [LBDR25] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa, and L. Ricci, “Skip index: Supporting efficient inter-block queries and query authentication on the blockchain,” Future Generation Computer Systems, vol. 164, p. 107556, Mar. 2025

Skip index

Headers of Ethereum blocks include a secure Bloom filter (*logsBloom*) that summarizes event logs.

- Filter size: 256 bytes
- Uses Keccak-256 cryptographic hash function for insertion/lookup.

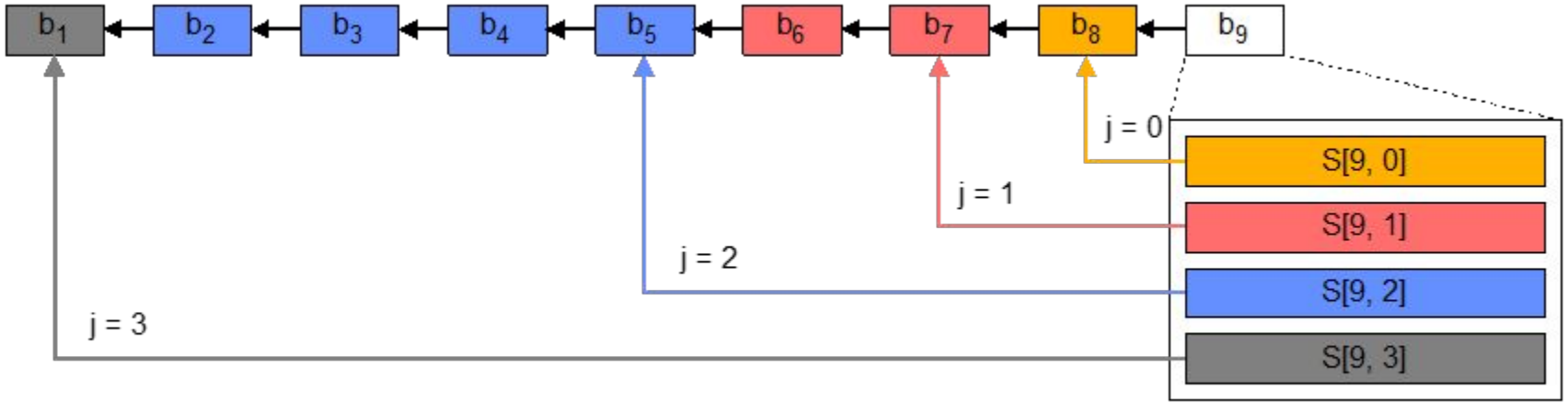


Source: [LBDR25] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa, and L. Ricci, “Skip index: Supporting efficient inter-block queries and query authentication on the blockchain,” Future Generation Computer Systems, vol. 164, p. 107556, Mar. 2025

```
jsonrpc:            "2.0"
id:                 1
result:
  difficulty:       "0x1913ff69551dac"
  extraData:        "0xe4b883e5bda9e7a59ee4bb99e9b1bc000921"
  gasLimit:         "0xe4e1b2"
  gasUsed:          "0xe4d737"
  hash:             "0xa917fcc721a5465a484e9be17cda0cc5493933dd3bc70c9adbee192cb419c9d7"
  logsBloom:        "0x00af00124b82093253a6960ab5a003170000318c0a00c18d418505009c10c905810e05d4a4511044b6245a06212201023395862 6c80039250781851410…
                    042610502402404092344 7ad1102000028b8f0e001e810021031840a2801831a0113b003a5485843004c10c4c10d6a04060a84d88500038ab10875a382c"
  miner:            "0x829bd824b016326a401d083b33d092293333a830"
  mixHash:          "0x7d416c4a24dc3b43898040ea788922d8563d44a5193e6c4a1d9c70990775c879"
  nonce:            "0xe6e41732385c71d6"
  number:           "0xc5043f"
  parentHash:       "0xd1c4628a6710d8dec345e5bca6b8093abf3f830516e05e36f419f993334d10ef"
  receiptsRoot:     "0x7eadd994da137c7720fe2bf2935220409ed23a06ec6470ffd2d478e41af0255b"
  sha3Uncles:       "0x7d9ce61d799ddcb5dfe1644ec7224ae7018f24ecb682f077b4c477da192e8553"
  size:             "0xa244"
  stateRoot:        "0x6350d0454245fb410fc0fb93f6648c5b9047a6081441e36f0ff3ab259c9a47f0"
  timestamp:        "0x6100bc82"
  transactions:
    [0…99]:
    [100…199]:
    [200…203]:
  transactionsRoot: "0xa17c2a87a6ff2fd790d517e48279e02f2e092a05309300c976363e47e0012672"
  uncles:
    0:              "0xd3946359c70281162cf00c8164d99ca14801e8008715cb1fad93b9cecaf9f7d8"
```

Skip index

Each block is enhanced with a sequence of Bloom filters summarizing the content of its predecessors at an exponentially increasing distance.



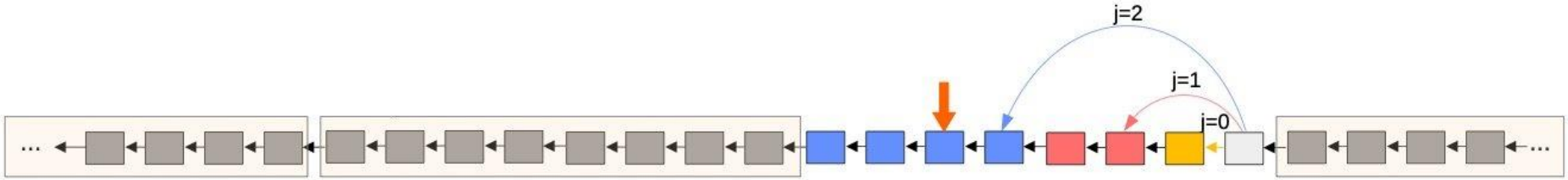
Source: [LBDR25] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa, and L. Ricci, “Skip index: Supporting efficient inter-block queries and query authentication on the blockchain,” Future Generation Computer Systems, vol. 164, p. 107556, Mar. 2025

Skip index

Skip index with E entries (i.e., filters).

Goal: Find first (i.e., most recent) occurrence of event x within a range of blocks [l, u].

1. Partition [l, u] in chunks of 2^E blocks.
2. Recursively explore the range corresponding to the first matching filter.



Source: [LBDR25] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa, and L. Ricci, “Skip index: Supporting efficient inter-block queries and query authentication on the blockchain,” Future Generation Computer Systems, vol. 164, p. 107556, Mar. 2025

Skip index

Query authentication process

1. Hash of skip index included in block header
2. Full node builds VO as the ordered sequence of indices of visited blocks, i.e., VO = (S_1, ..., S_v)
3. Light node uses VO to perform a guided search
   a. Compares hash(S_i) with header of block b_i
   b. Uses S_i to determine next block to be visited

=> Verification complexity = search complexity

Source: [LBDR25] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa, and L. Ricci, “Skip index: Supporting efficient inter-block queries and query authentication on the blockchain,” Future Generation Computer Systems, vol. 164, p. 107556, Mar. 2025

Skip index

- Bloom filters are **symmetric** accumulators (i.e., no need to compute a witness to check membership).

- Optimal size also depends on the number of elements in the set
  => Bloom filters are less space-efficient w.r.t. accumulators.

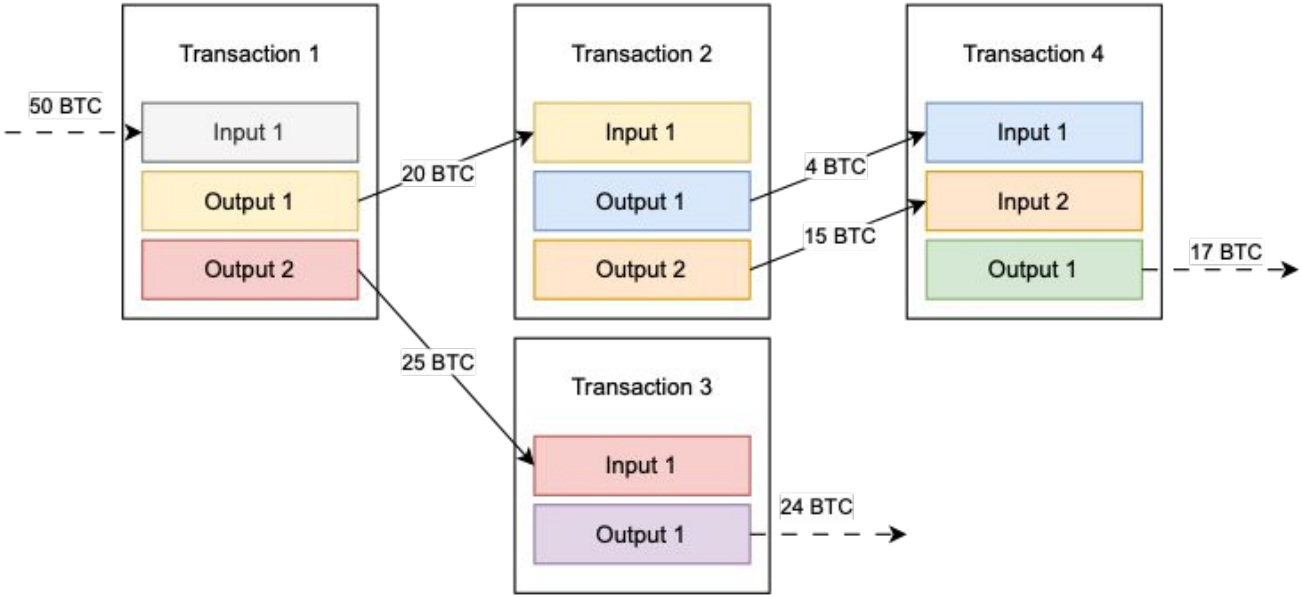
- Hashing operations are more efficient than EC operations (and do not require trusted setup).

Source: [LBDR25] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa, and L. Ricci, “Skip index: Supporting efficient inter-block queries and query authentication on the blockchain,” Future Generation Computer Systems, vol. 164, p. 107556, Mar. 2025

Stateless Transaction Validation

Transactions

In the UTXO model, new transactions spend outputs of previous transactions.



