NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Bitcoin

Construction of the Bitcoin Blockchain

Dr Sourav SEN GUPTA

Lecturer, SCSE, NTU

Cryptographic Components

Hash Functions

Motivation for Hashing

Several concepts of Hash in Computing, for related but different purposes

- Encoding of data into small, fixed size (used in standard Hash Tables)
- Encoding of data to authenticate message integrity (Cryptographic Hash)
- Metadata tag for users to apply dynamic, user-generated labels (#hashtag)
- Deriving multiple secret keys from master key or password (Password Hash)
- Remember or report utility locations in UNIX operating system (Hash Table)

In Blockchain, we will only care about Cryptographic Hash Functions.

Cryptographic Hash Functions

Function from arbitrary Domain to fixed Range (arbitrary sized message to fixed length digest)

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

- Efficiently computable : Expected complexity O(m) for an m-bit input
- Preimage resistance : Finding input x given $y = H(x)$ is not possible
- 2nd Preimage resistance : Given x, finding $y \neq x$ with $H(y) = H(x)$ is impossible
- Collision resistance : Finding any pair $x \neq y$ with $H(x) = H(y)$ is impossible

Hash functions in Cryptography : Keyed (MAC) or Keyless (collision resistant)

In Blockchain, we are interested in keyless collision resistant Hash Functions

Food for thought …

Function from arbitrary Domain to fixed Range
(arbitrary sized message to fixed length digest)

$H : \{0,1\}^* \rightarrow \{0,1\}^n$

- Preimage resistance : Finding input x given $y = H(x)$ is not possible
  Analogy : Given some birthday, would I be able to determine whose it is?

Function H is public, open to compute. How can it be Preimage Resistant?

What if I pre-compute $H(x)$ for all input values x, and then just compare y?

Food for thought ...

Function from arbitrary Domain to fixed Range
(arbitrary sized message to fixed length digest)

$H : \{0,1\}^* \rightarrow \{0,1\}^n$

- 2^{nd} Preimage resistance : Given x, finding $y \neq x$ with $H(y) = H(x)$ is impossible
  Analogy : Given your birthday, find another person with the same birthday.

Domain is larger than Range. How can it be 2^{nd} Preimage Resistant?

There may be another input in Domain for which the output matches.

Food for thought …

Function from arbitrary Domain to fixed Range
(arbitrary sized message to fixed length digest)

$H : \{0,1\}^* \rightarrow \{0,1\}^n$

- Collision resistance : Finding any pair $x \neq y$ with $H(x) = H(y)$ is impossible
  Analogy : Finding any two persons with the exact same birthday.

Domain is larger than Range. How can it be Collision Resistant?

There must be two inputs in Domain for which the outputs match.

Hash Function Standards

You need some function to "compress" the input
You also need the function to "iterate" arbitrarily

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

- MD5 : Started the journey back in 1993 (still used sometime!)
- SHA1 : Standardized by NIST in 1993-95 (collision found, 2017)
- SHA2 : Standardized by NIST in 2001 (224, 256, 384, 512 bits)
- SHA3 : Standardized by NIST in 2015 (224, 256, 384, 512 bits)

While Bitcoin uses SHA-256 from SHA2 family, Ethereum uses keccak256.

Application of Hash Functions [1]

Efficiently computable : $O(m)$ for an m-bit input string

Preimage resistance : $y = H(x)$ does not reveal input x

Collision resistance : $x \neq y$ with $H(x) = H(y)$ is impossible

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

Commitment Scheme : You commit to something and anyone can verify!

Commitment	**commitment = H(** data, random nonce **)**	→	Publish **commitment**
Verification	**commitment ?= H(** data, random nonce **)**	→	Output **True** or **False**

Is it Hiding? If you use Hash function, given commitment, is it possible to find data?

Is it Binding? If you use Hash function, is it possible to deny your commitment later?

[1] reading: Chapter 1.1 of the book “Bitcoin and Cryptocurrency Technologies”

Cryptographic Components

Digital Signatures

Motivation for Signatures

Digital Signature acts as a Proof-of-Identity
(almost like your physical signature, but …)

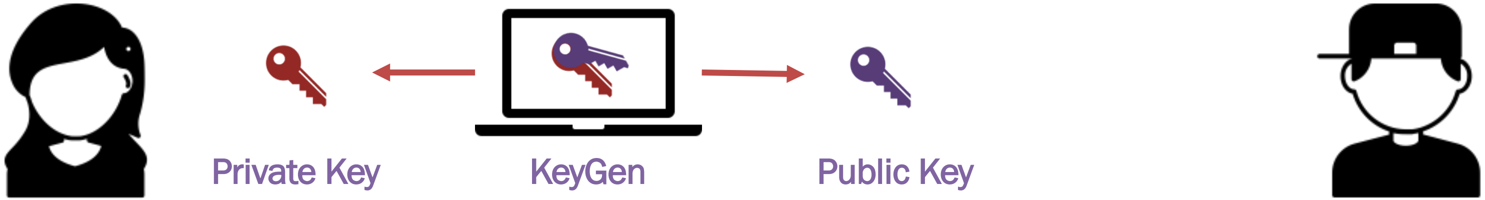
- Digital Signature needs to verify your identity and action on digital data
- Digital Signature, when copied for a different data, should not be valid
- Signature by a specific person on a specific message should be unique
- Once you sign a message/data, you should not be able to repudiate it

In Blockchain, we care about Digital Signature in terms of Ownership.

Digital Signature

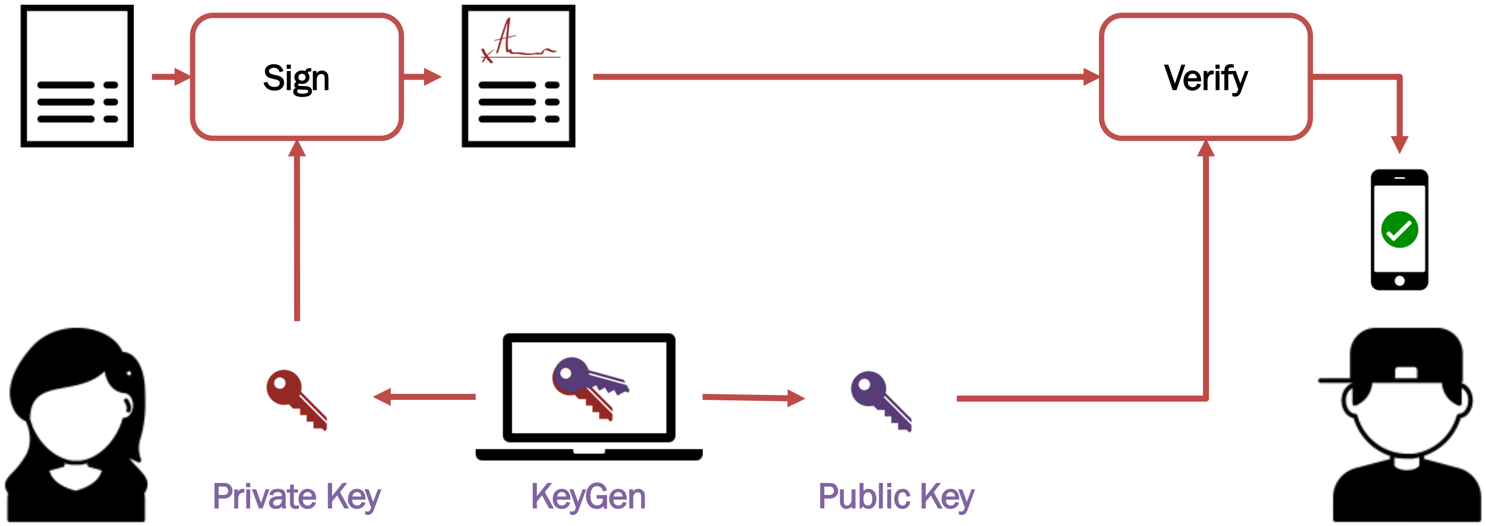
Comprises of two stages : **Sign and Verify**

- Sign : Must be unique to the person signing. Must relate to a "secret key".
- Verify : Must be accessible to anyone in public. Must relate to a "public key".
- We need two keys for the scheme. Natural idea is Public Key Cryptography.



