# Cryptography and Blockchain Technology

Shih-Fan Chou

sfchou@mail.ntust.edu.tw


Blockchain and Smart Contracts: Design Thinking and Programming for Fintech, Lo, Swee Wo, Wang, Yu, Lee, David, 2021, 978-9-811-22368-6, World Scientific.

# Introduction

- We discuss the fundamentals of information security and cryptography that undergird blockchain technology
- Information security concerns the protection of data from unauthorized access or modification
- Cryptography concerns the study of mathematical techniques used to achieve information security objectives when facing adversarial attacks
  - The art of writing or solving codes
  - Hash function, digital signature, public key infrastructure, encryption and privacy techniques
  - References mainly to the Bitcoin blockchain for application examples

# Introduction

- What made classical cryptography an art was the fact that there was little theory behind the construction or decryption of codes, and no systematic way of thinking about the requirements a secure code had to satisfy
- Its purpose was primarily to achieve secrecy, and because of the great expense involved, its use was limited to governments and military organizations
- Modern cryptography is not just an art but a science and a mathematical discipline
- The field now relies on rigorous proofs of security
  - A suite of algorithms based on the intractability of difficult problems, which are problems that cannot be solved in a "reasonable amount of time" by an efficient attacker

# Introduction

- Modern cryptography is integral to nearly all computer systems
  - Secret communication to the protection of the user
  - Data at rest (in storage)
  - Data in transit (sent over a network)
- Most of us are users of cryptography on a daily basis, whether we are sending an e-mail or paying for a ride with our transportation card

# Security Objectives

- The goal of cryptography is to achieve information security
- Three main objectives of information security have been known as the "CIA triad": confidentiality, integrity, and availability
- With the evolution of computing and technology such as cloud services and the peer-to-peer network, the CIA triad is no longer considered sufficient to ensure protection against attacks on the data and user
  - Two additional security objectives: authentication and accountability

# Confidentiality

- Confidentiality extends to both data confidentiality and user confidentiality (or user privacy)
- Data Confidentiality
  - Shield data from all unauthorized parties
    - Is achieved through data encryption
  - Shield data from unauthorized parties who are differentiated on the basis of access rights
    - E.g., A patient's hospital records will contain different types of data, such as identifying information, test results, diagnoses, treatments, prescriptions and medical insurance coverage
    - Hospital employees should only have access to the records on a need-to-know basis
    - The records management system will therefore need an authorization protocol that gives users different levels of access based on their identity

# Confidentiality

- User Privacy
  - There is reason to expect that a user remains unidentifiable
  - Ensure untraceability
    - Conceal the routes of goods, money, data and other operations, so that an observer cannot follow the trail to deduce identifying information about a user
  - Ensure unlinkability
    - Conceal the connection between multiple pieces of data from the same user, so that they appear unrelated to an observer
  - Largely remains an area of theoretical concern in academia but is also gaining more attention as big data, Internet-of-Things (IoT) and the usage of blockchain become more pervasive

# Integrity

- Data integrity is concerned with ensuring that data are not tampered
- Though often conflated with confidentiality, integrity is a distinct objective
- False assumption
  - Encryption is sufficient to ensure data integrity
    - Since encryption conceals the real content of a message, the content cannot be modified in any meaningful way
  - Instead, it is possible to manipulate a piece of encrypted data in ways that simply invalidate its content
    - Recipient has no idea if the encrypted data are received with errors or were maliciously modified
    - Need tools specific to integrity protection to secure data from unauthorized modifications
- Is achieved through the use of hash function, message authentication codes and/or digital signature

# Availability

- Data and online services are of no use if they are not available when authorized users need them
- Is closely associated with a malicious activity known as a Denial-of-Service (DoS) attack
- In IoT networks, a DoS attack has evolved to a Distributed DoS (DDoS) attack
  - An attacker first injects malicious software (or malware) into any vulnerable devices connected to the Internet
  - Once infected with malware, devices become bots that can be remotely controlled to send bogus requests to the target server
- Availability is usually addressed through good system design, such as intrusion detection or intrusion prevention systems

# Authentication

- Authentication extends to both data authentication and user authentication
- Data authentication
  - Verify that the origin of a piece of data is what it claims to be
  - Message authentication codes and digital signatures
- User authentication
  - Verify that a user is whom they claim to be
  - Authentication factors to a login system, based on what you know (e.g., username and password), what you have (e.g., access card), or who you are (e.g., biometric information)
  - Message authentication codes and digital signature, where users are required to demonstrate something they know (i.e., the secret/private key)

# Accountability

- Thwart a user from falsely denying that they have sent or received a particular communication
- Similar to the concept of non-repudiation
- Accountability goes a step further by creating a digital artefact that makes it impossible for the sender to deny having sent the data
- Draw an undeniable link between the data and the users who have processed it, ensuring accountability for actions performed on the data
  - Digital signature
- With the rise of "deepfakes" and disinformation, and eroding public trust in news media, ensuring accountability is increasingly becoming a matter of social and political stability

# Attacks

- *Know thyself, know thy enemy, and in every battle you will be victorious.*

— *Sun Tzu*, The Art of War

- To respond effectively to a threat to information security, we must know what we are up against
- Understanding possible attack strategies enables us to put in place the appropriate safeguards to ensure the security of a system and its cryptographic techniques