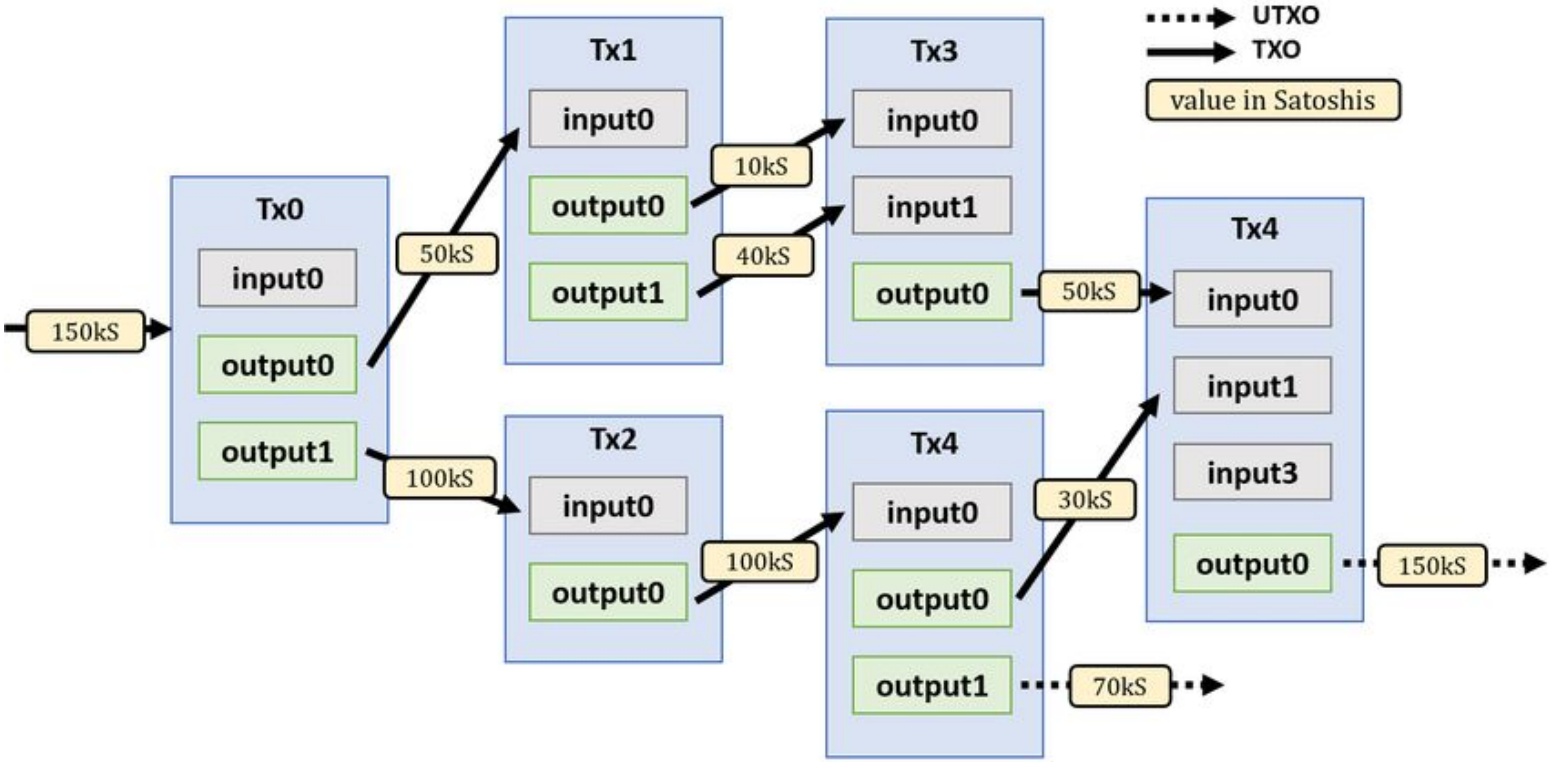
Transaction validation

Valid transaction: input funds correspond to unspent funds.

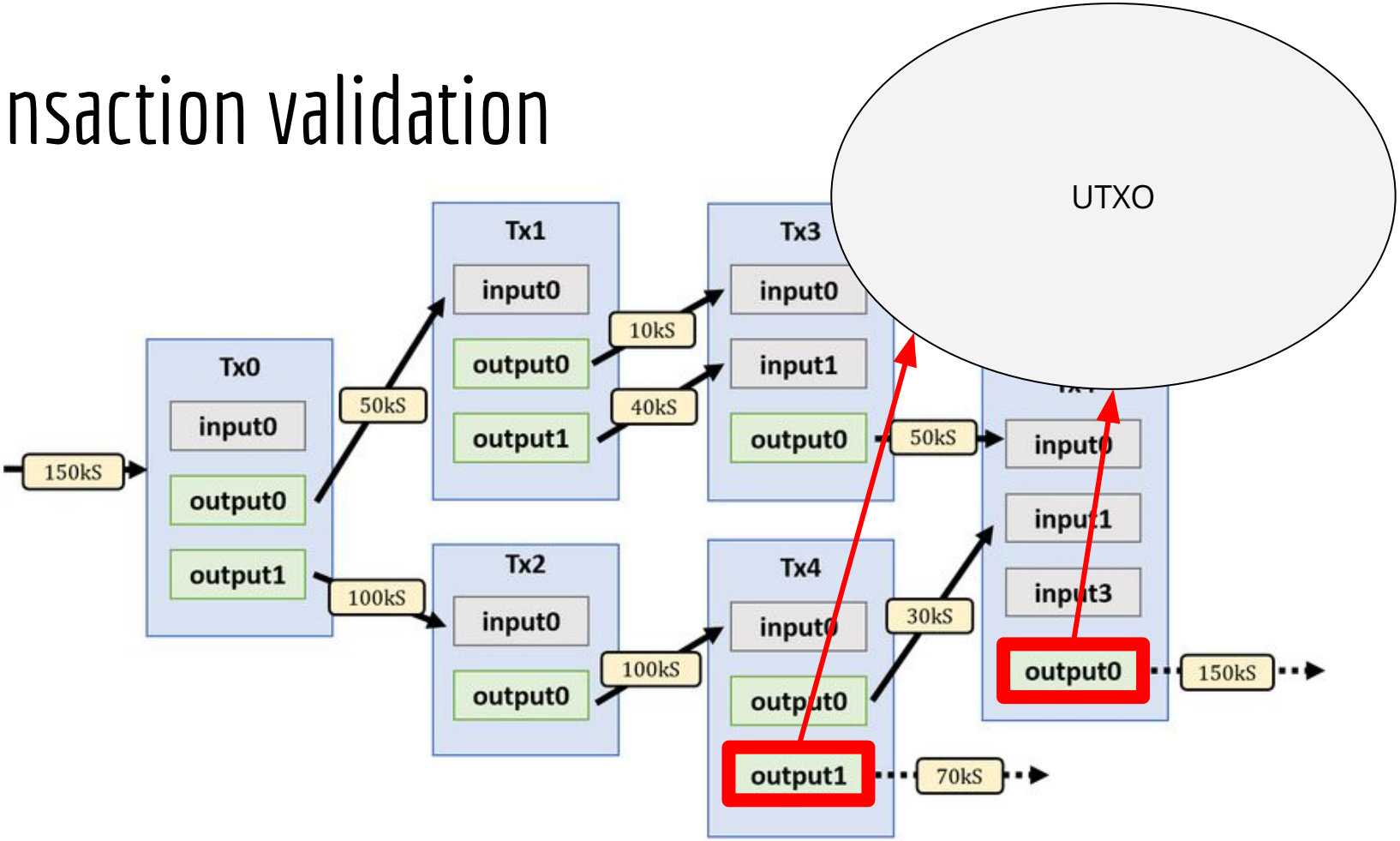
1) TX inputs coincide with outputs of previously confirmed transactions.

2) No other confirmed TX uses these funds as its inputs.

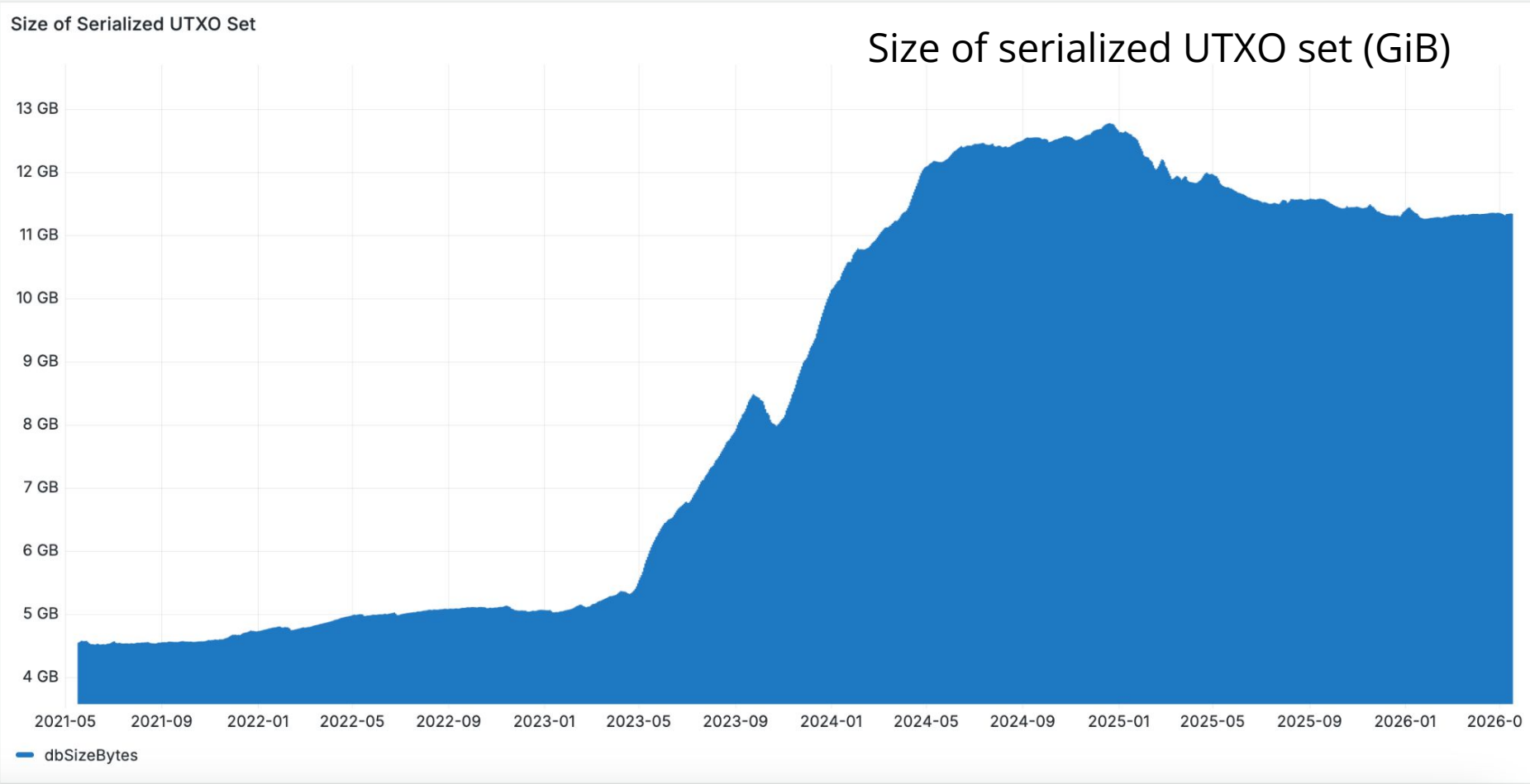
Transaction validation



Transaction validation



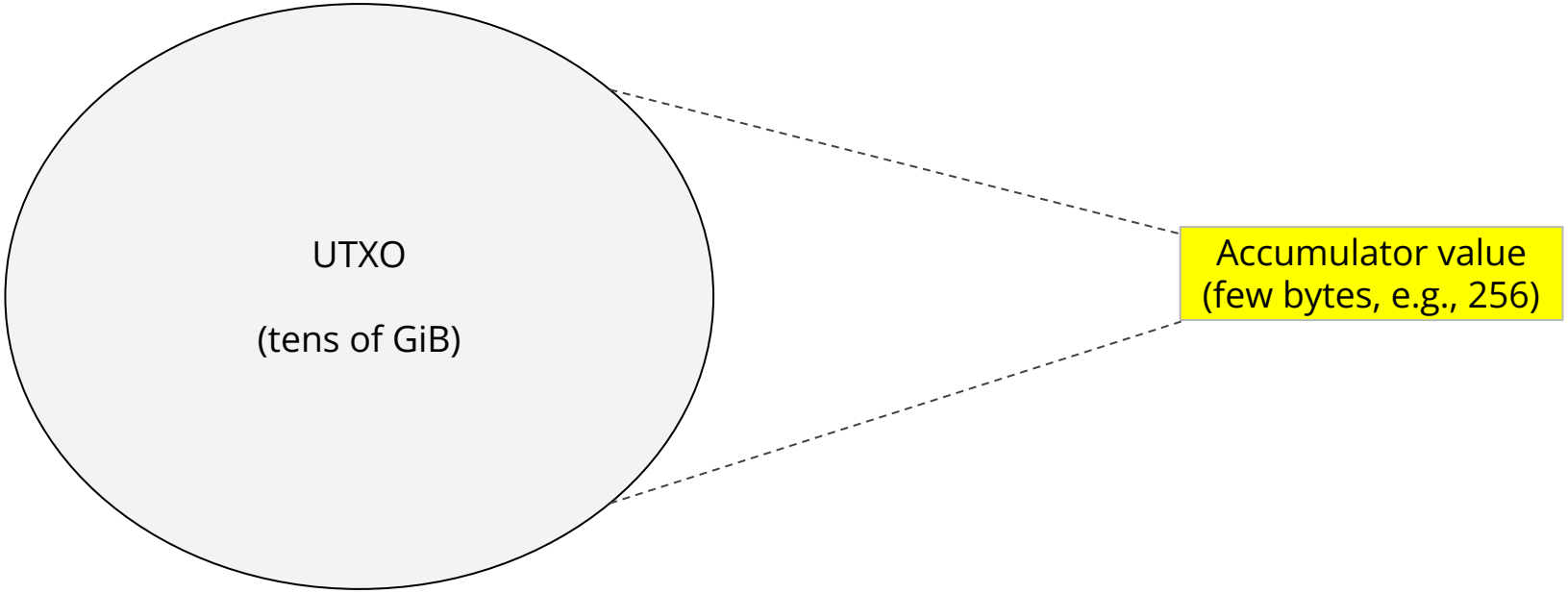
Size of serialized UTXO set (GiB)



Source:
https://statoshi.info/d/000000009/unspent-transaction-output-set?orgId=1&theme=light&from=2021-04-30T22:00:00.000Z&to=now&timezone=browser&refresh=10m&viewPanel=panel-8

Transaction validation

Key idea: Compress UTXO set using set accumulators.



