Data Structures,
Digital Signatures and Identities

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University

May 11, 2022

Recap Lecture 2

- *Bitcoin and Blockchains:* What are
  - Preserving value
  - Having a ledger
  - Blocks of transactions
  - Proof of Work
- *Hash Function Basics:*
  - Definition
  - Basic principles
- *Hash Functions Properties:*
  - Collision Resistance
  - Hiding
  - Puzzle Friendliness
- *The Merkle-Damgard Transform:*
  - SHA-256
  - Compression Function
  - Merkle-Damgard Transform

Hash Pointers
Blockchains
Merkle Trees

Digital
Signatures

Identities:
Public Keys

Simple
Online Cash

Overview

Introduction

- *Data Structures:*
  - Hash pointers: pointers & hashing dereferenced values
  - Blockchains: hash pointer linked lists
  - Merkle Trees: hash pointer based binary trees
- *Digital Signatures*
  - Digital signature schemes
  - Generating keys, signing, verifying
  - The unforgeability game
  - Elliptic curve digital signature algorithm
- *Identities*
  - Public keys
  - Properties
- *Simple Online Cash:*
  - Centralized Coin I
  - Centralized Coin II: preventing double spending

Hash Pointers
Blockchains
Merkle Trees

Digital Signatures

Identities: Public Keys

Simple Online Cash

Hash Pointers

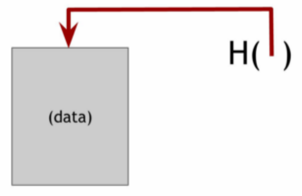
Hash Pointer

From bitcoinbook.cs.princeton.edu

Definition [Hash Pointer]: A *hash pointer* consists of

1. An address that points to a piece of data
2. A hash value of the data stored at that address

Hash Pointers



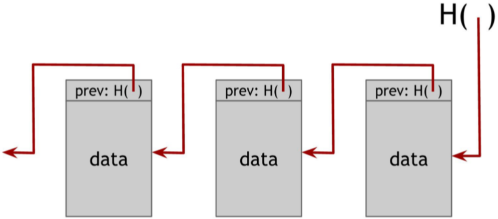
Hash Pointer

From bitcoinbook.cs.princeton.edu

Intuition

- The pointer indicates where the data can be found
- One can verify whether the data was modified:
  - If not modified, hashing the data yields the stored hash value
  - If modified, hashing the data disagrees with the stored value

Block Chain



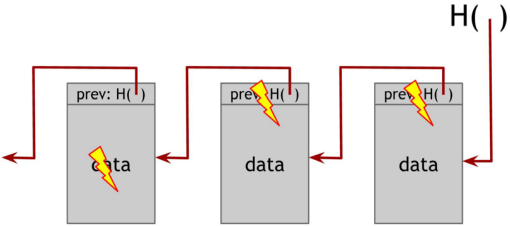
A block chain connects pieces of data using hash pointers

From `bitcoinbook.cs.princeton.edu`

Definition [Block Chain]: A *block chain* is

- a *linked list* where entries are referred to as *blocks*.
- Blocks are linked with hash pointers (instead of ordinary pointers)
- The *head* is a hash pointer to the most recent (rightmost) block

Tamper-Evident Log

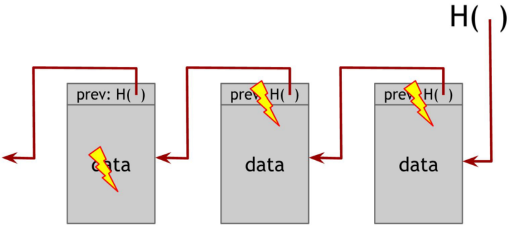


Modifying data implies clashes in subsequent blocks

From `bitcoinbook.cs.princeton.edu`

- Modifying block k leads to clash with hash in block $k+1$
- *Recall:* This works because hash function is *collision resistant*
- The evil-doer continues, modifies the hash in block $k+1$
  ☞ clash in block $k+2$ ...

Tamper-Evident Log

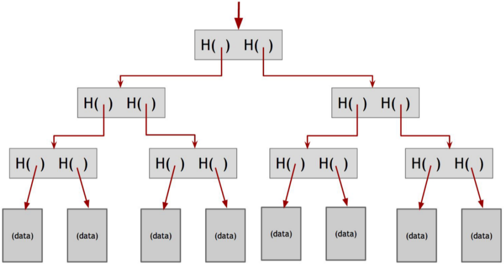


Modifying data implies clashes in subsequent blocks

From `bitcoinbook.cs.princeton.edu`

- ... eventually the evil-doer will try to modify the head
- *Conclusion:* Storing head safely renders entire block chain tamper-evident, up to first block
- *Note:* First block often referred to as *genesis block*.