Digital Signature (almost)

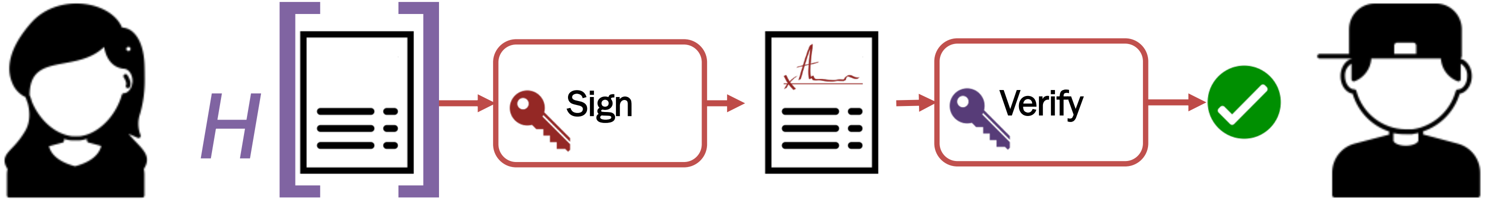
Comprises of two stages : **Sign and Verify**



Digital Signature

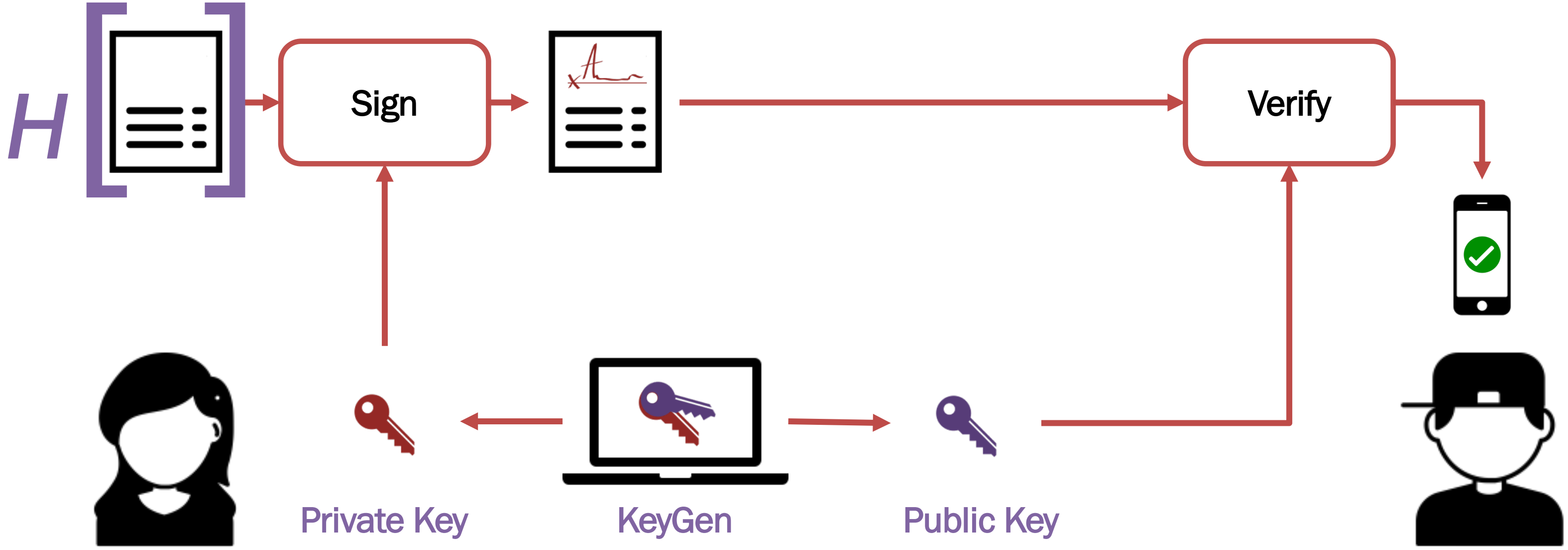
Promises three Security Properties

- **Entity Authentication :** Receiver should be able to verify Source of message
- **Non-Repudiation :** Sender of the message with a signature can't Deny later
- **Message Integrity :** Message should not be Tampered during transmission



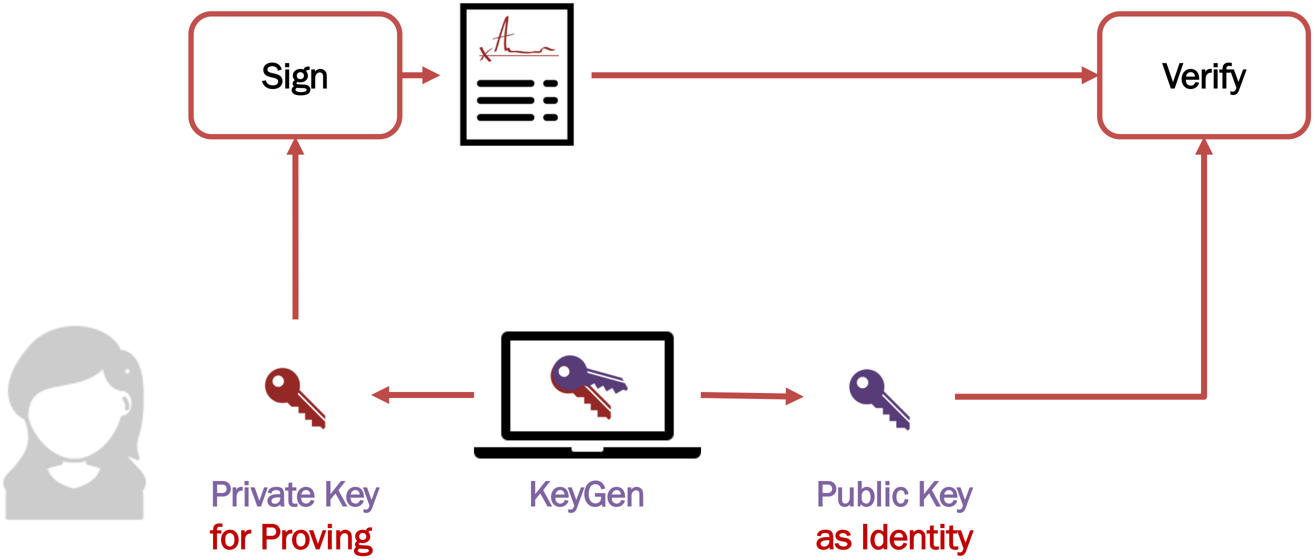
Digital Signature

Comprises of two stages : **Sign and Verify**



Identity and Authentication

Comprises of two stages : **Sign and Verify**



Digital Signature Standards

FIPS 186-4 Standards

Digital Signature Algorithm (DSA)

RSA Signature Algorithm

Elliptic Curve DSA (ECDSA)

Most standard Blockchains use ECDSA.