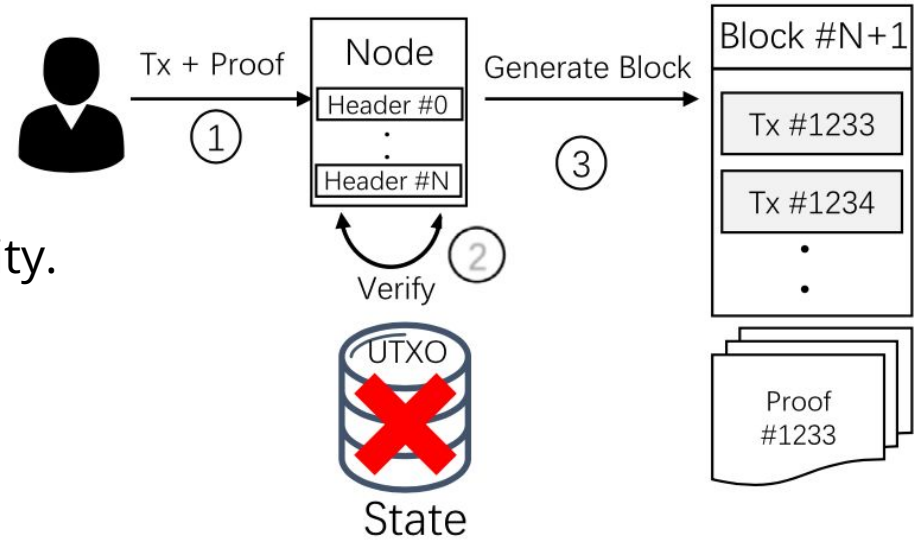
Transaction validation

Stateless design

Validator nodes assess the validity of a transaction by simply checking whether the inputs belong to the accumulator, instead of examining the entire set of unspent outputs.

=> Proof of transaction validity: proof of inputs membership in UTXO set.

Transaction validation



1) TX creators attach proof of validity.

2) Light validator node verifies TX
against the accumulator in block header.
If all inputs belong to the UTXO set, then the transaction is valid.

3) TX is added to the new block.

Source: [CW20] H. Chen, Y. Wang, “MiniChain: A lightweight protocol to combat the UTXO growth in public blockchain”, J. Parallel Distrib. Comput. 143 (2020) 67–76.

Transaction validation

- Boneh et al. [BBF19] first proposed an RSA-based accumulator with **batch deletions** and **insertions**.

- No centralized trusted accumulator manager (perfect for untrusted environments).

- Accumulator value embedded in each block header (i.e., commitment to latest UTXO state).

Source: [BBF19] D. Boneh, B. Bünz, B. Fisch. “Batching techniques for accumulators with applications to IOPs and stateless blockchains”. In: A. Boldyreva, D. Micciancio (Eds.), Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I, in: Lecture Notes in Computer Science, vol. 11692, Springer, 2019, pp. 561–586.

Transaction validation

MiniChain protocol [CW20]

The UTXO set is split in two data structures:

1) **Spent Transaction Outputs (STXO)** set.
2) **Transaction Outputs (TXO)** set.

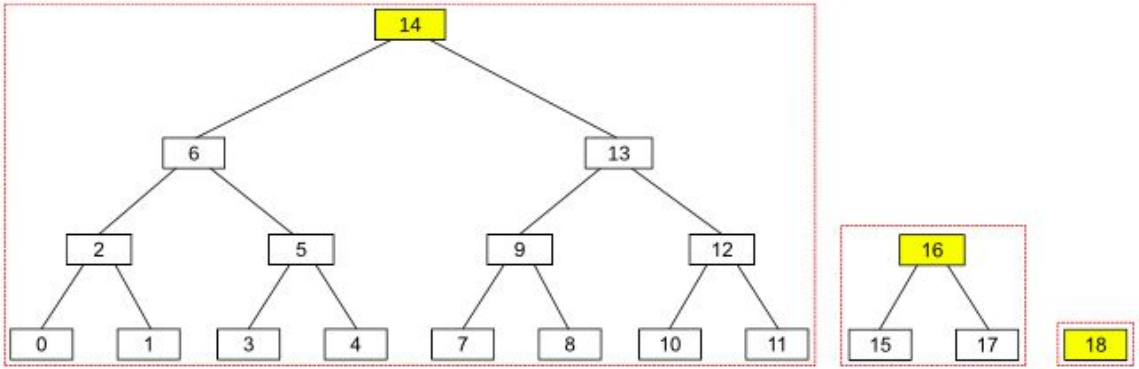
An output can be spent if and only if:

1) It belongs to TXO (=> membership proof)
2) It does not belong to STXO (=> non-membership proof)

Source: [CW20] H. Chen, Y. Wang, “MiniChain: A lightweight protocol to combat the UTXO growth in public blockchain”, J. Parallel Distrib. Comput. 143 (2020) 67–76.

Transaction validation

TXO is managed with a **Merkle Mountain Range (MMR)** (i.e., append-only list of Merkle Trees)

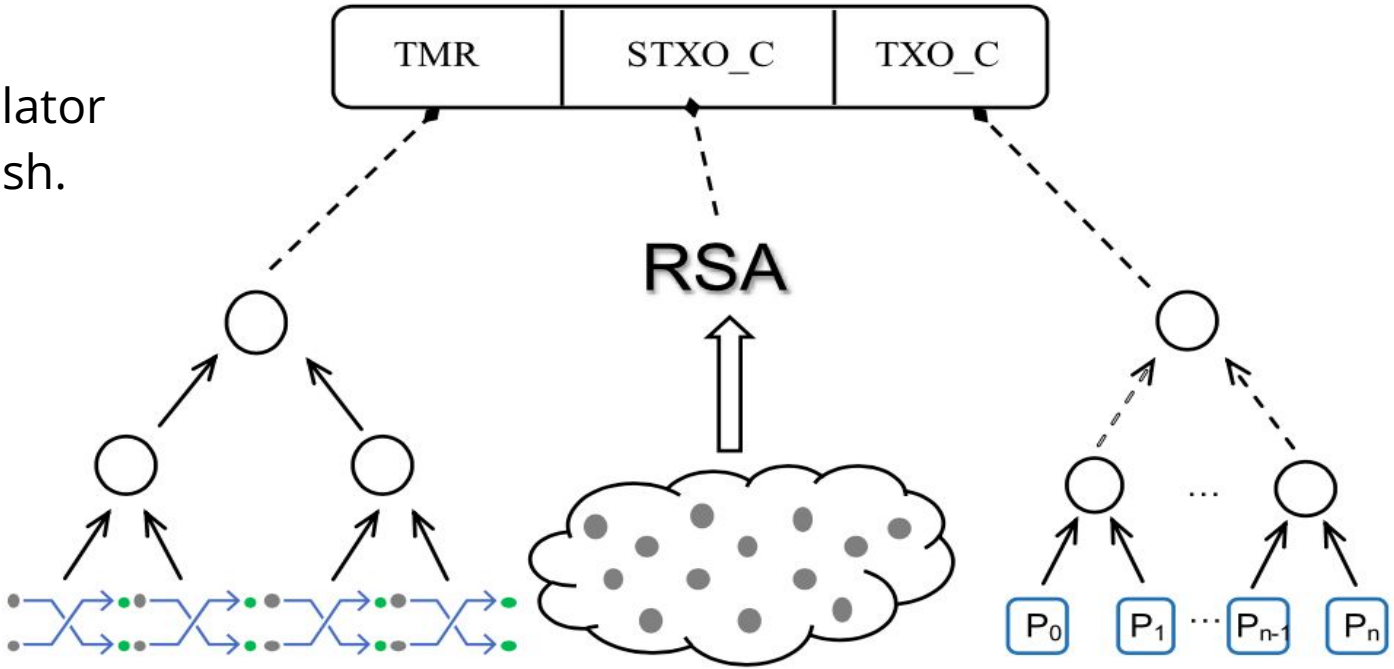


STXO is managed with a **dynamic RSA-based accumulator** (must support element insertion).

Source: [CW20] H. Chen, Y. Wang, “MiniChain: A lightweight protocol to combat the UTXO growth in public blockchain”, J. Parallel Distrib. Comput. 143 (2020) 67–76.

Transaction validation

Each block header stores RSA accumulator and MMR peaks hash.



Source: [CW20] H. Chen, Y. Wang, “MiniChain: A lightweight protocol to combat the UTXO growth in public blockchain”, J. Parallel Distrib. Comput. 143 (2020) 67–76.

Anonymity enhancement

Anonymity enhancement

Bitcoin is not really anonymous

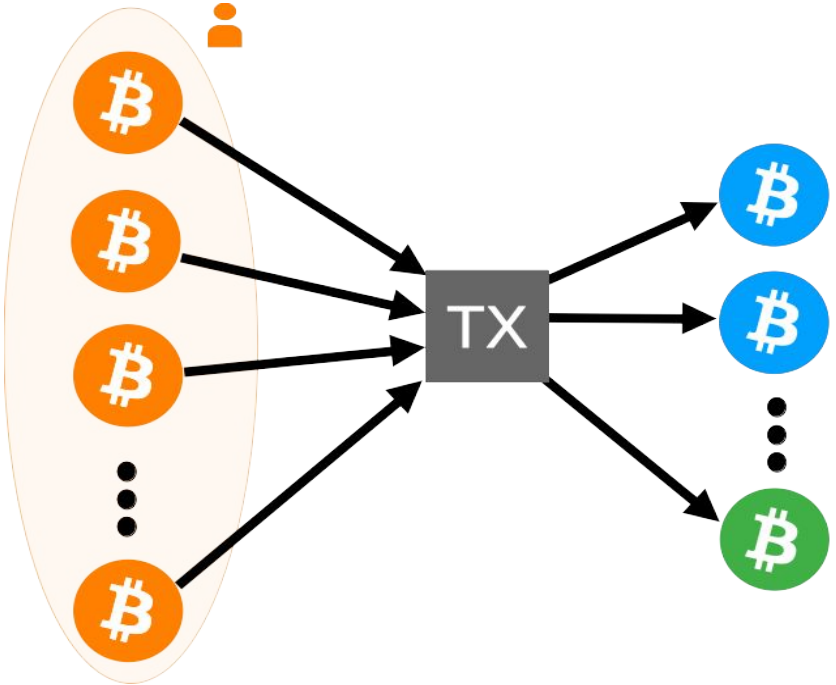
- Transactions are public
- Users control multiple addresses acting as pseudonyms
- Users can be **deanonymized** (attackers can also exploit off-network information)

Source: https://upload.bitpapa.com/uploads/blog_item/thumbnail/402/128_The_Ultimate_Guide_to_Buying_Crypto_with_Anonymous_Bitcoin.JPG

Anonymity enhancement

Multi-input heuristic rule

If two (or more) addresses are used as inputs to the same transaction, they are likely to be controlled by the same user.



Source: [Nak08] S. Nakamoto. “Bitcoin: A peer-to-peer electronic cash system.” 2008.

Anonymity enhancement

Bitcoin **laundries** mix together funds from different users to obfuscate TX history.

Drawbacks:

- Service must be trusted to return coins.
- Compromised or malicious laundry offers no anonymity.