# Attacks

- Active attack
  - The attacker performs aggressive and often blatant actions with the aim of affecting or altering a system's operations
    - E.g., e-mail phishing, modifying data in transit between users, mounting a DDoS attack, and mounting a brute-force attack (trial and error) to guess a username and password combination
  - An active attack is easy to detect but difficult to defend, involving a thorough analysis to close all loopholes in a system or cryptographic technique
  - By the time an attack is detected, the damage may already have been done or underway

# Attacks

- Passive attack
  - The attacker uses covert methods to tap into a network and eavesdrop on or record communications
  - The goal is to gain access to a system or steal information surreptitiously
    - E.g., surveillance, man-in-the-middle attacks and keystroke logging
  - Passive attacks are difficult to detect but easy to prevent
  - One of the most straightforward methods is to encrypt communication between users

# Cryptography

- Key Terms: Bit, Scheme, Algorithm, Function
- A bit (short for binary digit) is the smallest unit of computer data
- 4 bits of data therefore have $2^4$ or 16 possible combinations

| | | | |
|---|---|---|---|
| 0000 | 0100 | 1000 | 1100 |
| 0001 | 0101 | 1001 | 1101 |
| 0010 | 0110 | 1010 | 1110 |
| 0011 | 0111 | 1011 | 1111 |

- A cryptographic scheme is an overall description of how a security system works
- Schemes consist of one or more algorithms, which are sets of instructions defining a sequence of operations to be carried out by a computer
  - E.g., an encryption scheme may consist of a secret key generation algorithm, an encryption algorithm and a decryption algorithm

# Cryptography

- A function is a process or relation between values that takes some input and produces some output
  - E.g., the secret key generation algorithm has a KeyGen() function that takes as input the security parameter *n*, and produces a secret key *k* of *n* bits
- In fact, almost all cryptographic techniques can be broken given enough time and computing power
- We still consider them secure because what is "enough" is generally not achievable in our lifetime or several lifetimes
- Instead of unconditional security, we aim for cryptographic techniques to be computationally secure

# Cryptography

- A cryptographic scheme is computationally secure if no efficient attacker is able to break it in a reasonable amount of time
  - Efficient attacker: an attacker with a reasonable amount of computing power
  - A reasonable amount of time: a time frame during which the ability to solve a problem has ceased to be useful or interesting
    - E.g., 4-bit key v.s. 256-bit key
- We can think of computational security as a practical balance between security and efficiency
- All computationally secure cryptographic schemes rely on the provision of pseudorandomness
  - The distribution of a string should satisfy statistical tests of randomness, even though it is in fact not truly random
  - E.g., Ciphertexts
  - Users who do not have enough information, the ciphertext is just a pseudorandom string

# Cryptography

- Pseudorandomness is important because it prevents a ciphertext from inadvertently disclosing information about the statistical distributions of the secret key or the original message
- An attacker cannot distinguish one ciphertext from another
- Consider an encryption scheme, where an encryption algorithm is denoted as Enc(), $m$ is a message in plaintext, c is the encrypted ciphertext and $k$ is the encryption key

$$c \leftarrow Enc_k\,(m)$$

- A computationally secure encryption scheme requires that given two ciphertexts $c_1$ and $c_2$ ,

$$c_1 \leftarrow Enc_k\,(m_1\,)$$

$$c_2 \leftarrow Enc_k\,(m_2\,)$$

- Both $c_1$ and $c_2$ are indistinguishable to an efficient attacker
  - Both $c_1$ and $c_2$ have pseudorandom distributions, and do not reveal any information about the statistical distributions of $k$, $m_1$ or $m_2$

# Blockchain

- The concept of a blockchain first appeared in 2008 in a white paper by Bitcoin inventor Satoshi Nakamoto
  - Nakamoto proposes Bitcoin as a peer-to-peer payment system, and "a chain of blocks" as the structure for recording Bitcoin transactions in an immutable and transparent manner
- The idea of using digital currencies for decentralized peer-to-peer payment is not new
- Before Bitcoin, without a trusted central authority (such as a bank) to oversee transactions, there was no practical way to thwart three possible acts by malicious users:
  - Spending without authorization (i.e., without ownership of the account)
  - Spending without having enough balance
  - Double-spending (i.e., sending the same amount of digital currencies to more than one receiver without deduction in the account balance)

# Blockchain

- Consider the banking system
- To initiate a fund transfer, users first have to log in to their Internet banking account (proves a user's ownership of the account)
- When the user requests a fund transfer, this involves the bank checking, among others, if the user has enough balance to perform the transaction
- If so, the amount transferred will be deducted from the balance, so double spending will not be possible
- If a user were to copy and resend the transaction confirmation, this would not result in multiple duplicate fund transfers to the same receiver, because the bank would prohibit fraudulent transactions

# Blockchain

- Bitcoin blockchain was the first sound design of an open and secure *Distributed Ledger Technology (DLT)* that thwarts these three malicious acts without the need for a central authority
- DLT is a method of record-keeping whereby every person involved in the record-keeping possesses a copy of the ledger
- Whenever a new record is created, it is updated in each and every user's ledger

# Blockchain

- DLT is straightforward to implement in a trusted setting, which assumes either a trusted central authority with no self-interest, or that all users will behave honestly by:
  - Not sending bogus or meaningless transactions that will consume unnecessary computation and storage resources
  - Updating the ledger in an honest and timely manner whenever a new transaction comes in
  - Not making unauthorized deletions, insertions, or modifications to transaction details on the ledger
  - Including only valid transactions on the ledger and rejecting all invalid transactions