Merkle trees



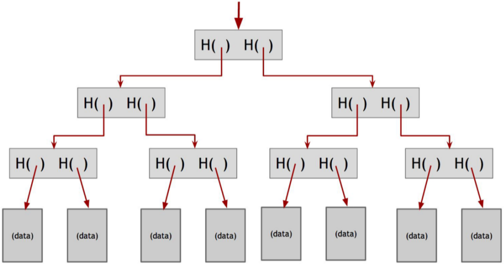
Merkle Tree

From `bitcoinbook.cs.princeton.edu`

Definition [Merkle Tree]: A *Merkle tree* is

- a *binary tree* where nodes are linked with hash pointers
- *Leaves* refer to blocks containing data
- *Internal nodes* consist of two hash pointers, each of which points to one of the two children

MERKLE TREES



Merkle Tree

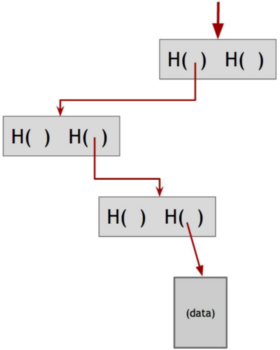
From `bitcoinbook.cs.princeton.edu`

- For accessing data in a Merkle tree, only storing the root (the uppermost node) is required
- The root is referred to as *Merkle root*
- Ordering leaves introduces additional helpful structure
  ☞ *Sorted Merkle tree*

PROOF OF MEMBERSHIP

Task: Prove that a particular data block belongs to the Merkle tree

- Show the data block
- Show the blocks on the path from the data block to the root
- For n nodes in the tree, only $O(log(n))$ items to be shown
- It takes about $log(n)$ time to verify
- Verification further requires only hash of root ☞ modified data will lead to clash eventually



Path to particular block of data

From bitcoinbook.cs.princeton.edu

Proof of non-membership

Task: Prove that a data block does not belong to a *sorted* Merkle tree

- *Reminder:* In a sorted Merkle tree, leaves are ordered
  - E.g. by alphabetical, lexicographical, numerical ordering
  - Every order applies
- *Proof:* Show a path to
  - the block just before the block in question
  - the block just after the block in question
- *Verification:*
  - Both blocks pointed out differ from the block in question
  - Therefore, if the two blocks are consecutive in order, non-membership of data block is verified

Hash Pointers
Blockchains
Merkle Trees

Digital
Signatures

Identities:
Public Keys

Simple
Online Cash

Digital Signatures

Basic Properties

1. Only you can make your signature
2. Anyone can verify it is your signature
3. The signature is tied to a specific document
    - One cannot re-use the signature for other documents

Digital Signature Scheme I

Definition [Digital Signature Scheme]: A *digital signature scheme* consists of the following three algorithms:

1. $\text{sk}, \text{pk} = \text{generateKeys}(\textit{keysize})$ generates a secret key ″sk″ and a public key ″pk″ of size *keysize*
   - sk is kept private and used to sign messages
   - pk is published
   - anyone with pk can verify signatures generated using sk
2. $\text{sig} = \text{sign}(sk, message)$ signs a *message* using *sk* and generates signature ′sig′
3. $\text{isValid} = \text{verify}(pk, message, sig)$ is true if *sig* is a signature for *message* having been signed using ′sk′ that is paired with ′pk′

Digital Signature Scheme II

Properties

- Valid signatures must verify. In other words,

$$\text{verify}(pk, message, sig) == True$$

  if and only if

$$\text{sig} = \text{sign}(sk, message)$$

- generateKeys and sign can be randomized algorithms
  - In fact, generateKeys should be randomized, because it should generate different keys for different people
- 'verify' is always deterministic
- Signatures are *existentially unforgeable*

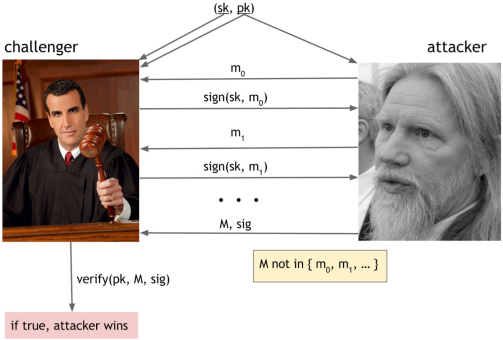
The Unforgeability Game

- The *adversary*, holding pk, claims that he can forge signatures
- The *challenger*, holding both sk and pk, tests this claim
- The *adversary* is able to make the challenger sign a reasonable amount of documents of his choice
  - While 1 million documents may be reasonable,
  - 2^{80} documents is certainly unrealistic
  - *Formally:* number of documents polynomial w.r.t. *keysize*
  - Models real life conditions: an attacker may have means to manipulate the one who signs
- *End of Game:* The *adversary* generates signature 'sig' for unseen document M. If

$$\text{verify}(pk, M, sig) == True$$

the *adversary* wins. Otherwise, the challenger wins.