Anonymity enhancement

Anonymity:

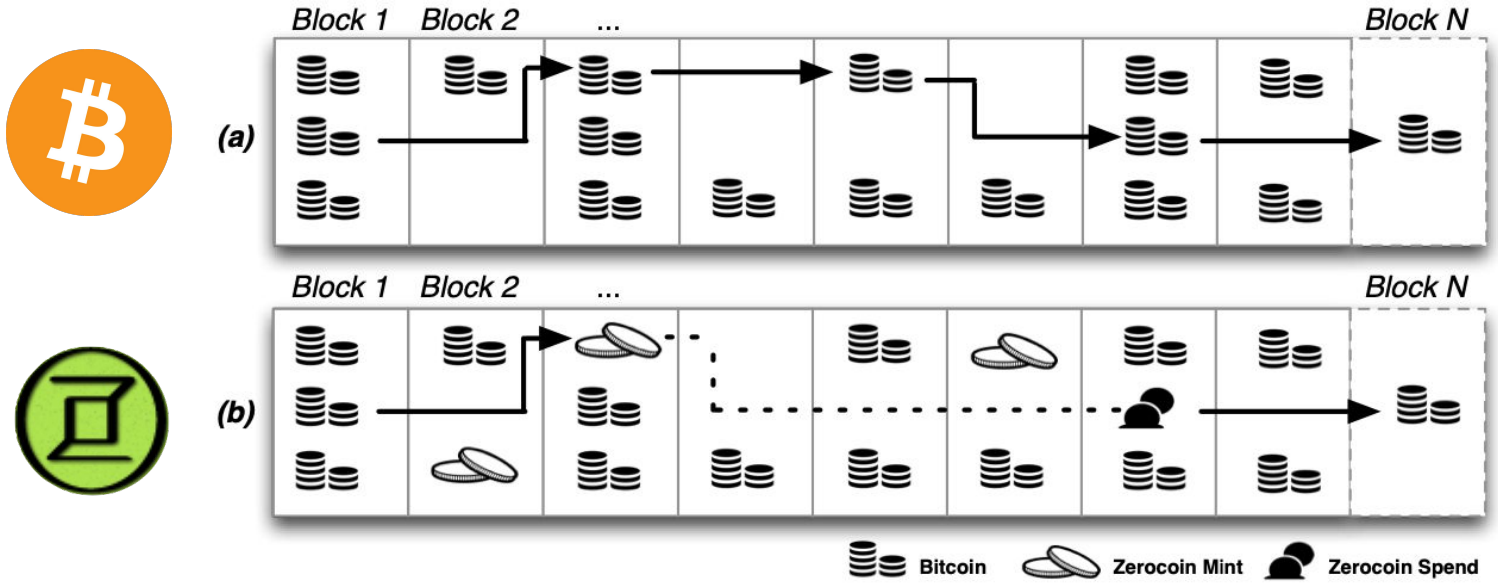
Neither the origin, destination, or amount of a payment should be revealed.

Zerocoin

- Decentralized and anonymous payment scheme.
- Born as an extension to the Bitcoin protocol (i.e., it is built on top of Bitcoin).
- Later evolved into Zerocash and Zcash.

Source: [MGGR13] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoin: Anonymous Distributed E-Cash from Bitcoin,” 2013 IEEE Symposium on Security and Privacy. IEEE, pp. 397–411, May 2013.

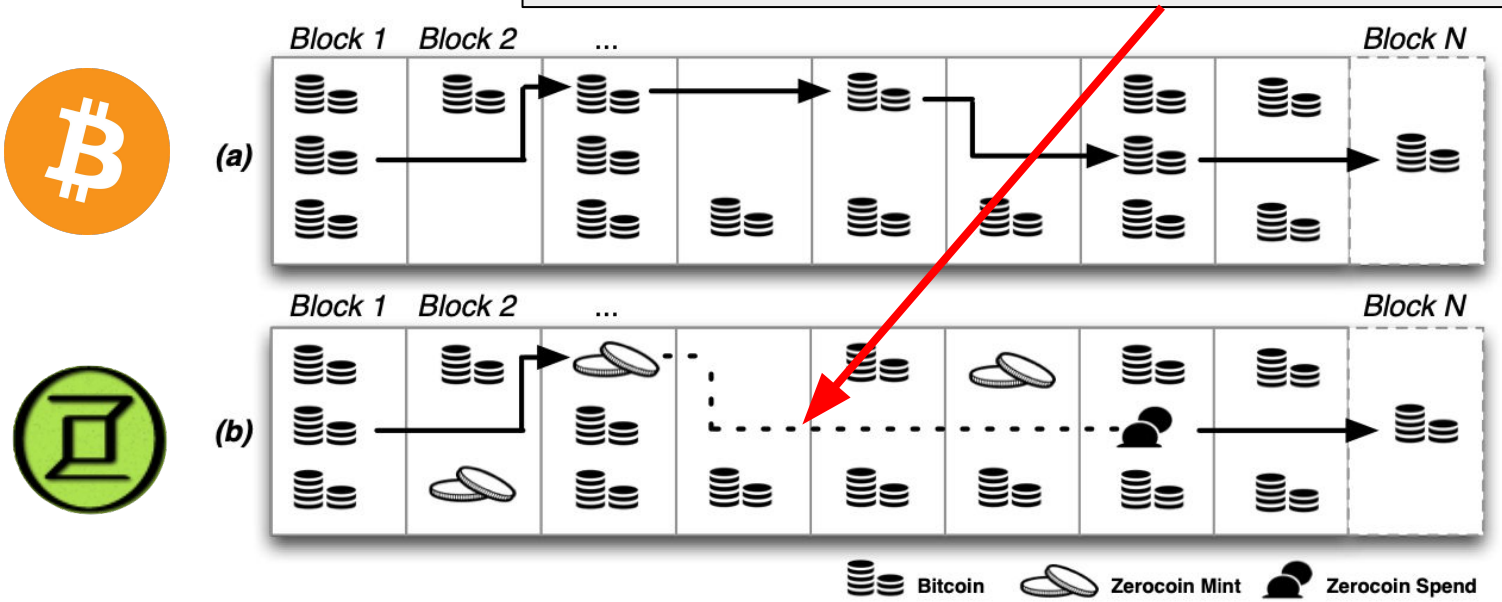
Zerocoin



Source: [MGGR13] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoin: Anonymous Distributed E-Cash from Bitcoin,” 2013 IEEE Symposium on Security and Privacy. IEEE, pp. 397–411, May 2013.

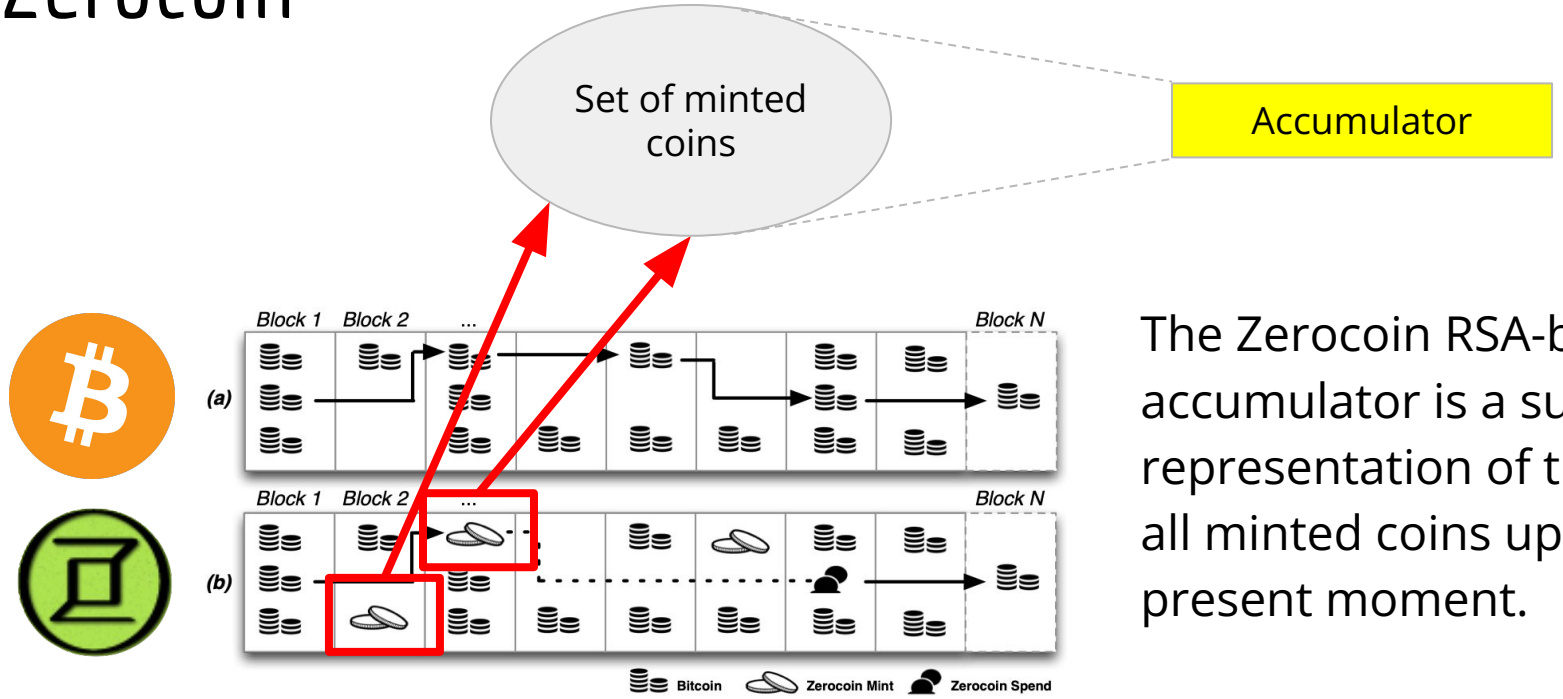
Zerocoin

True anonymity: Break link between the address that was used to create a certain coin and address used to redeem the same coin (without trusted parties).



Source: [MGGR13] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoin: Anonymous Distributed E-Cash from Bitcoin,” 2013 IEEE Symposium on Security and Privacy. IEEE, pp. 397–411, May 2013.

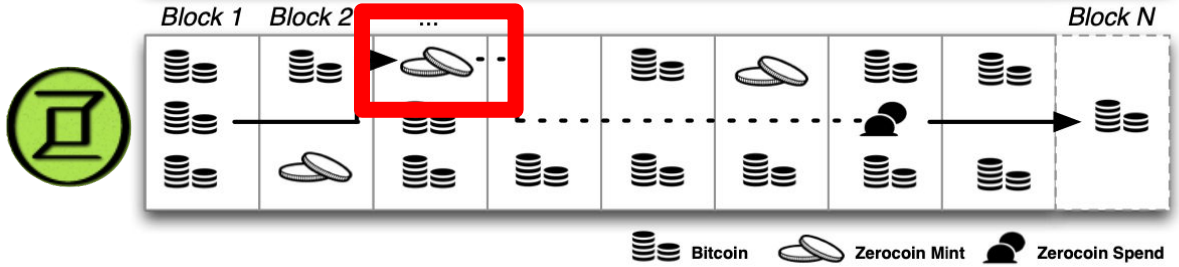
Zerocoin



The Zerocoin RSA-based accumulator is a succinct representation of the set of all minted coins up to the present moment.

Source: [MGGR13] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoin: Anonymous Distributed E-Cash from Bitcoin,” 2013 IEEE Symposium on Security and Privacy. IEEE, pp. 397–411, May 2013.