# Blockchain

- A trusted setting without a central authority is unrealistic in most real-life settings
- Blockchain, specifically public blockchain like Bitcoin, does not assume trust
- It enables a decentralized peer-to-peer payment system where the ledger recording the transactions is maintained and updated by a group of users who do not necessarily trust each other
- It is able to bring the community of users together to agree on a single version of the truth through "crypto-economy", which combines cryptography and economic incentives
  - Cryptography records data on the transactions in a secure manner and ensures that it is costly for a user to cheat or perform unauthorized actions
  - Economic incentives reward honest and trustworthy users who help to maintain the integrity of the ledger

# Cryptographic Hash Function

- Hash function: a method of mapping input of arbitrary length onto an output of fixed length
  - We call this output a hash value, or "hash" or "message digest"
- A hash function, denoted as Hash(), takes as input a message *m* of arbitrary length and produces a hash value of fixed length *l*:

$$H = \text{Hash}(m)$$

- We can also write this process as

$$\text{Hash}(): \{0,1\}^* \rightarrow \{0,1\}^l$$

- Usage of hash functions
  - The generation of hash tables for indexing data to enable efficient data storage and retrieval
- However, cryptographic hash functions are different as they possess rigorous security definitions
  - Cryptographic hash functions can be used for file integrity protection, generation of one-time passwords and digital signature computation, and so on

# Cryptographic Hash Function

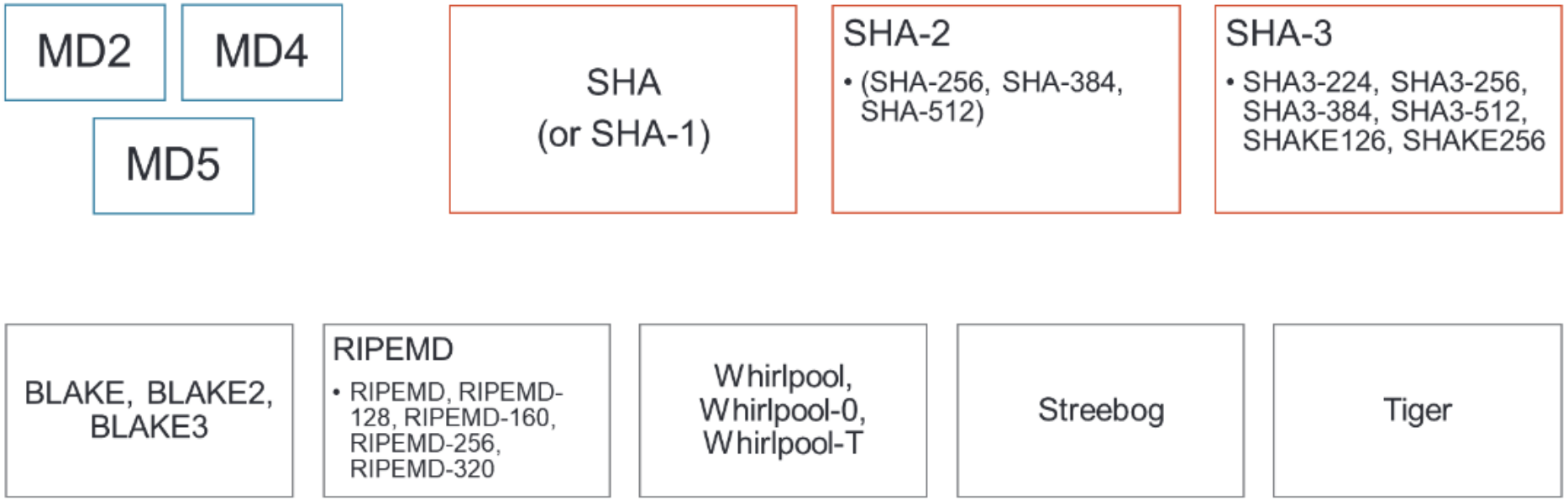



**Figure 1.1:** Examples of hash functions.

# Cryptographic Hash Function

- A blockchain is an immutable ledger
  - This immutability is a result of the use of distributed ledger technology coupled with hash functions
  - Do not allow any modification to a block and its transactions to go undetected