The Unforgeability Game



The Unforgeability Game.

From `bitcoinbook.cs.princeton.edu`

- The challenger holds (sk,pk), the attacker (adversary) only pk
- The attacker receives signatures for messages $m_0, m_1, \ldots, m_n$ and eventually generates a signature for $M \notin \{m_0, \ldots, m_n\}$
- If $\text{verify}(pk, M, sig) == True$, the attacker wins

UNFORGEABILITY

DEFINITION [UNFORGEABILITY]: For someone who

- knows the public key 'pk'
- knows

$$\mathrm{sig_i} = \mathrm{sign}(sk, m_i)$$

  for a reasonable amount of messages $m_i, i = 0, ..., n$,
- *but does not* know 'sk',

the chances to generate a valid signature for $M \notin \{m_0, ..., m_n\}$ in the name of (sk,pk) are so small that it never happens in practice

ECDSA: Facts

- The ECDSA implements the digital signature scheme in Bitcoin
- ECDSA is a US government standard
  - Update of earlier DSA to implement elliptic curves
- Bitcoin uses ECDSA over standard elliptic curve ″secp256k1″
  - Provides 128 bits of security; as difficult to break as performing 2^{128} hash function calls
  - Much more common: using elliptic curve ″secp256r1″ ☞ using ″secp256k1″ is particularity of Bitcoin

Bitcoin Digital Signature Algorithm

Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA: Characteristics

- Private key: 256 bits
- Public key, uncompressed: 512 bits
- Public key, compressed: 257 bits
- Message to be signed: 256 bits
- Signature: 512 bits

ECDSA: Practical Issue

- ECDSA can only sign messages of length 256 bits
- *Solution:* Hash messages, and sign the resulting message digests

Digital Signatures: Randomness

- *Reminder:* ’generateKeys’ relies on randomized algorithm
- ’sign’ itself may rely on randomized algorithms as well
- Bad source of randomness when calling ’generateKeys’can leak secret key
  ☞ when making signatures using badly randomized keys, revealing public key can leak private (secret) key
- Once private key is leaked, adversaries can forge signatures

Hash Pointers
Blockchains
Merkle Trees

Digital
Signatures

Identities:
Public Keys

Simple
Online Cash

IDENTITIES: PUBLIC KEYS

Motivation

- *Digital signatures:* Public keys ′pk′ reflect virtual identities
- ′pk′: ″the one″ who signs a message
- Anyone holding the matching secret key ′sk′ can speak for ′pk′

Consequences

- One can create new identities whenever one wants
  ☞ just call ′generateKeys′ to create another one
- *Practice:* Publish *hash of ′pk′* as identity
- Verifying identities of messages:
  - Check that ′pk′ hashes to published string
  - Verify message using ′pk′
- Identities look entirely random ☞ ″anonymous face in the crowd″

Identities: Final Remarks

- Real identities representing 'pk' cannot be uncovered by examining 'pk'.
- However, one can study statements made by 'pk'
  ☞ Statements may reveal real world identity
- *Remedy:* Create new identities, and continue to work with them (still issues remaining... to be dealt with later)
- Public key identities support *decentralization*:
  - Nobody stores identities
  - Anyone can destroy and create identities at free will
- Identities are often (and somewhat misleadingly) referred to as *addresses*
- Good randomness prevents duplicating identities in practice