Zerocoin



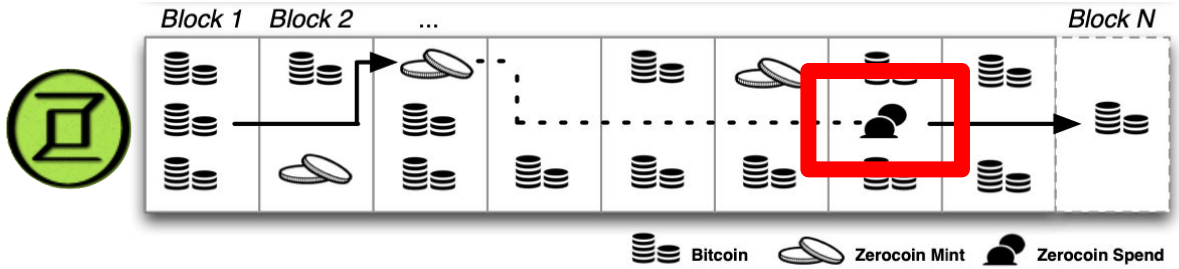
Minting a coin

1) User U generates random **serial number** S and a **nonce** r (secret);
2) C = commit(S, r) (with C prime);
3) Miners insert C into the accumulator.

Pedersen commitment scheme for step (2)
RSA-based accumulator for step (3)

Source: [MGGR13] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoin: Anonymous Distributed E-Cash from Bitcoin,” 2013 IEEE Symposium on Security and Privacy. IEEE, pp. 397–411, May 2013.

Zerocoin



Spending a coin

U produces two Zero-Knowledge proofs:

1) U has a commitment inserted into the accumulator.
2) U discloses S and proves the knowledge of r s.t. r and S correspond to a commitment (i.e., a coin).

Step (1) does not reveal C.

Step (2) does not reveal r.

Source: [MGGR13] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoin: Anonymous Distributed E-Cash from Bitcoin,” 2013 IEEE Symposium on Security and Privacy. IEEE, pp. 397–411, May 2013.

Zerocoin

Thanks to ZK proofs, the redeemed funds cannot be linked to a specific and previously minted coin.

To link coin C to the the serial number S used in the withdrawal, one must:

1) either know r (secret) ;

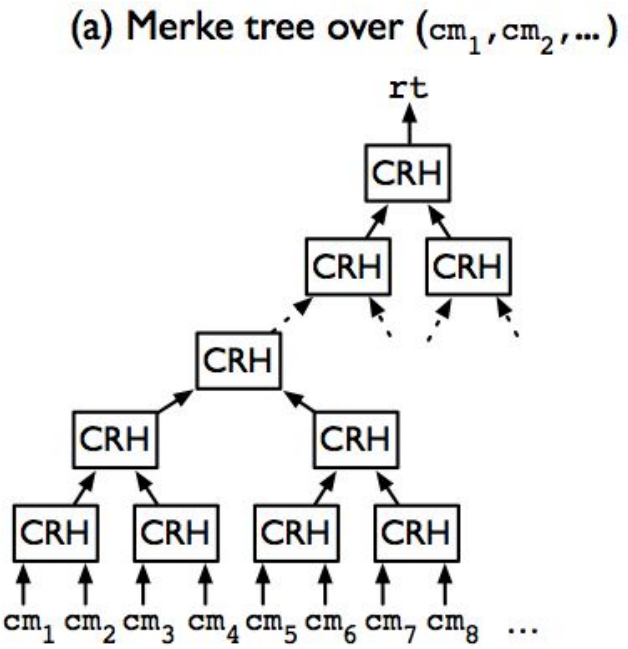
2) directly know which coin the user proved knowledge of, neither of which are revealed by the proof.

Zerocash

Zerocash

Nodes maintain a **Merkle tree** over all coin commitments seen so far.

Users demonstrate ownership of a coin commitment, via its decommitted values and a Merkle proof.



Source: [BCC+14] E. Ben Sasson et al., “Zerocash: Decentralized Anonymous Payments from Bitcoin,” 2014 IEEE Symposium on Security and Privacy. IEEE, May 2014.

Identity management

Identity management

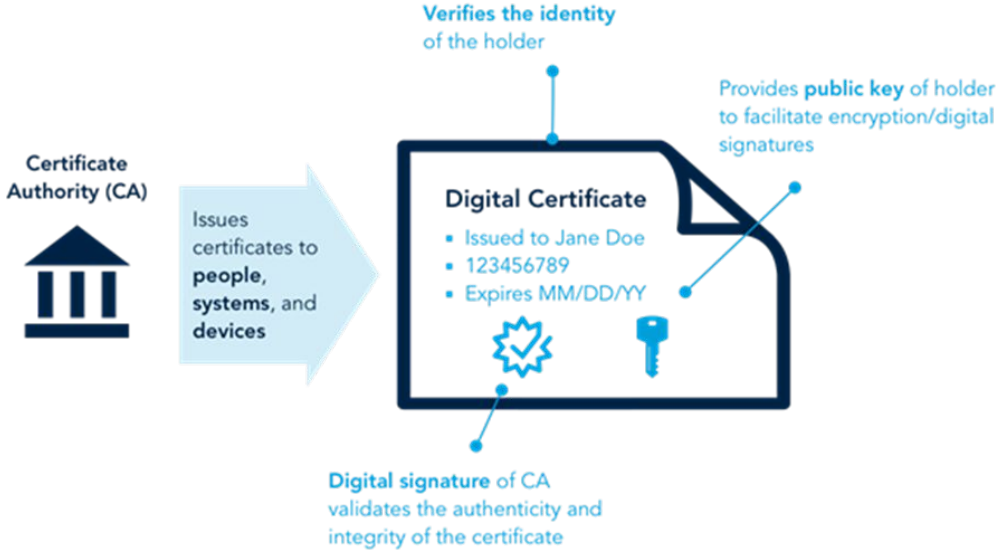
Blockchain technology plays a key role within two different paradigms.

1) **Decentralized Public Key Infrastructures (PKIs)**

2) **Self-Sovereign Identities (SSIs)**

Decentralized PKIs

Public key (PK) certificates associate identities (e.g., websites, individuals or organizations) to cryptographic keys.



Source: https://id4d.worldbank.org/sites/id4d-ms8.extcc.com/files/inline-images/18%20digital%20certificates.png

Decentralized PKIs

CAs should be trusted not to maliciously exploit all the collected information, but also to effectively protect it from external attacks.

=> Single point of failure!

Idea: Replace CAs with Distributed Ledger Technology (DLT).

DLT is employed to maintain a database of (identity, public key) associations.

Decentralized PKIs

Certificates are recorded on chain => Storage overhead

How to perform identity authentication and assess certificate validity?

Not so efficient.

- As ledger size grows storage overhead is growing.

Should also be able to support certificate updates and revocation: PKI should (e.g., when public key becomes compromised)

Certcoin

Decentralized PKI protocol that uses the Namecoin blockchain as a public bulletin board.

Idea: Build a **Merkle tree** on the set of current (identity, public key) pairs.

- Assess pair validity through membership proofs.
- Complexity decreases from O(N) to O(log N) where N is the number of registered identities.

Source: [FVY14] C. Fromknecht, D. Velicanu, and S. Yakoubov. "A decentralized public key infrastructure with identity retention." Cryptology ePrint Archive, 2014.

Certcoin

- System maintains a global Merkle tree (current state is recorded on-chain).

- When a public key is created, updated or revoked the miner updates accumulator and corresponding witnesses.

- Users can check that accumulator correctly incorporates (resp. does not incorporate anymore) the newly added (resp. removed) values.

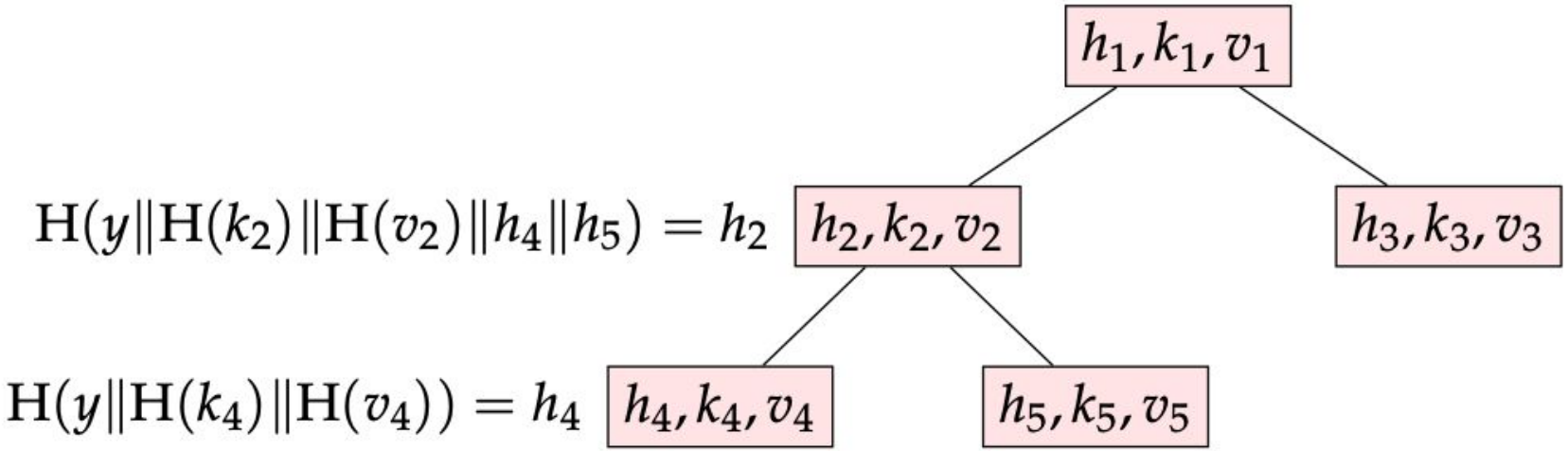
DPKIT

- Decentralized PKI for managing X.509 certificates (stored on chain).
- Ensures certificate transparency (publicly auditable, anyone can verify certificates or detect misbehavior).
- Provides a scheme for indexing certificates, proving their validity and occurred revocation.

Source: [BHMS21] X. Boyen, U. Herath, M. McKague, and D. Stebila, “Associative Blockchain for Decentralized PKI Transparency,” Cryptography, vol. 5, no. 2, p. 14, May 2021.

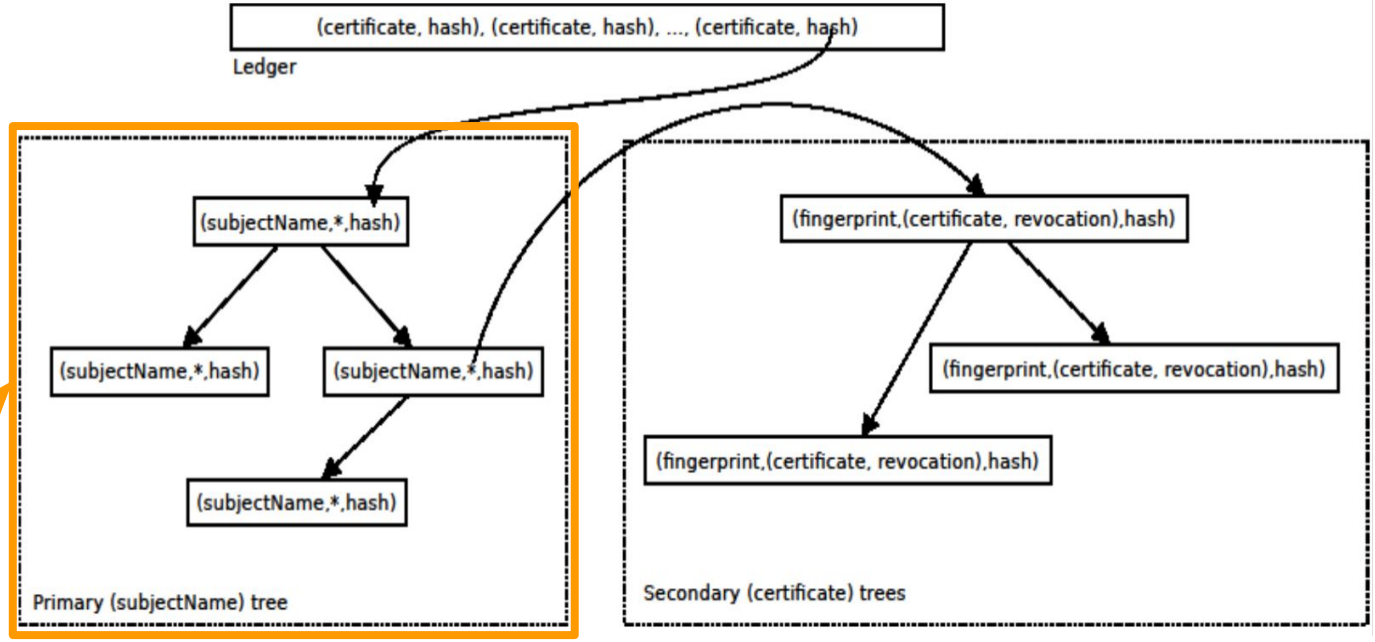
DPKIT

Merkle Binary Search tree (Merkle BST): stores key-value bindings.