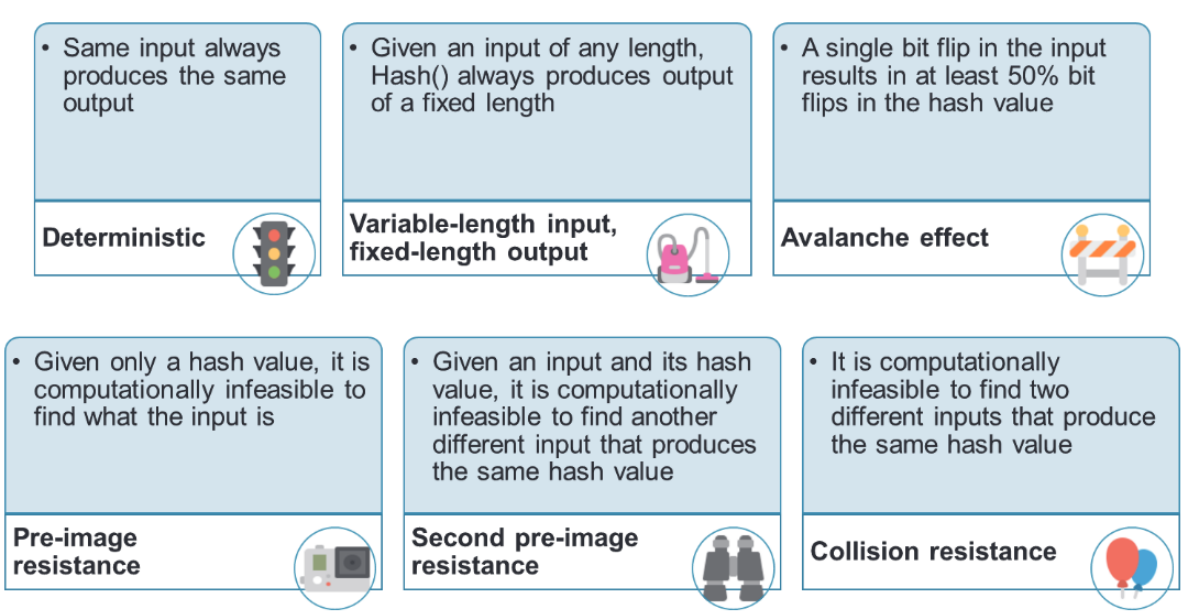



**Figure 1.2:** Properties of hash function.

# Cryptographic Hash Function

- Formally, the properties of pre-image resistance, second pre-image resistance, and collision resistance can be written as follows:
  - Pre-image resistance (or one-wayness): Given *h* where *h* = Hash($m_1$ ), it is infeasible to find out the value of $m_1$ without trying on average $2^l - 1$ times, where *l* is the length of the hash value in bits
  - Second pre-image resistance: Given $h_1$ = Hash($m_1$), it is computationally infeasible to find another message $m_2$, where $m_2 \neq m_1$ but Hash($m_2$) = Hash($m_1$)
  - Collision resistance: It is computationally infeasible for an attacker to find two distinct inputs $m_3$ and $m_4$ , such that Hash($m_3$ ) = Hash($m_4$ )

# Cryptographic Hash Function

- Many attacks against hash functions exploit the mathematical underpinnings of collision resistance based on the "birthday problem"
- We are tasked to find two people who have the same birthday out of a roomful of people
- Assuming that all 365 days in a year are equally probable for a birthday (and ignoring leap years), we need at most only 23 people in the room to have a 50% probability of finding two people who share a birthday
- It may seem counterintuitive that a relatively small set produces such a high probability of collision

# Cryptographic Hash Function

- However, consider the following problem: How many pairs can we form out of four persons?
- The answer is six, given by the formula for $C_r^n$
- Thus, $C_2^4 = 6$

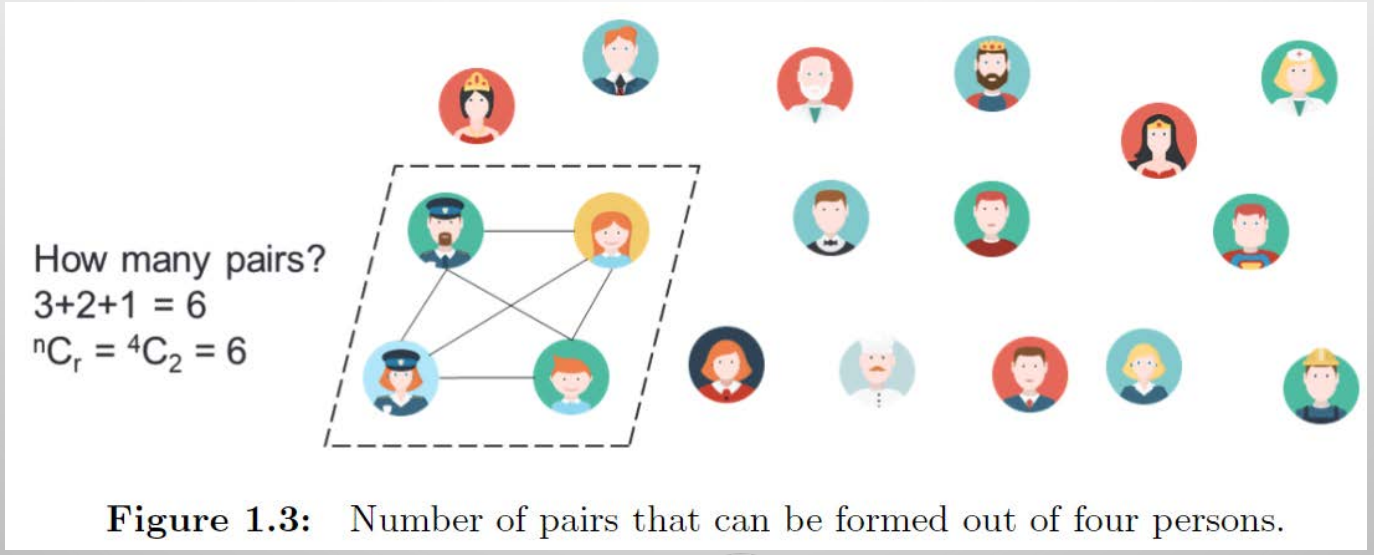



**Figure 1.3:** Number of pairs that can be formed out of four persons.

# Cryptographic Hash Function

- The length of the hash value is also closely related to security
- The cryptographic community witnessed this in 2017 when Google announced that it had achieved the first concrete collision attack on SHA-1, having developed a technique to generate a collision
  - SHA-1 produces a 160-bit hash value, meaning that on average it takes $2^{80}$ tries to find a collision through a brute-force attack, which is still computationally infeasible given today's computing power
  - However, Google was able to expose a practical attack that exploited flaws in the design of SHA-1 that paved the way for it to be phased out of use