Some use sophisticated Ring Signature.

FIPS PUB 186-4

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

Digital Signature Standard (DSS)

CATEGORY: COMPUTER SECURITY **SUBCATEGORY: CRYPTOGRAPHY**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900
http://dx.doi.org/10.6028/NIST.FIPS.186-4
Issued July 2013

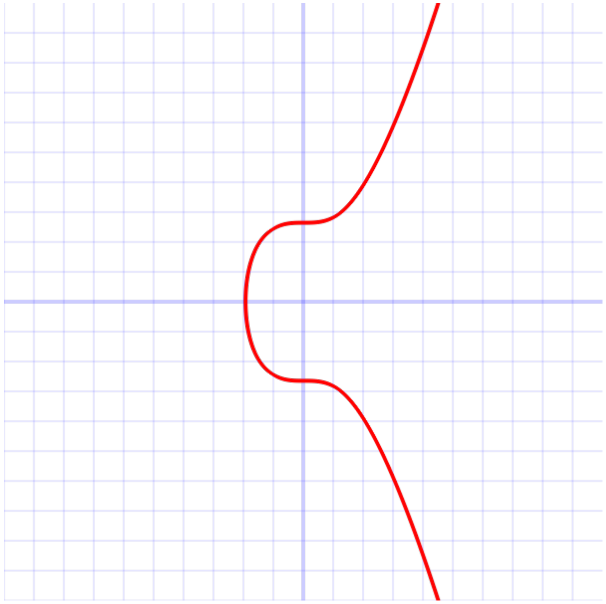
ECDSA Digital Signature [2]

Elliptic Curve is the underlying Algebraic Structure

Bitcoin uses a specific set of ECDSA parameters

- Standard NIST elliptic curve **secp256k1**
- 256-bit Private Key, 512-bit Public Key
- 256-bit Message, 512-bit Signature

SHA256 perfectly suits the input specification.



Curve sec256k1 over Reals : $y^2 = x^3 + 7$

[2] reading: Chapter 1.3 of the book "Bitcoin and Cryptocurrency Technologies"

Data Structures

Linked Lists : Hash Chain

Recall : Cryptographic Hash Functions

Efficiently computable : O(m) for an m-bit input string

Preimage resistance : $y = H(x)$ does not reveal input x

Collision resistance : $x \neq y$ with $H(x) = H(y)$ is impossible

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

Commitment Scheme : You commit to something and anyone can verify!

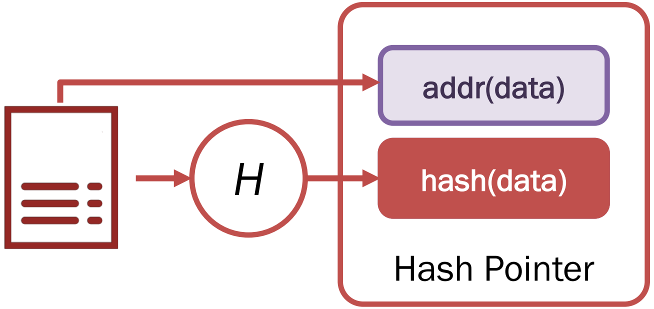
Commitment	**commitment = H(** data, random nonce **)**	→	Publish **commitment**
Verification	**commitment ?= H(** data, random nonce **)**	→	Output **True** or **False**

If you store the hash value of some data, you will know if it was modified or tampered.

Pointer : Hash Pointer

Store the hash digest of data along with the address pointer to the data.

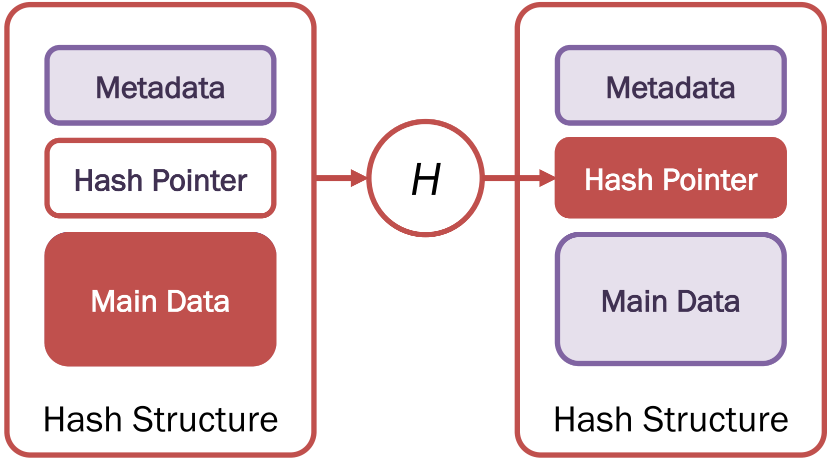
Modifications (tampering) on the data will be evident to the hash pointer.



Node : Hash Structure

Store the hash digest of one structure as an integral part of another structure.

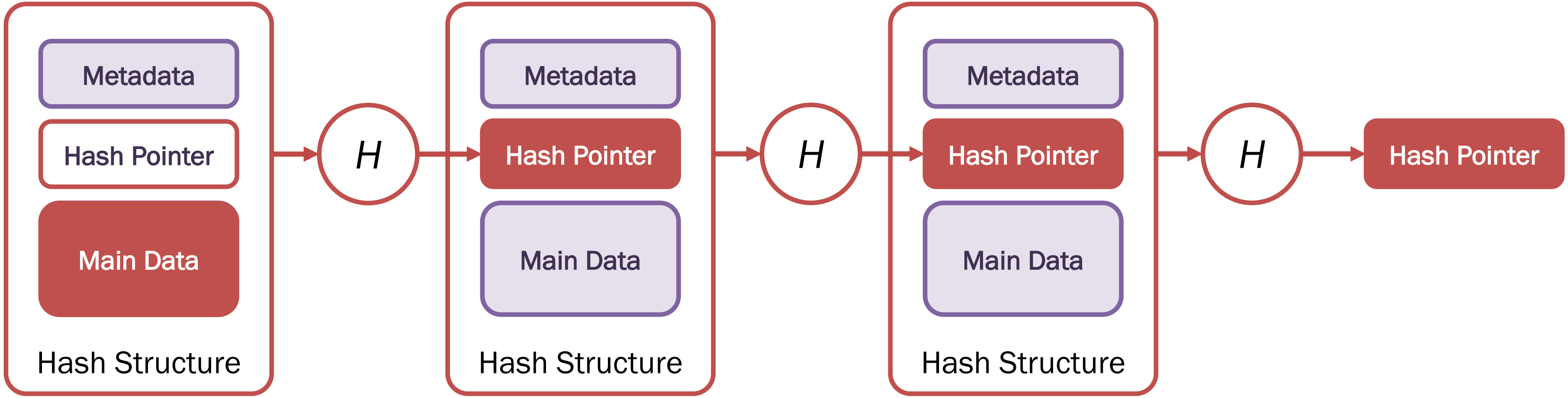
Modifications (tampering) on the first structure will be evident to the second.



Linked List : Hash Chain [3]

Store the hash digest of the previous node as an integral part of the next node.

Modifications (tampering) on any node will be evident to the chain thereafter.