Source: [BHMS21] X. Boyen, U. Herath, M. McKague, and D. Stebila, “Associative Blockchain for Decentralized PKI Transparency,” Cryptography, vol. 5, no. 2, p. 14, May 2021.

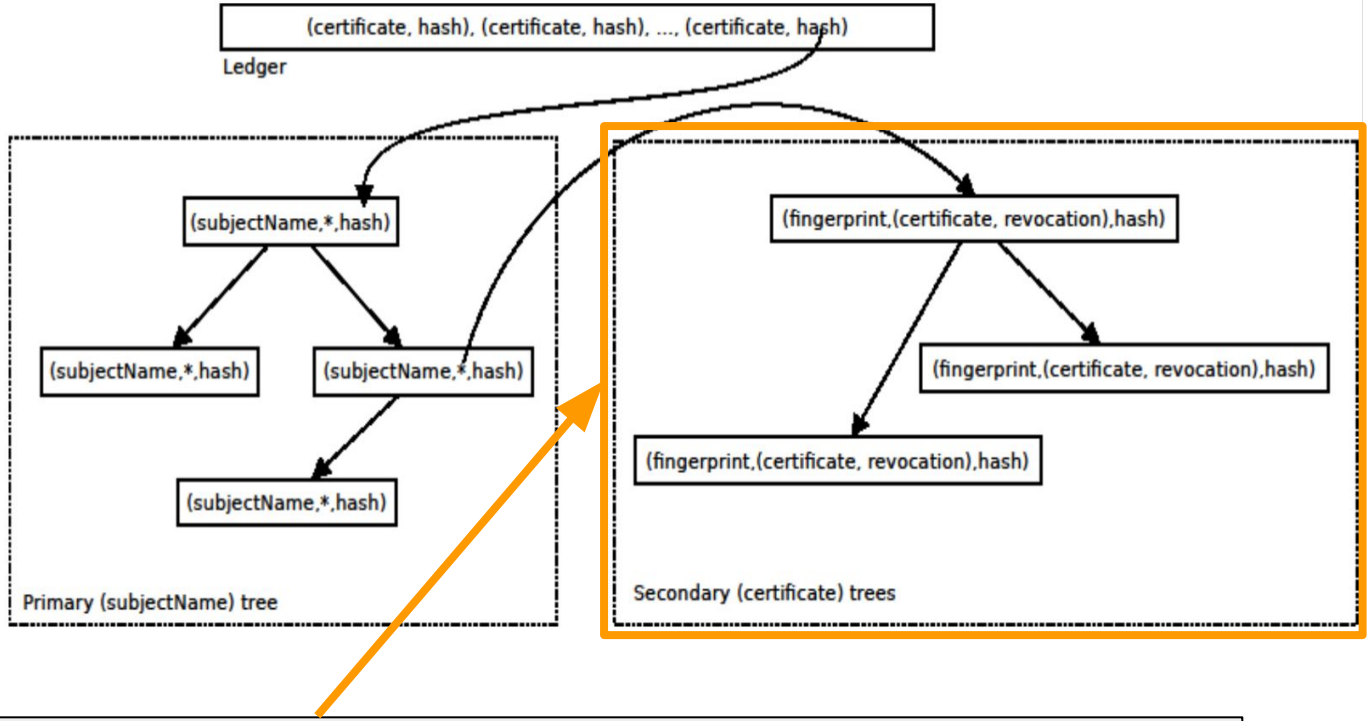
DPKIT



Primary tree: A Merkle BST which stores pointers to certificate trees, indexed by subject names.

Source: [BHMS21] X. Boyen, U. Herath, M. McKague, and D. Stebila, “Associative Blockchain for Decentralized PKI Transparency,” Cryptography, vol. 5, no. 2, p. 14, May 2021.

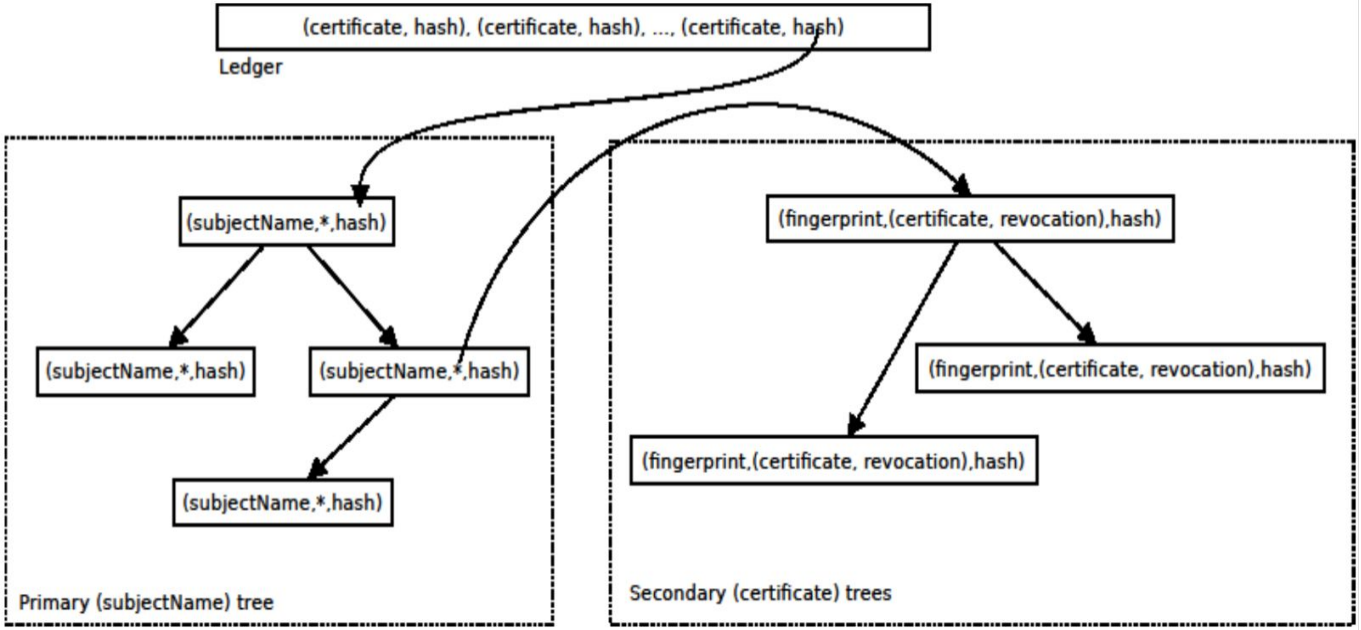
DPKIT



Secondary tree: stores (certificate, revocation) pairs with certificate fingerprints as keys

Source: [BHMS21] X. Boyen, U. Herath, M. McKague, and D. Stebila, “Associative Blockchain for Decentralized PKI Transparency,” Cryptography, vol. 5, no. 2, p. 14, May 2021.

DPKIT

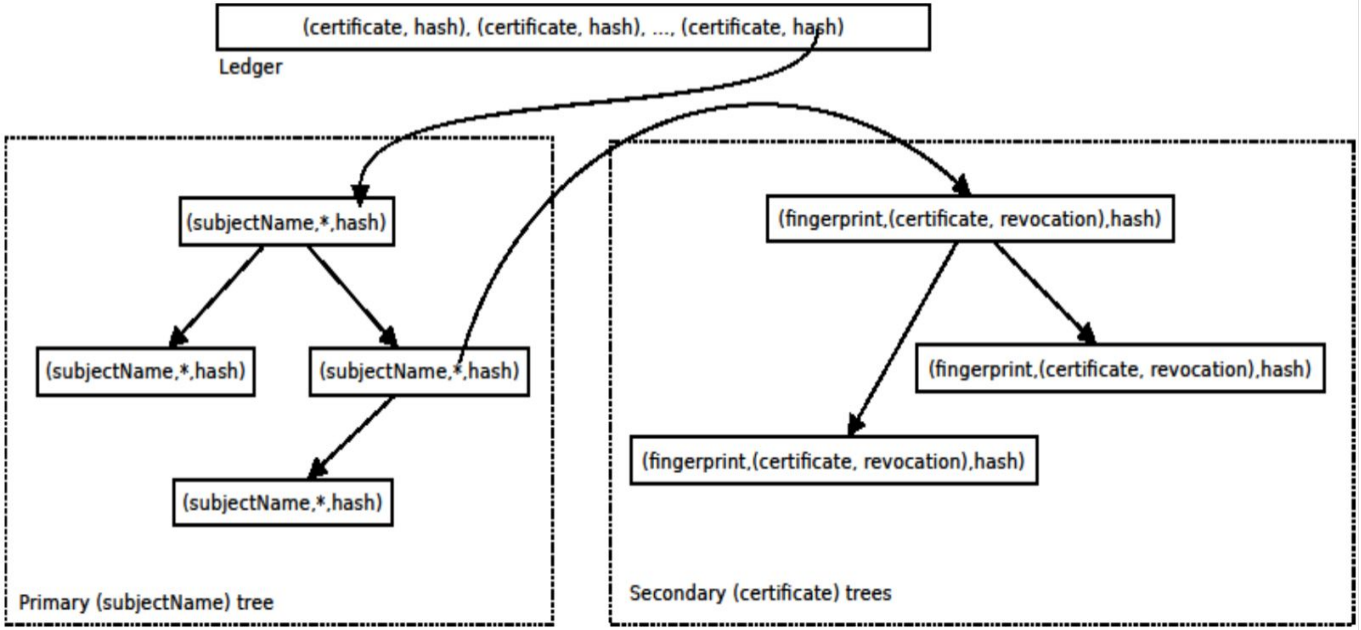


Certificate existence/validity is demonstrated through membership proofs.

1) the certificate tree in the primary BST
2) the one that identifies the certificate inside the secondary tree.

Source: [BHMS21] X. Boyen, U. Herath, M. McKague, and D. Stebila, “Associative Blockchain for Decentralized PKI Transparency,” Cryptography, vol. 5, no. 2, p. 14, May 2021.

DPKIT

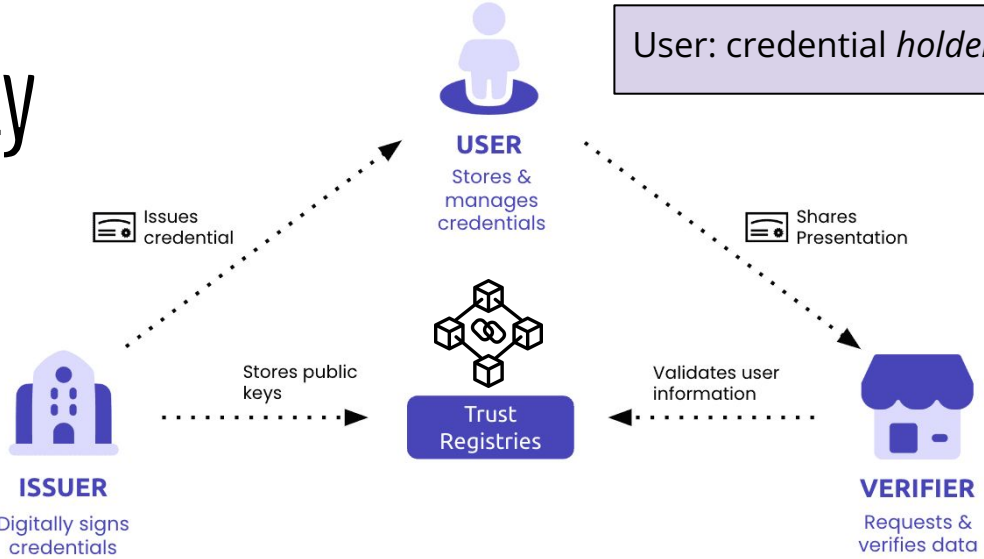


Proof of revocation is a membership proof in the secondary tree.