# Cryptographic Hash Function

- Hash functions form the basis of a myriad of cryptographic techniques, including message authentication codes, digital signatures, one-time passwords and checksums
- Hash function is also used to protect the integrity of transactions and blocks
    - This is built upon the distributed ledger technology

# Merkle Hash Tree

- Merkle Hash Tree is a binary tree
  - Each parent node has two children nodes
  - It is created using a bottom-up construction, where nodes at the bottom are called leaf nodes
  - Each leaf node contains cryptographic hash of a piece of data and every non-leaf node contains cryptographic hash of its children nodes

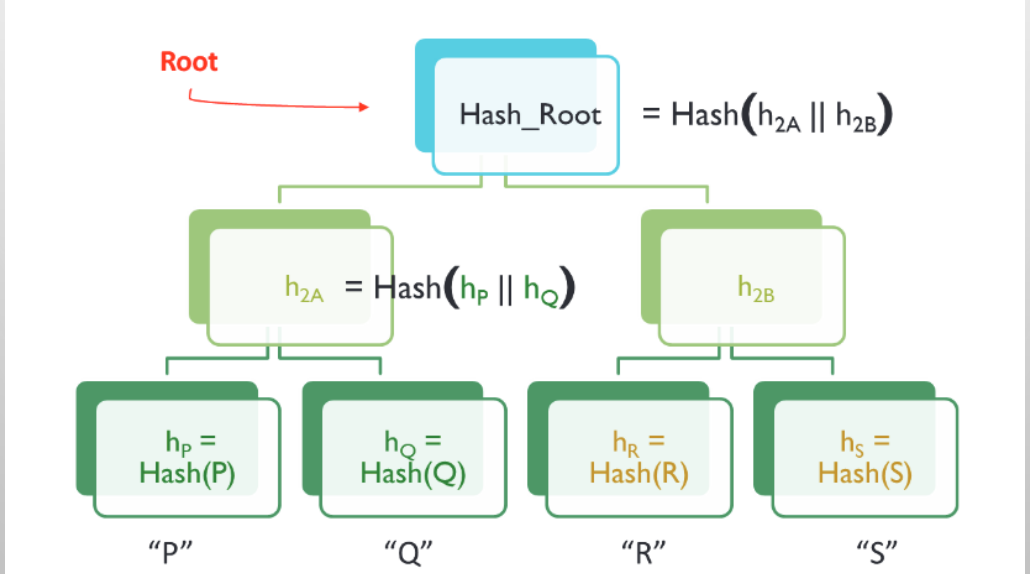



**Figure 1.5:** An example of Merkle Hash Tree with data "P", "Q", "R" and "S".

# Merkle Hash Tree

- Any changes in the data “P”, “Q”, “R” or “S” would always be reflected in the *Hash_Root*
- The construction of a Merkle Hash Tree allows efficient and secure verification of data
- If the Merkle Root, *Hash_Root*, is published and immutable to changes, then one can verify whether the data “P” is included in the Merkle Hash Tree by asking the Prover to supply proofs consisting of $\{h_Q, h_{2B}\}$
  - The tuple $\{h_Q, h_{2B}\}$ is called the Merkle Path for data “P”
- The Merkle Path for data “P” together with the hash of “P” is then sufficient for the Verifier to reconstruct its own Merkle Hash Tree and compare the resulting Merkle Root with the immutable *Hash_Root*
  - If the hash values match, then the Verifier can be ascertained that the data “P” is included in the Merkle Hash Tree
  - If “P” is not included in the Merkle Hash Tree but the Prover lies that it is, the Prover needs to find a collision in the Merkle Root hash

# Merkle Hash Tree

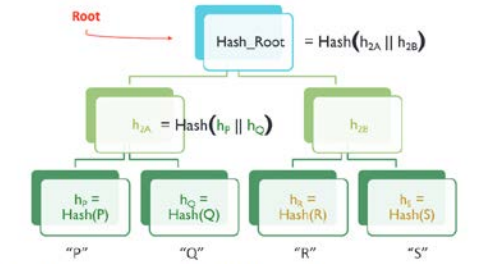


Figure 1.5: An example of Merkle Hash Tree with data "P", "Q", "R" and "S".