Hash Pointers
Blockchains
Merkle Trees

Digital Signatures

Identities: Public Keys

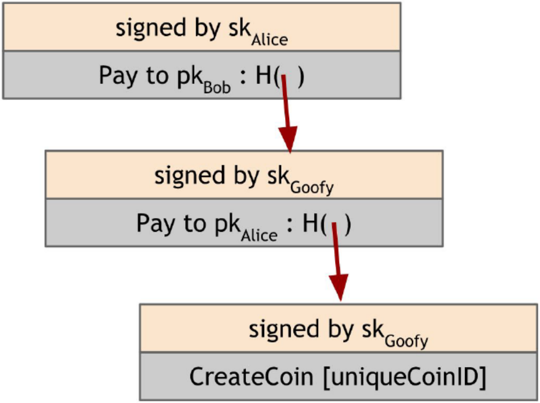
Simple Online Cash

Online Cash: Simple Coin I

Rules

1. *One designated identity (”Goofy”) can create coins*
   - Each coin has unique coin ID: `UniqueCoinID`
   - ”Goofy” constructs string ”CreateCoin [`UniqueCoinID`]
   - ”Goofy” signs string using his secret key
   - Anyone can verify that the new coin was created by ”Goofy”
2. *Anyone owning a coin can tranfer it to anyone else*
   - Transfers are strings ”Pay *[Coin]* to *[Alice]*”
   - *[Coin]* is a hash pointer referencing the coin
   - *[Alice]* is a public key
   - The one who transfers the coin signs the pay string
   - Anyone can verify ownership by following hash pointers until coin creation
   - Verify as well that all prior owners signed transactions

SIMPLE COIN I



Chain of transactions, including creation (bottom) and two times spending a coin

From `https://bitcoinbook.cs.princeton.edu`

Simple Coin I: Security Issue

Double Spending

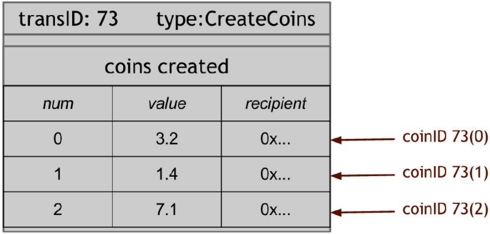
- Alice transfers coin to Bob
- Alice transfers same coin also to Chuck
- When verifying both transactions, both appear to be valid
  - Following hash pointers approve of existence of coin and Alice's ownership
  - Both Bob and Chuck point out that Alice commits to transfer by her signature

Online Cash: Simple Coin II

Preventing Double Spending

- The designated identity (”Scrooge”) publishes all transactions as blocks of a block chain
- ”Scrooge” signs each transaction (one block of the chain)
- Transactions can be either of type
  - ”CreateCoins” to create new coins
  - ”PayCoins” to transfer coins between identities
  - ”Scrooge” is the only one to create coins
- We also make transactions more flexible
  - create fractions of coins
  - break coins into smaller parts in transfers

SIMPLE COIN II: CREATING COINS



Transaction that creates coins

From `https://bitcoinbook.cs.princeton.edu`

- Coins can have different values
- Coins are assigned to owners
- ”Scrooge” signs the transaction

Simple Coin II: Transferring Coins

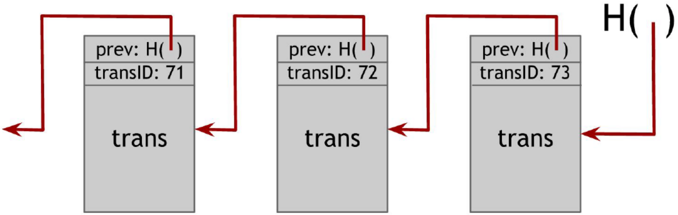
transID: 73	type:PayCoins	
consumed coinIDs: 68(1), 42(0), 72(3)		
coins created		
num	*value*	*recipient*
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...
signatures		

Transaction that transfers coins

From `https://bitcoinbook.cs.princeton.edu`

- Consumes (destroys) coins; creates new ones of same total value
- Lists ID's of all coins consumed
- Owners involved sign transaction

Simple Coin II: Block Chain



Blockchain supporting simple coin example

From `https://bitcoinbook.cs.princeton.edu`

- Blocks signed by designated identity ”Scrooge”
- Actions in transaction signed by coin owner or creator
- Everyone can verify validity of transactions
- Linearity: double spending attempts immediately evident

Simple Coin II: Drawback

Issue: Central Authority

”Scrooge” cannot fake transactions. However:

- ”Scrooge” creates coins
  - ”Scrooge” could create many coins, implying loss of value of coins
  - ”Scrooge” can be selective in distributing coins
  - ”Scrooge” can keep many coins for himself
- ”Scrooge” appends blocks to block chain
  - ”Scrooge” can deny service to particular identities
  - ”Scrooge” can force identities to pay transaction fees

Materials / Outlook

- See *Bitcoin and Cryptocurrency Technologies*, 1.2 – 1.5
- See `https://bitcoinbook.cs.princeton.edu/` for further resources
- Next lecture: "Decentralization"
  - See *Bitcoin and Cryptocurrency Technologies* 2.1 – 2.5