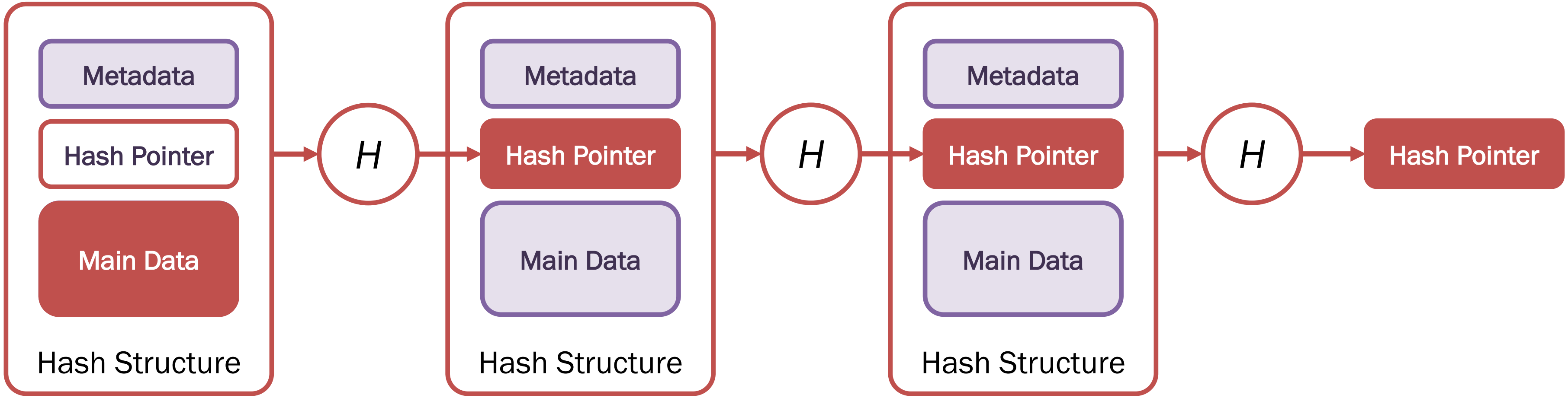


[3] reading: Chapter 1.2 of the book "Bitcoin and Cryptocurrency Technologies"

Linked List : Hash Chain

Computational complexity of Updating an existing Node : O(n)

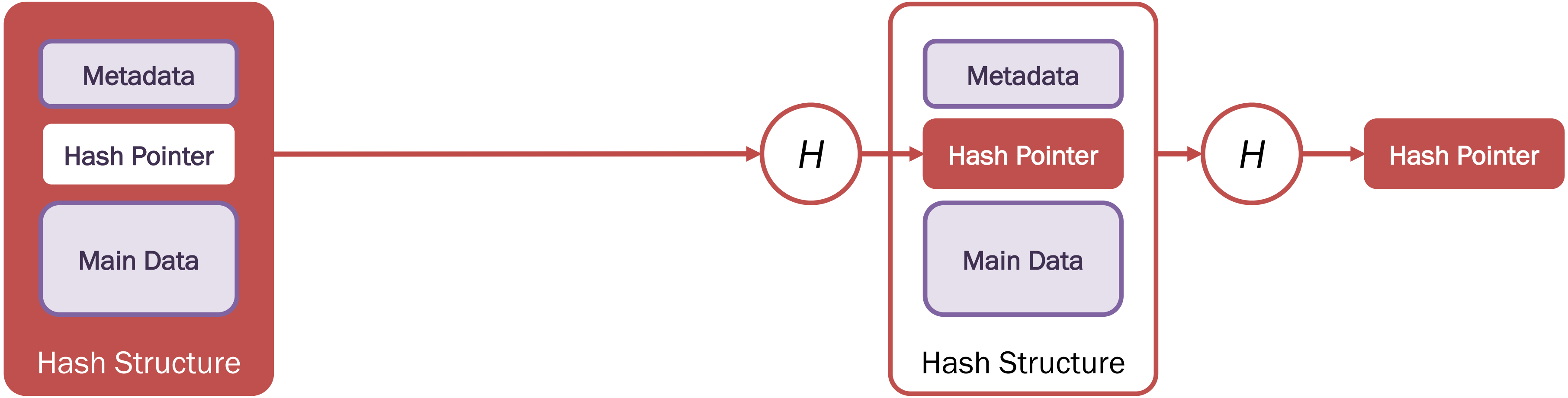
where n is the number of nodes in the chain after the node.



Linked List : Hash Chain

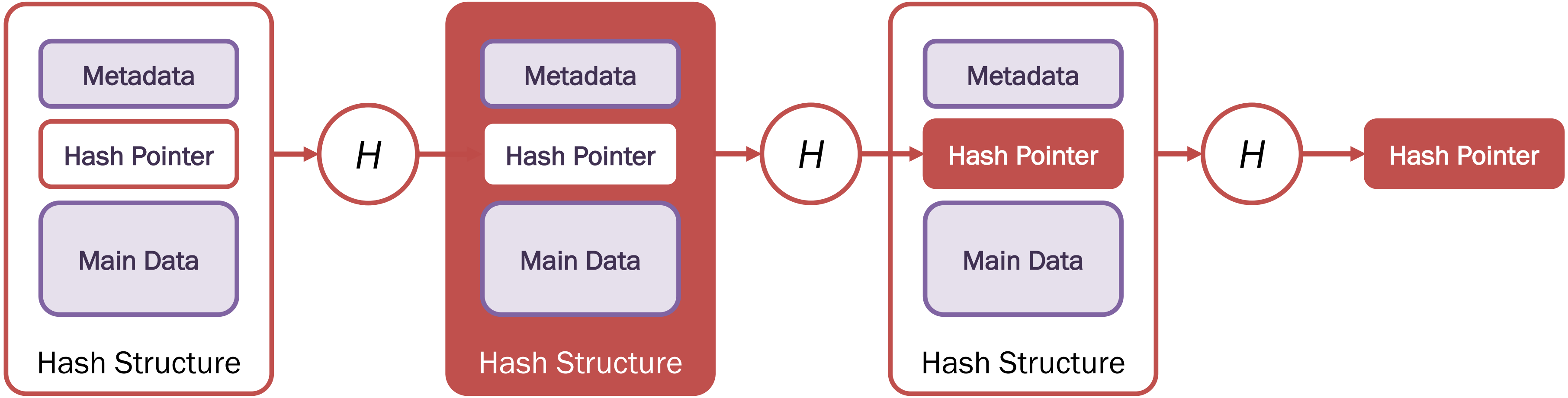
Computational complexity of Deleting an existing Node : O(n)

where n is the number of nodes in the chain after the node.