- Suppose the Verifier checks if data "N" is included in the Merkle Hash Tree shown in Figure 1.5
- To lie that "N" is in the Merkle Hash Tree (instead of "P"), the Prover must supply proofs that are in one of the following forms:
  - $\{h_1, h_{2B}\}$, where $h_1$ is a hash value that will result in the same $h_{2A}$ when computed with $h_N$, i.e., Hash($h_N \mid\mid h_1$) is the same as Hash($h_P \mid\mid h_Q$)
  - $\{h_Q, h_2\}$,where $h_2$ is a hash value that will result in the same *Hash_Root* when computed with the Hash($h_N \mid\mid h_Q$), i.e., Hash(Hash($h_N \mid\mid h_Q$)$\mid\mid h_2$) is the same as Hash($h_{2A} \mid\mid h_{2B}$)
  - $\{h_3, h_4\}$, where $h_3$ and $h_4$ are both hash values that when combined with $h_N$ will result in the same *Hash_Root* that has been published and immutable
- To supply either one of the proofs above, the Prover will have to find a collision in the Merkle Root (*Hash_Root*), an act which is computationally infeasible due to the security properties of a hash function

# Hash Function in the Bitcoin Blockchain

- A blockchain is essentially a chain of blocks
- It is a digital ledger, and the "digital chain" is enabled by the use of hash function

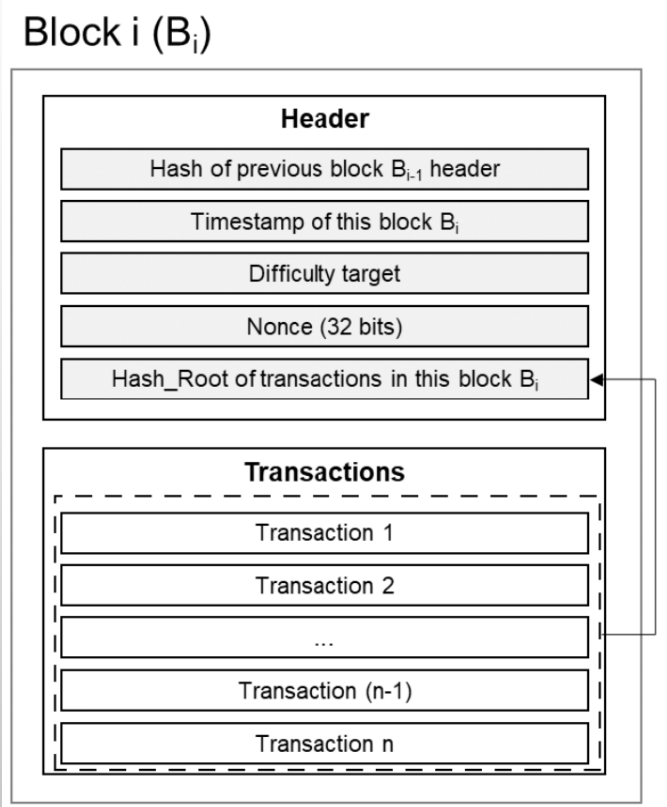



**Figure 1.6:** Bitcoin block header structure.

# Hash Function in the Bitcoin Blockchain

- In the *i*th block, denoted as $B_i$, there is a list of transactions
  - These transactions are integrity-protected using a Merkle Hash Tree, and the Merkle Root is part of block $B_i$'s header
- To integrity-protect the block header, Bitcoin does not employ a straightforward calculation of the block header's hash
- Instead, the proof-of-work (PoW) consensus algorithm in Bitcoin requires miners to vary the 32-bit nonce and iteratively calculate the block header hash until the resulting hash satisfies the difficulty target
- Without loss of generality, we refer to the block header hash that satisfies the difficulty target as the "valid block header hash"