Source: [BHMS21] X. Boyen, U. Herath, M. McKague, and D. Stebila, “Associative Blockchain for Decentralized PKI Transparency,” Cryptography, vol. 5, no. 2, p. 14, May 2021.

Self-Sovereign Identity

Blockchain is used to record metadata, public keys, and all information needed for verification.



Decentralized IDs: recognize individuals and organizations that participate to the network.

Sources: Flaticon.com

Self-Sovereign Identity

In an SSI system, there are several operations that involve credentials.

1) **Issuance**: an issuer creates and signs a credential for a holder.

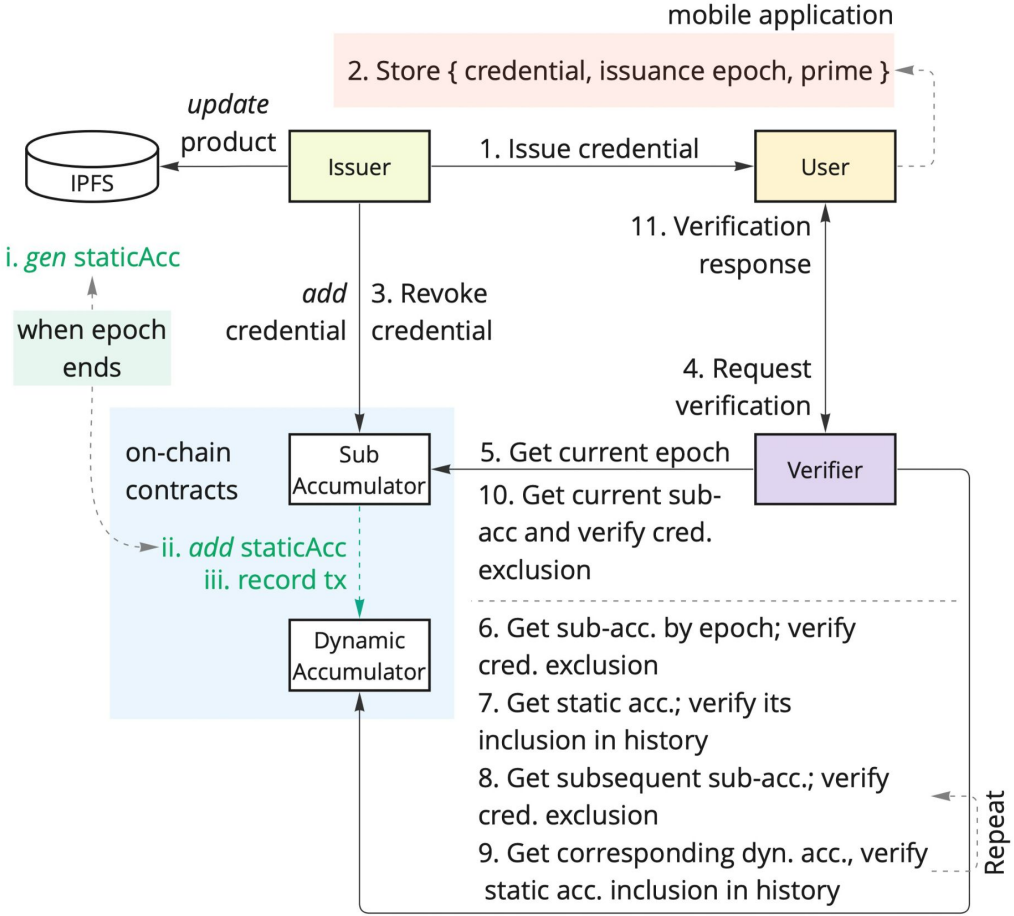
2) **Revocation**: an issuer invalidates a previously issued credential. This can happen for several reasons (e.g., user frauds, credential expiration, issuance by mistake, ensuring the currency of credential data).

3) **Verification**: a verifier checks whether a presented credential is authentic and valid.

CredChain

An SSI framework aimed at improving the efficiency of revocation.

Both revocation and verification involve on-chain interactions via smart contracts.



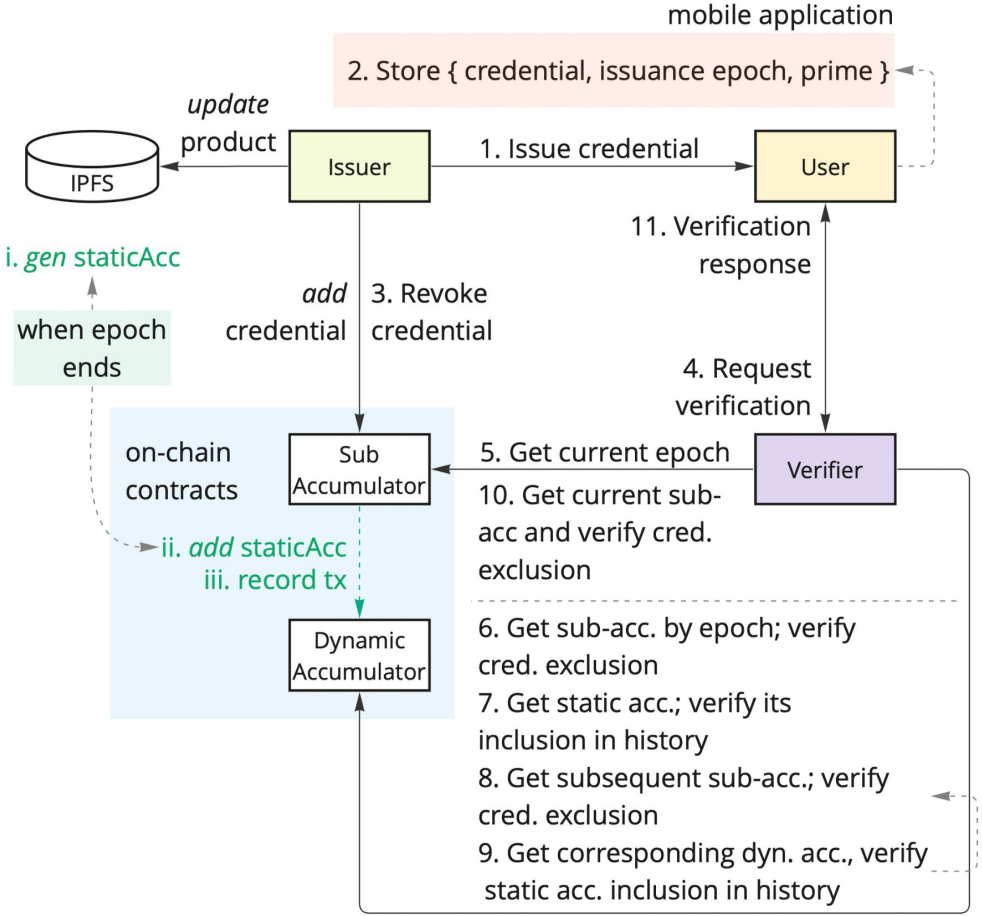
Source: [SMP23] D. Schumm, R. Mukta, and H. Paik, “Efficient Credential Revocation Using Cryptographic Accumulators,” 2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). IEEE, pp. 127–134, Jul. 2023.

CredChain

Relies on various types of set accumulators.

- **Bloom filters** (probabilistic "sub-accumulator")
- **RSA-based** ("static" and "dynamic" accumulators).

Epoch: period of time until the sub-accumulator reaches its capacity.



Source: [SMP23] D. Schumm, R. Mukta, and H. Paik, "Efficient Credential Revocation Using Cryptographic Accumulators," 2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). IEEE, pp. 127–134, Jul. 2023.

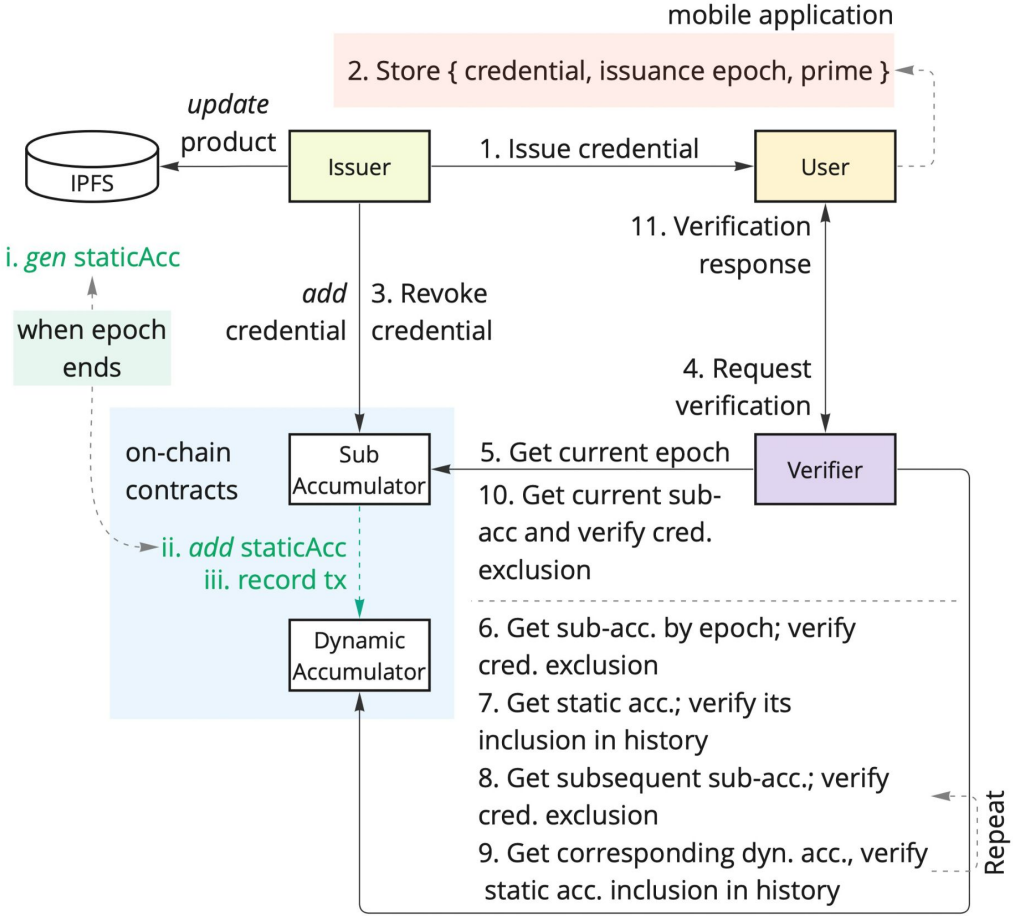
CredChain

Revoked credentials are added to the sub-accumulator.

To verify credentials:

1) Check their exclusion from the sub/static accumulators.

2) Check inclusion of static accumulator in revocation history.



Source: [SMP23] D. Schumm, R. Mukta, and H. Paik, “Efficient Credential Revocation Using Cryptographic Accumulators,” 2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). IEEE, pp. 127–134, Jul. 2023.

Hyperledger Indy

Open-source permissioned DLT for managing self-sovereign identities.

The ledger records credential **schemas** and **definitions**.

- Schema: specifies credential name, version and attributes.
- Definition: signals issuer's intention to create credentials with a given schema (and defines keys that used to sign such credentials).