Linked List : Hash Chain

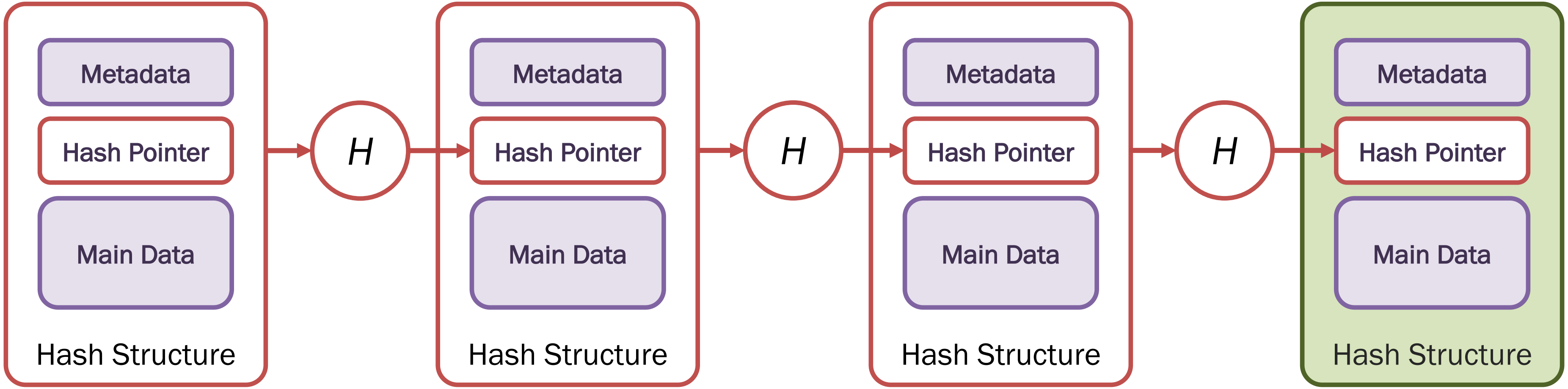
Computational complexity of Inserting a new Node : O(n)
where n is the number of nodes in the chain after the node.



Linked List : Hash Chain

Computational complexity of Appending a new Node : O(1)

This is independent of the number of nodes *n* in the chain.



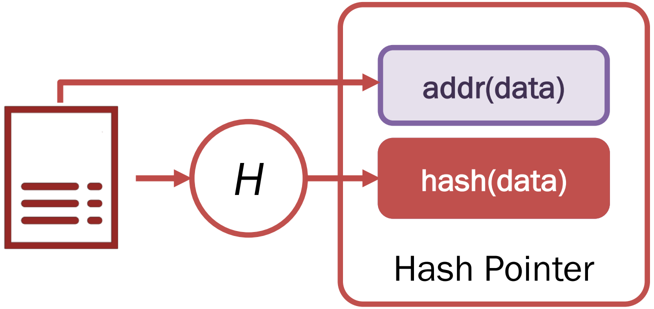
Data Structures

Binary Trees : Merkle Tree

Recall : Hash Pointer

Store the hash digest of data along with the address pointer to the data.

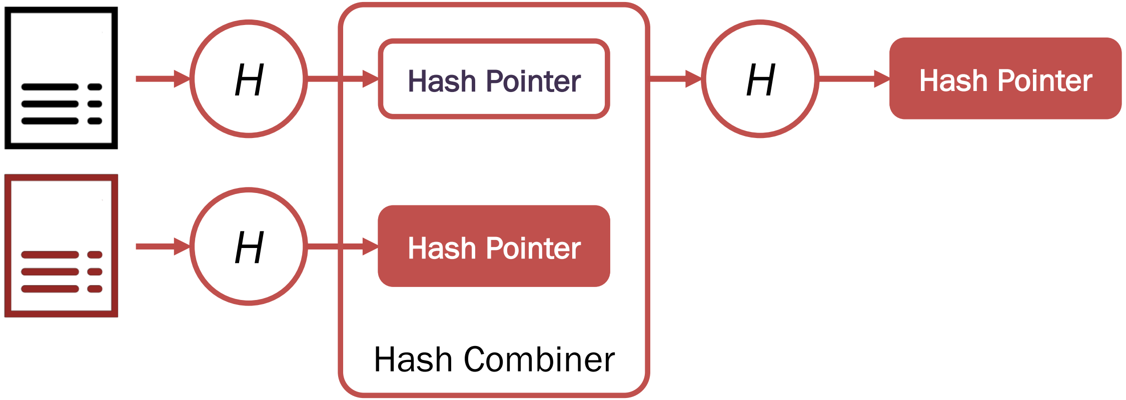
Modifications (tampering) on the data will be evident to the hash pointer.



Compression : Hash Combiner

Combine hash pointers of two data into a single structure, with a hash pointer.

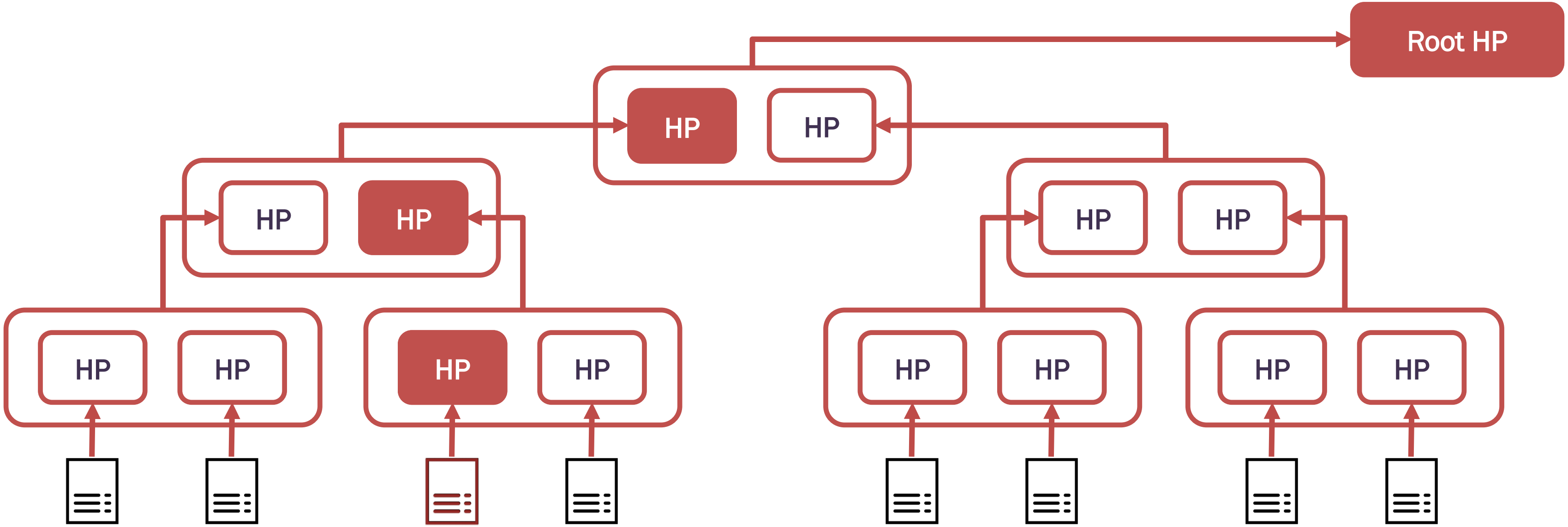
Modifications (tampering) on any data will be evident to the final hash pointer.



Binary Tree : Merkle Tree [3]

Combine hash pointers of all data into a single structure, in the form of a tree.

Modifications (tampering) on any data will be evident to the root hash pointer.