# Hash Function in the Bitcoin Blockchain

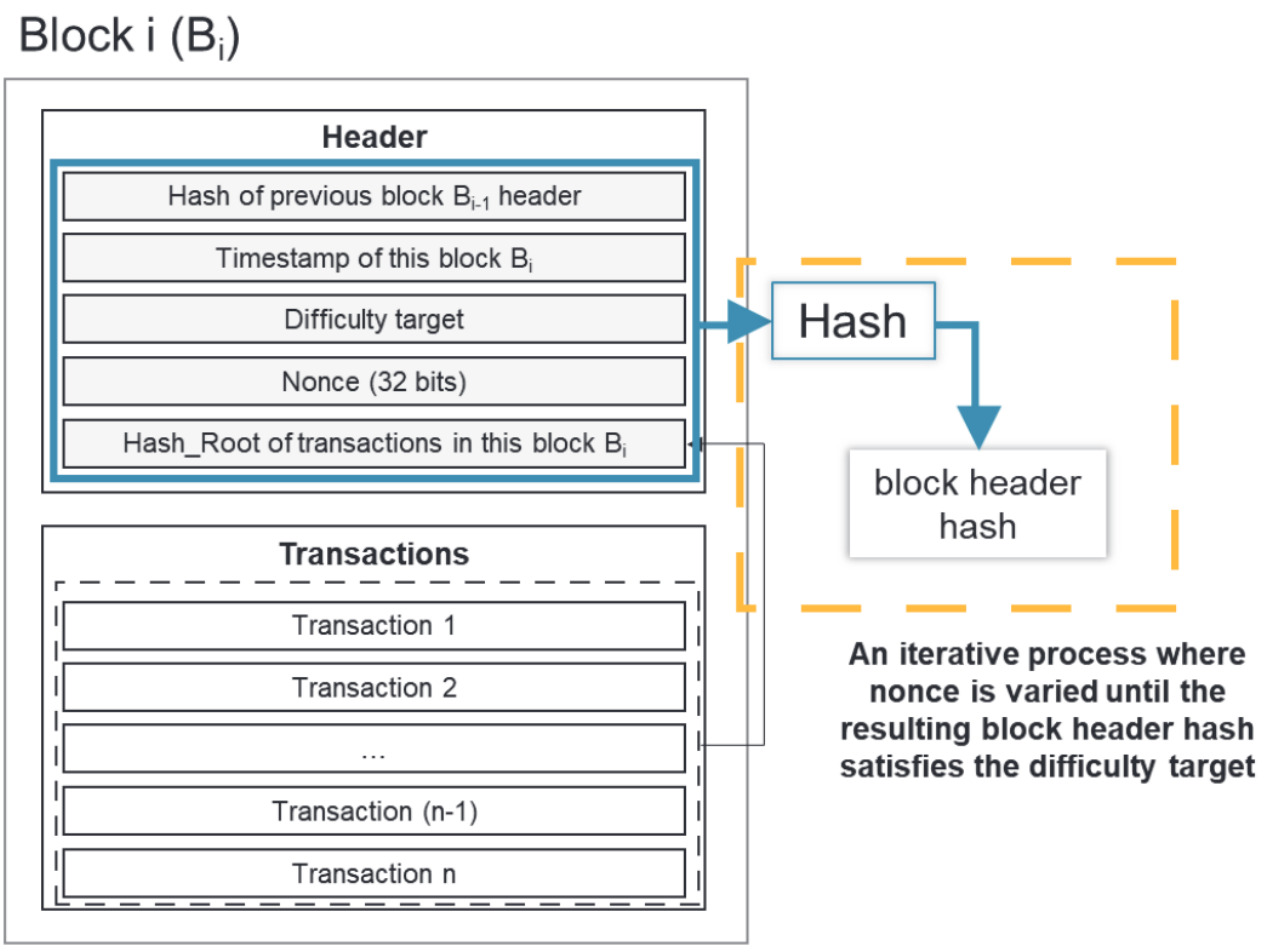



**Figure 1.7:** Bitcoin calculation of block header hash (Proof-of-Work).

# Hash Function in the Bitcoin Blockchain

- As a secure hash function outputs a hash value that is pseudo-random and it is infeasible to predict which nonce would result in a valid block header hash, the PoW consensus algorithm is essentially a lottery system among miners
- Each miner can only vary the nonce, recompute the block header hash and check if the block header hash is valid
  - This effort constitutes proof of work
- Each miner has an equal probability in finding a nonce that gives a valid block header hash

# Hash Function in the Bitcoin Blockchain

- When a valid block header hash is found, the miner will broadcast the block in its entirety to the network
- Other miners in the network will accept this block as valid by checking that after hashing the solution, the resulting block header hash satisfies the difficulty target
- The value of a difficulty target is part of the consensus rule globally accepted by miners in the network

# Hash Function in the Bitcoin Blockchain

- If the difficulty target states that "A valid block header hash is one that starts with at least four zeros in hexadecimal", then, for a block $B_i$, any one of the following hash values can be regarded as a valid block header hash:
  - 00004a89fd89403a36e19c5d67ffcc9bba626cf8dc9a2109bcdbb2471b952683
  - 0000ed4bf83abd0f4b4ce53ed89fe59285f1016ccf85d738d0f6aa9924e368d4
  - 000002eb9ddb8b6d986196f3bf4a3958b2605e917e734c0d9e24d800dc3cbb6b
  - . . . and the list goes on
- This presents an opportunity for an attacker who attempts to revert a transaction

# Example

- Eve agrees to pay Bob 1 btc (bitcoin) in exchange for a software license activation code
- Eve creates a bitcoin transaction that pays Bob the agreed-upon amount and this transaction is recorded in block 500, $B_{500}$, with a block header hash value of "00006120ab0fbd24a1b61dbdca856c3795314945f405b4e90d9a1d3452b85127"
- Upon seeing the transaction in $B_{500}$, Bob immediately sends the activation code to Eve
- At the same time, Eve creates another block 500 *without* her transaction to Bob (denote this block as $B_{500}$') and performs the proof-of-work to get a valid block header hash for $B_{500}$'

# Example

- Suppose this block has a block header hash value of "000035a474b760c0d13ff431a806553a3364c302117ceff8e08ef5fe37b12353"
- Eve broadcasts $B_{500}$' to the network
- At this point in time, the network would observe two blocks at height 500
  - $B_{500}$ with a header hash value of "00006120ab0fbd24a1b61dbdca856c3795314945f405b4e90d9a1d3452b85127"
  - $B_{500}$' with a header hash value of "000035a474b760c0d13ff431a806553a3364c302117ceff8e08ef5fe37b12353"

# Example

- To other users in the network, both blocks are considered valid
- Bob's fate would then depend on which block is accepted by at least 51% of the users
  - The block which is accepted by majority users and therefore belongs in the longer chain will be considered as the final state of the ledger
  - The discarded block is commonly called a stale block
- If block $B_{500}$' is included in the longer chain and block $B_{500}$ is discarded (because both share the same parent, $B_{499}$, thus only one would remain), Eve's transaction of 1 btc to Bob would not be part of the final state of the ledger
- In Bitcoin, transaction finality is characterized by a concept called the six-block confirmation
  - A transaction will become computationally difficult to reverse when it is buried at least six blocks deep in the blockchain