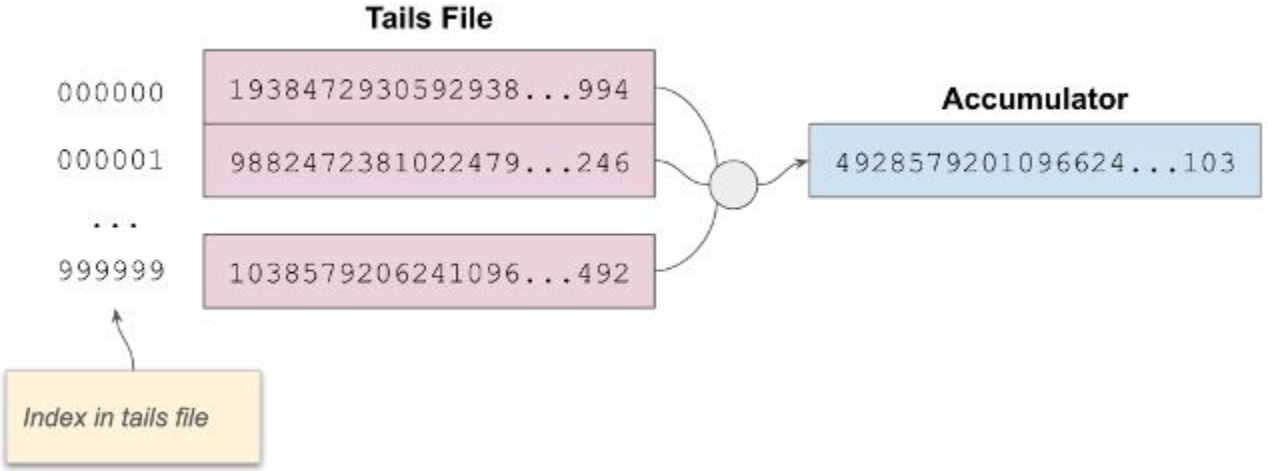
Source: https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/concepts/revocation/cred-revocation.html

Hyperledger Indy

In Hyperledger Indy, **credential revocation** is possible only if issuers publish a **revocation registry** on the ledger (i.e., metadata referencing a definition and specifying how to handle revocation).

Source: https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/concepts/revocation/cred-revocation.html

Hyperledger Indy

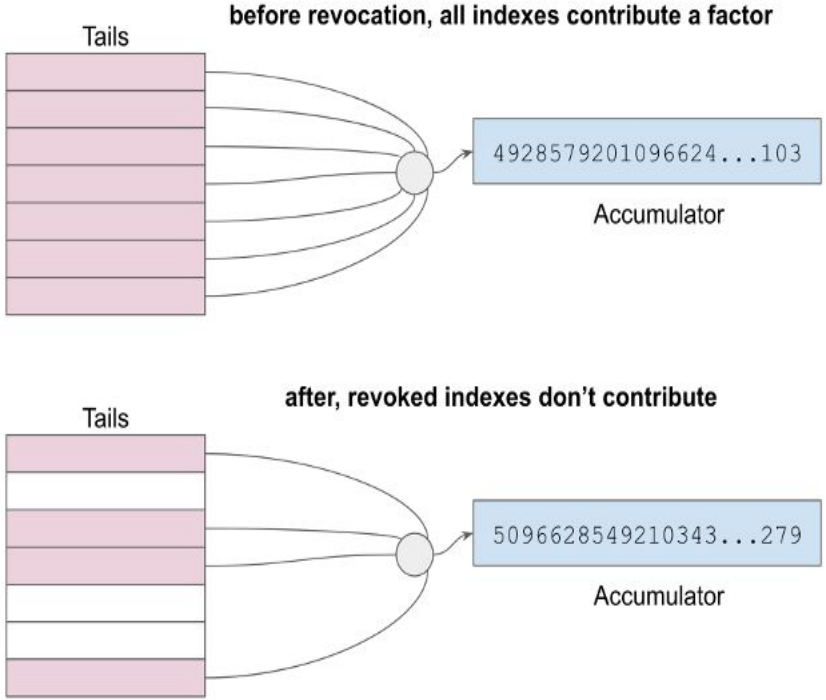


Public tails file: entries represent all possible credentials that could be issued.

Only valid (i.e., issued and not revoked) credentials contribute to the accumulator value.

Source: https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/concepts/revocation/cred-revocation.html

Hyperledger Indy



Source: https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/concepts/revocation/cred-revocation.html

Hyperledger Indy

When a credential is issued, the issuer transmits the following information to the holder (who will later become a prover):

1. actual **credential file**;
2. **private factor:** the index corresponding to the credential in the tails file;
3. **witness**: product of all other tails file entries.

Source: https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/concepts/revocation/cred-revocation.html

Hyperledger Indy

Primary proof. Proving credential properties:
- “What is your birthdate?”
- “Please disclose your address”

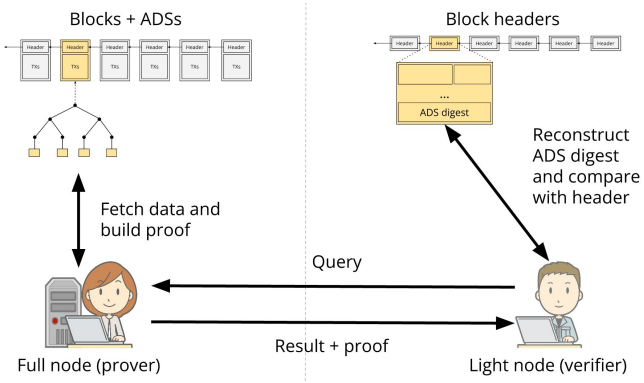
Proof of non-revocation. The prover must demonstrate that the credentials behind the primary proof have not been revoked.
=> Derive accumulator value from private factor and witness (just one multiplication).

Witness delta: mechanism for updating provers’ witnesses after accumulator has changed.

Source: https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/concepts/revocation/cred-revocation.html

Discussion

Query authentication



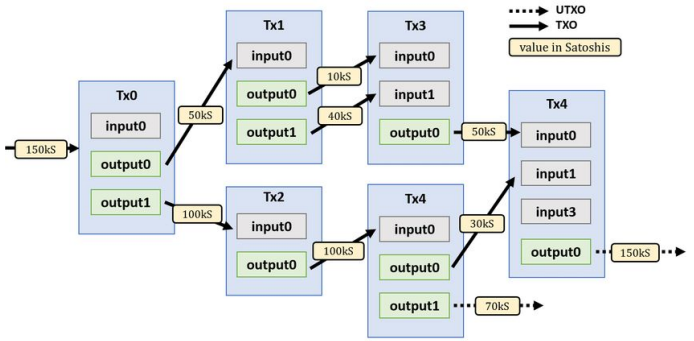
Pros:

- Enable authenticated queries.
- Reduce verification object size.
- Compress information included in multiple blocks.

Cons:

- Block structure modification required.
- A third party for trusted setup phase might be required.
- Arithmetic-based solutions might be slower than hashing operations.

Stateless transaction validation



Pros:

- Mitigate the impact of UTXO set growth (especially for resource-constrained nodes).

Cons:

- Increase network communication overhead (due to membership proofs attached to TX payloads).

  => Proof aggregation (i.e., batching) suggested as possible solution.

Anonymity enhancement

Pros:

- Decouple coin minting operations from spending transactions.
- Compress set of unspent coins (as in stateless TX validation).
- ...

Cons:

- Increasing burden on miners (because of proof validation).
- ...

Identity management

Pros:

- Reduce ledger size (in PKIs).
- Improve certificate search and validation (in PKIs).
- Improve credential revocation efficiency (in SSI).
- ...

Cons:

- Presence of a trusted party may not be desirable to achieve full decentralization (but may be fine in permissioned systems).
- ...

Set accumulators

- Compress large sets into a single digest, while ensuring correctness of operations performed on those sets.

- Perfect for scenarios where:
  - data are continuously appended to the blockchain;
  - the need for data retrieval collides with the difficulty of reading sequentially from the data structure.

- Common goals:
  - improving overall system scalability;
  - making the network more accessible to resource-constrained nodes.
  - ...

Table 6
A summary of the presented applications of set accumulators to blockchain systems. For each proposal, we indicate the use case, the accumulator type (specifying whether it is primarily based on cryptographic hash functions) and provide a brief description.

Use case	Proposal	Ref.	Accumulator type	Hash-based	Description
Query authentication	Bitcoin SPV	[1]	MHT	Yes	Transaction verification
	vChain	[45]	Bilinear	No	Boolean range query verification
	GCA^2-tree	[46]	ESA	No	Aggregate range query verification
	GEM^2-tree	[47]	Merkle B-tree	Yes	Range query verification
	Loporchio et al.	[48]	Merkle R-tree	Yes	Range query verification
	FalconDB	[49]	Bilinear	No	Collaborative database
Stateless transactions validation	Boneh et al.	[55]	RSA-based	No	UTXO and account-based validation
	EDRAX	[56]	Sparse MHT [41]	Yes	UTXO and account-based validation
	MiniChain	[57]	MMR, RSA-based	Yes	UTXO-based validation
	Bailey, Sankagiri	[58]	MHT	Yes	UTXO-based validation
	Utreexo	[59]	MHT forest	Yes	UTXO-based validation
	SecurePrune	[60]	RSA-based	No	Block pruning scheme
Anonymity enhancement	Zerocoin	[65]	RSA-based	No	Anonymity protocol
	Zerocash	[66]	MHT	Yes	Anonymity protocol
	Garman et al.	[67]	MHT	Yes	Anonymous payment scheme
	MiniLedger	[68]	RSA-based, MHT	Yes	Anonymous payment scheme
Identity management	Certcoin	[74]	Dynamic MHT [77]	Yes	Decentralized PKI
	DPKIT	[78]	Merkle BST	No	Decentralized PKI
	Feng et al.	[79]	RSA-based [80]	No	Decentralized PKI
	Hyperledger Indy	[81]	Bilinear-based [82]	No	Self-sovereign identity management

Source: [LBDR23] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa and L. Ricci. "A survey of set accumulators for blockchain systems." Computer Science Review 49 (2023): 100570.

Thank you
for your attention

Image credits: Freepik.com

References

[BCC+14] E. Ben Sasson et al., “Zerocash: Decentralized Anonymous Payments from Bitcoin,” 2014 IEEE Symposium on Security and Privacy. IEEE, May 2014.

[BP97] N. Barić and B. Pfitzmann, “Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees,” Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 480–494, 1997.

[BHMS21] X. Boyen, U. Herath, M. McKague, and D. Stebila, “Associative Blockchain for Decentralized PKI Transparency,” Cryptography, vol. 5, no. 2, p. 14, May 2021.

[CB21] P. Chatzigiannis and F. Baldimtsi, “MiniLedger: Compact-Sized Anonymous and Auditable Distributed Payments,” Lecture Notes in Computer Science. Springer International Publishing, pp. 407–429, 2021.

[CW20] H. Chen, Y. Wang, “MiniChain: A lightweight protocol to combat the UTXO growth in public blockchain”, J. Parallel Distrib. Comput. 143 (2020) 67–76.

[FVY14] C. Fromknecht, D. Velicanu, and S. Yakoubov. "A decentralized public key infrastructure with identity retention." Cryptology ePrint Archive. 2014.

[LBDR21] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa and L. Ricci. “Authenticating spatial queries on blockchain systems.” IEEE Access 9 (2021): 163363-163378.

[LBDR23] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa, and L. Ricci. "A survey of set accumulators for blockchain systems." Computer Science Review 49 (2023): 100570.

References

[LBDR25] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa, and L. Ricci, “Skip index: Supporting efficient inter-block queries and query authentication on the blockchain,” Future Generation Computer Systems, vol. 164, p. 107556, Mar. 2025.

[MGGR13] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoin: Anonymous Distributed E-Cash from Bitcoin,” 2013 IEEE Symposium on Security and Privacy. IEEE, pp. 397–411, May 2013.

[Nak08] S. Nakamoto. “Bitcoin: A peer-to-peer electronic cash system.” 2008.

[Ngu05] L. Nguyen, “Accumulators from Bilinear Pairings and Applications,” Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 275–292, 2005.

[SMP23] D. Schumm, R. Mukta, and H. Paik, “Efficient Credential Revocation Using Cryptographic Accumulators,” 2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). IEEE, pp. 127–134, Jul. 2023.

[XZX19] C. Xu, C. Zhang, and J. Xu, “vChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases,” Proceedings of the 2019 International Conference on Management of Data. ACM, Jun. 25, 2019.

[ZKP17] Y. Zhang, J. Katz, and C. Papamanthou, “An Expressive (Zero-Knowledge) Set Accumulator,” 2017 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, pp. 158–173, Apr. 2017.