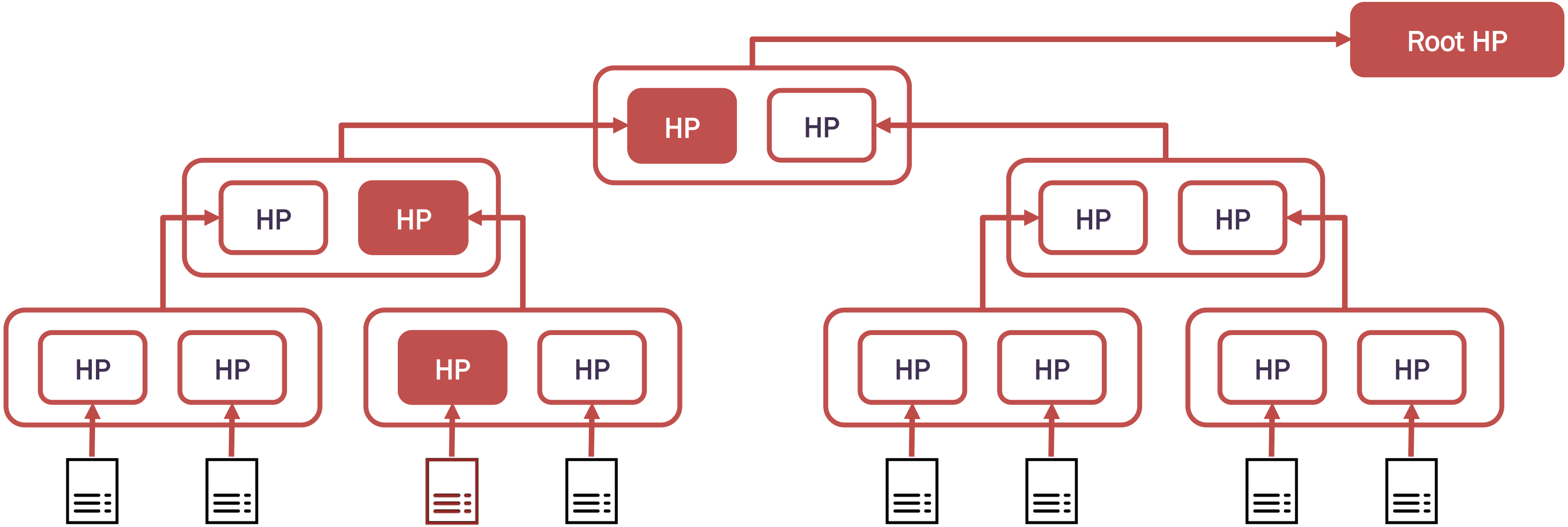


[3] reading: Chapter 1.2 of the book “Bitcoin and Cryptocurrency Technologies”

Binary Tree : Merkle Tree

Computational complexity of Updating an existing Node : O(log *n*)

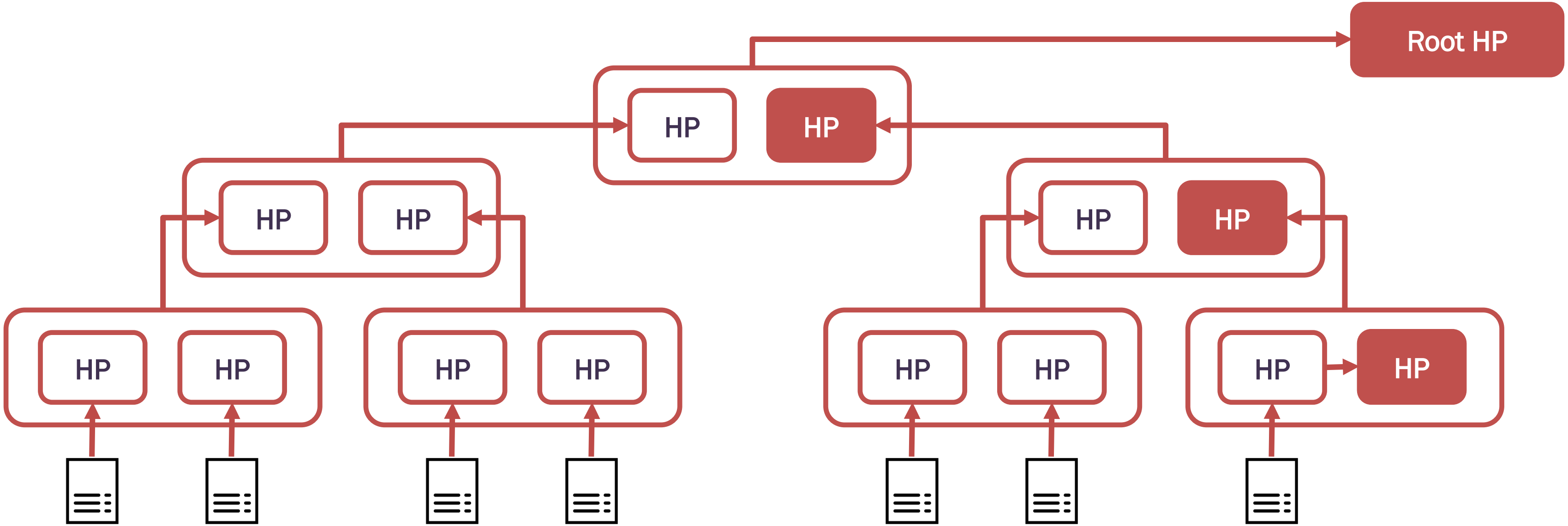
where *n* is the number of data nodes in the leaf level of the tree.



Binary Tree : Merkle Tree

Computational complexity of Deleting an existing Node : O(log *n*)

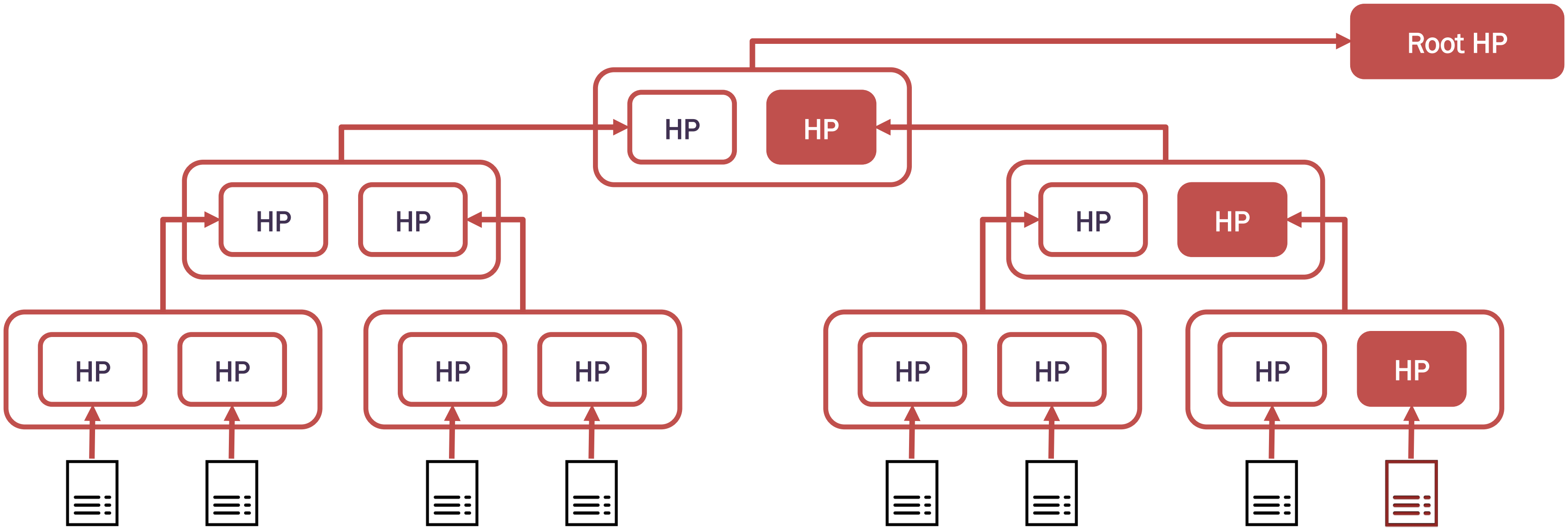
where *n* is the number of data nodes in the leaf level of the tree.



Binary Tree : Merkle Tree

Computational complexity of Inserting a new Node : O(log *n*)

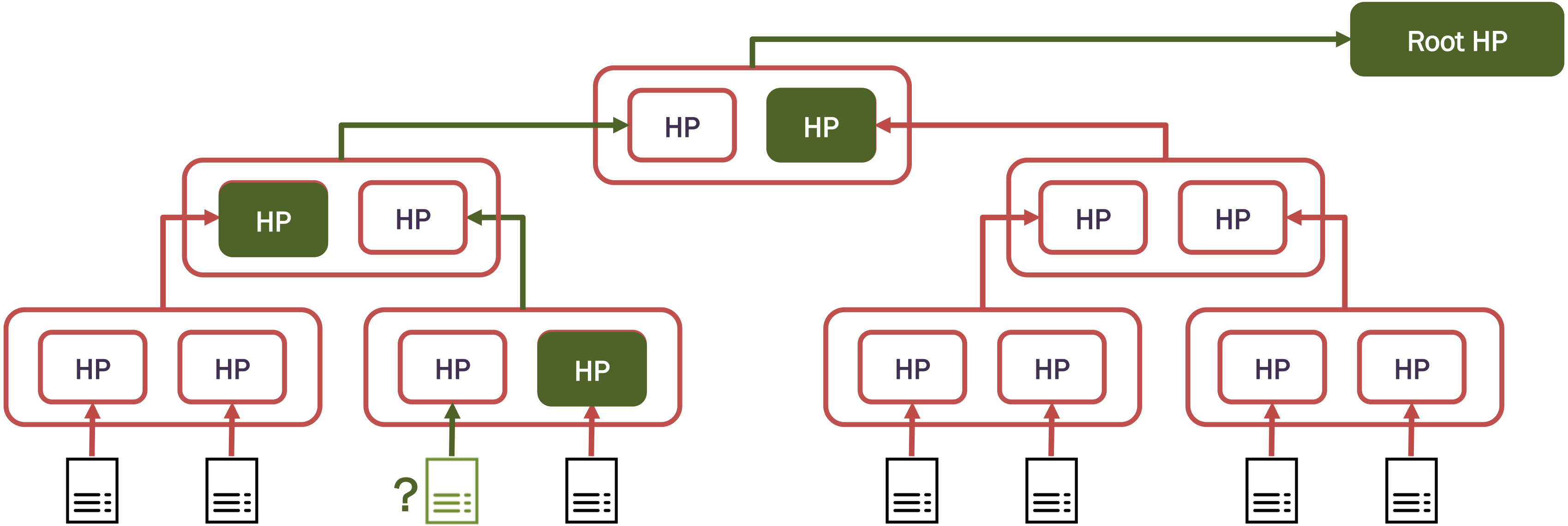
where *n* is the number of data nodes in the leaf level of the tree.



Binary Tree : Merkle Tree

Complexity of Proof of Membership : O(log *n*) HPs and Checks

where *n* is the number of data nodes in the leaf level of the tree.



Ledger of Records

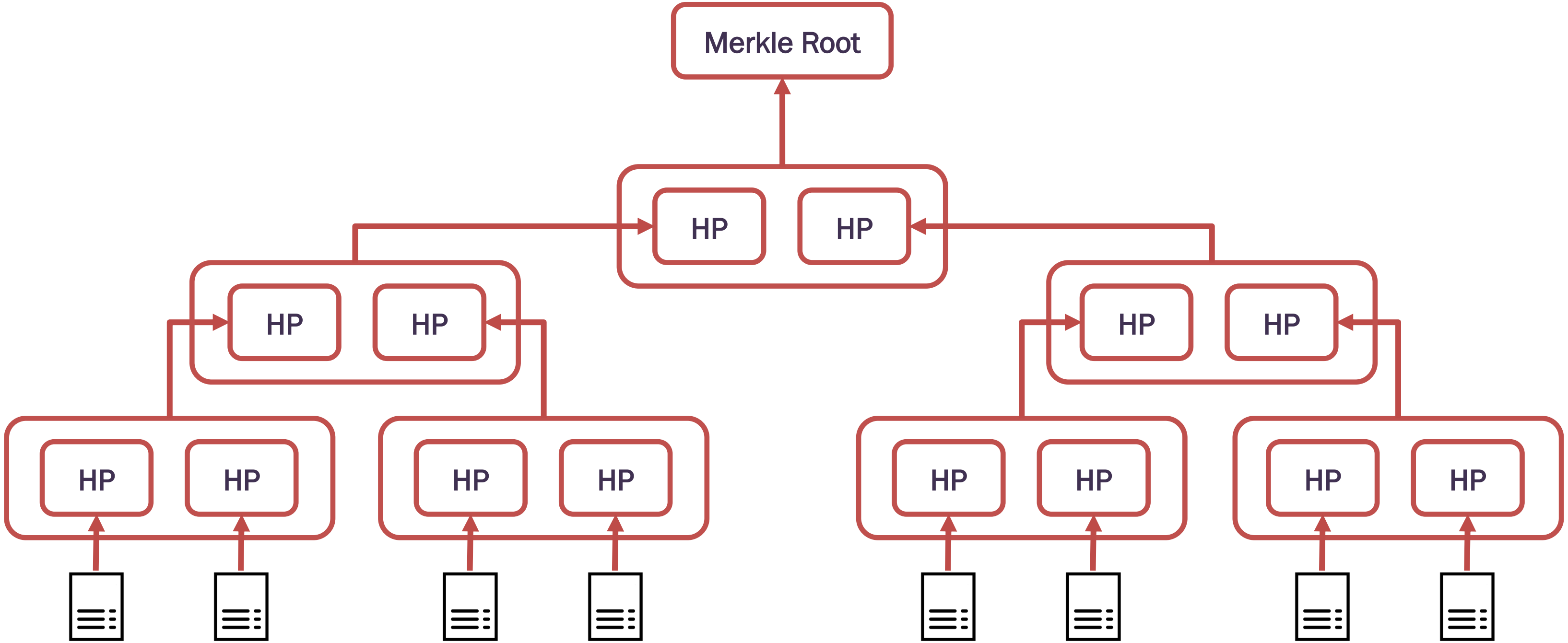
Interconnecting Blocks and Records

Individual Records

These can be individual records or documents that are considered atomic.

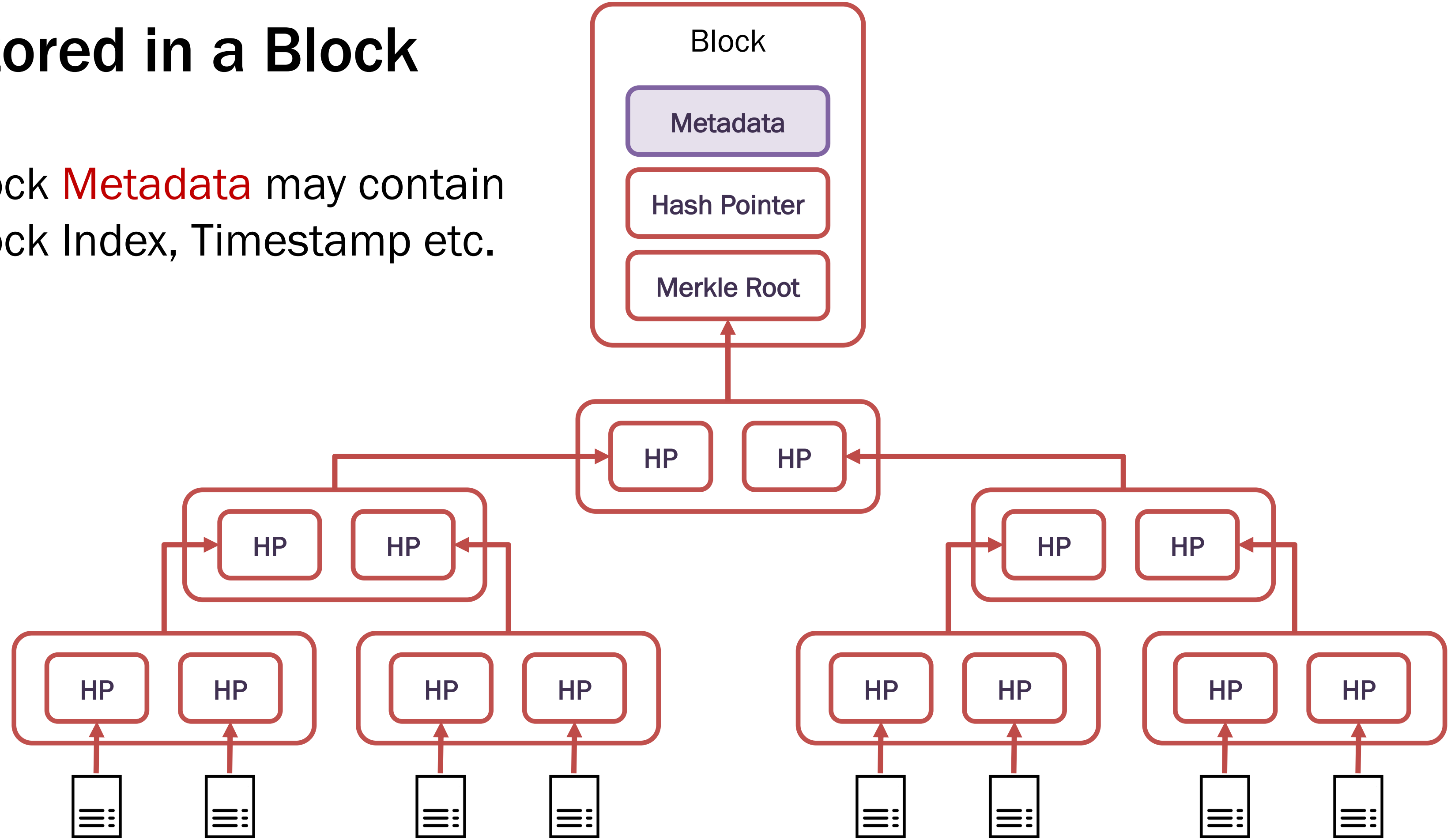
Accumulated into a Merkle Tree

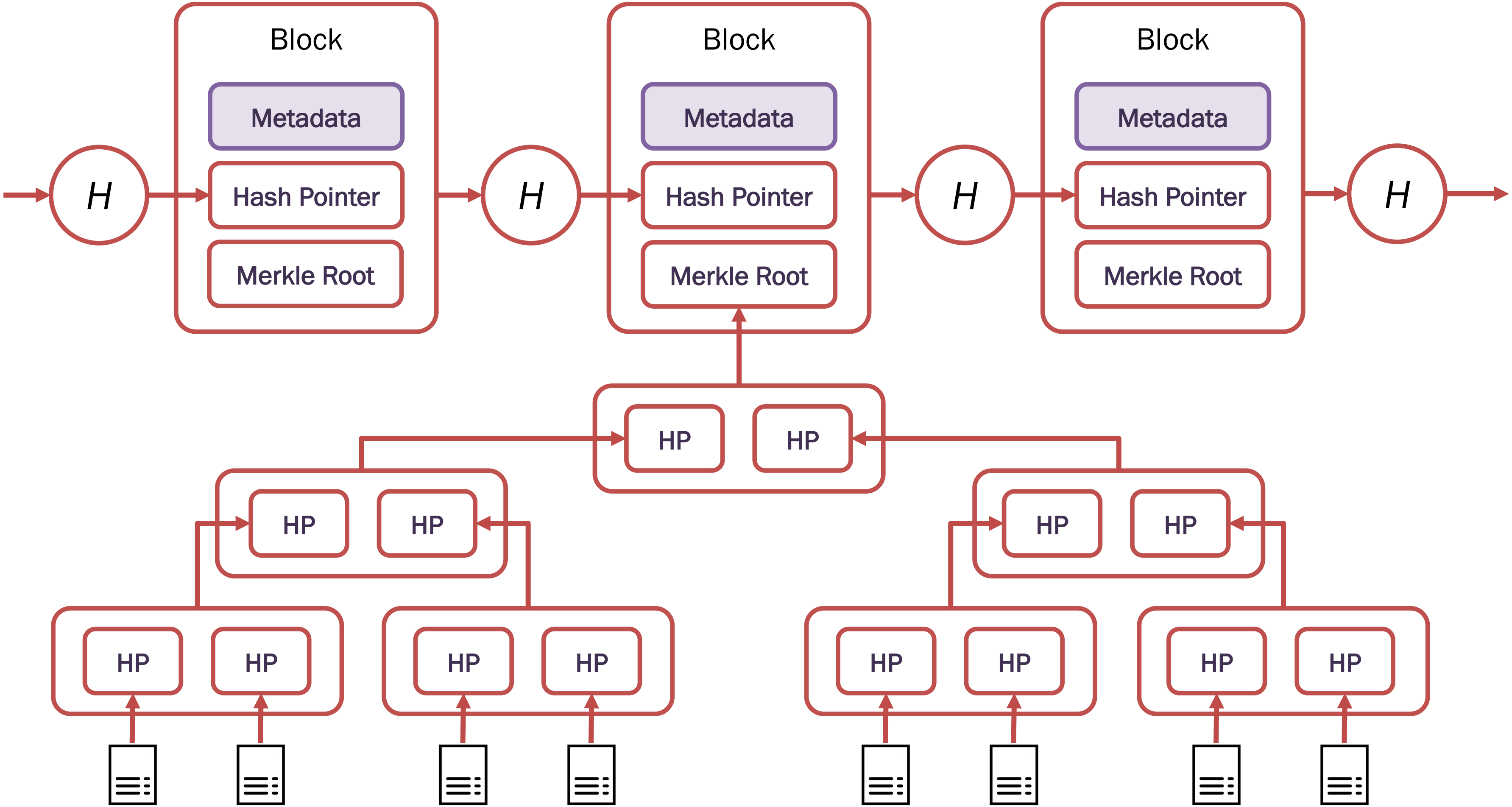
Storing only the Merkle Root is sufficient for Tamper Evidence of all records.



Stored in a Block

Block Metadata may contain Block Index, Timestamp etc.





H
Block
Metadata
Hash Pointer
Merkle Root
H
Block
Metadata
Hash Pointer
Merkle Root
H
Block
Metadata
Hash Pointer
Merkle Root
H
HP
HP
HP
HP
HP
HP
HP
HP
HP
HP
HP
HP
HP
HP