# Hash Function in the Bitcoin Blockchain

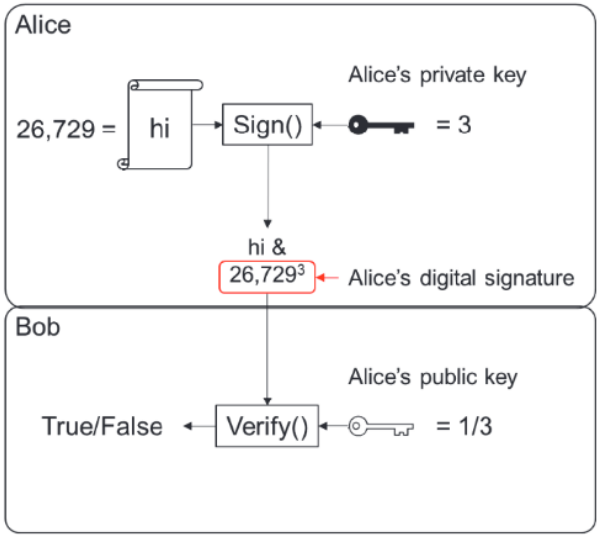
- The Bitcoin PoW's difficulty target is set such that a valid block header hash can only be found in, on average, ten minutes
- In the earlier scenario where Eve attempts to reverse her transaction to Bob, the one-block confirmation of Eve's transaction makes it possible for Eve to create another valid block and possibly get the newly created block accepted by the majority of the users
- However, if Bob sends the activation code to Eve only after $B_{500}$ is appended with six additional blocks, it will make it practically impossible for Eve's attempt to reverse her transaction
  - For her to do so, she needs to create $B_{500}$' , $B_{501}$' , $B_{502}$' , $B_{503}$' , $B_{504}$' , $B_{505}$' , $B_{506}$' and convince the network to adopt her chain of blocks

# Hash Function in the Bitcoin Blockchain

- Due to the security properties of hash function, there is no faster way for Eve to create the chain of *B'* blocks other than to run the proof-of-work algorithm honestly and compute block header hashes iteratively
- As long as Eve does not possess the majority hash power of the network, the six-block confirmation ensures that Eve can never catch up with the rest of the honest miners in her quest of creating the chain of *B'* blocks

# Digital Signature

- Digital signature achieves user accountability, data authentication and integrity protection
- In cryptography, a digital signature is a pseudorandom string that is created via arithmetic computation and secured based on a computationally hard problem




Calculate $26{,}729^3$ (**Sign**). Send message (26,729) and signature 19,096,251,818,489 and public key (1/3) to Bob

Given 26,729 and 19,096,251,818,489 and 1/3. Calculate $19{,}096{,}251{,}818{,}489^{1/3}$ (**Verify**). If the result is the same as the received message "26,729", output TRUE (message is authentic). Otherwise, output FALSE (reject message).

**Figure 1.8:** A high-level but inaccurate illustration of the working mechanism of a digital signature.

# Example

- Alice attempts to send Bob a message “hi”
- This message is converted into its decimal equivalent
  - From ASCII codes 104 for “h” and 105 for “i” to the binary representation 0110100001101001 to the decimal representation 26,729
- A digital signature scheme takes Alice’s private key, that is the decimal number 3, and raises 26,729 to the power of 3
- The result of the exponentiation is Alice’s digital signature
- When Alice sends her message and the corresponding signature to Bob, he will verify the signature using her public key, the number 1/3, by raising the digital signature to the power of Alice’s public key
- If the result of the exponentiation is equal to Alice’s message, then Bob concludes that the message is authentic and it indeed comes from Alice

# Digital Signature

- Signing requires the signer's message and private key
- Verifying requires the verifier to perform the arithmetic comparison using the signer's public key, message, and digital signature
- Actual implementation of digital signature addresses the following practical considerations:
  - Our input messages to sign are usually much longer than a statement such as "hi"
  - Our public and private keys are long (around 1024 bits, i.e., a number that is around 309 digits, or 2048 bits) and their relationship is not a simple $a$ and $1/a$
    - The relationship between private and public keys is such that given a user's public key, it is computationally infeasible to find out or compute the user's private key
  - Modular arithmetic is used and we leverage many important characteristics of modular arithmetic
    - The ability to work only with integers, the provision of groups, rings and fields
    - Assumptions of computationally hard problems when modular arithmetic is applied

# Digital Signature Scheme

**Key Generation**

- The key generation algorithm has a function KeyGen.DS() that takes as input a security parameter $n$ and produces a private signing key $sk$ and a public verification key $pk$.

$$(sk, pk) \leftarrow \text{KeyGen.DS}(1^n)$$

**Signing**

- The signing algorithm has a function Sign() that takes as input a message to be signed $m$ and a private signing key $sk$, and produces a signature on $m$, $\text{Sig}_m$.

$$\text{Sig}_m \leftarrow \text{Sign}_{sk}(m)$$

**Verifying**

- The verification algorithm has a function Verify() that takes as input the tuple $<m', \text{Sig}_{m'}>$ and a public verification key $pk$, and produces either 1 ("true") or 0 ("false").

$$1 \text{ or } 0 \leftarrow \text{Verify}_{pk}(\text{Sig}_{m'}, m')$$

# Digital Signature Scheme

- The correctness requirement of a digital signature is as follows
  - Given a key pair (*sk*, *pk*) and the signature $Sig_m \leftarrow Sign_{sk}(m)$, the verification algorithm $Verify_{pk}(Sig_m, m)$ will always output 1
  - In other words, a digital signature produced using the private key *sk* corresponding to the public key *pk* will always be successfully verified using the public key *pk*

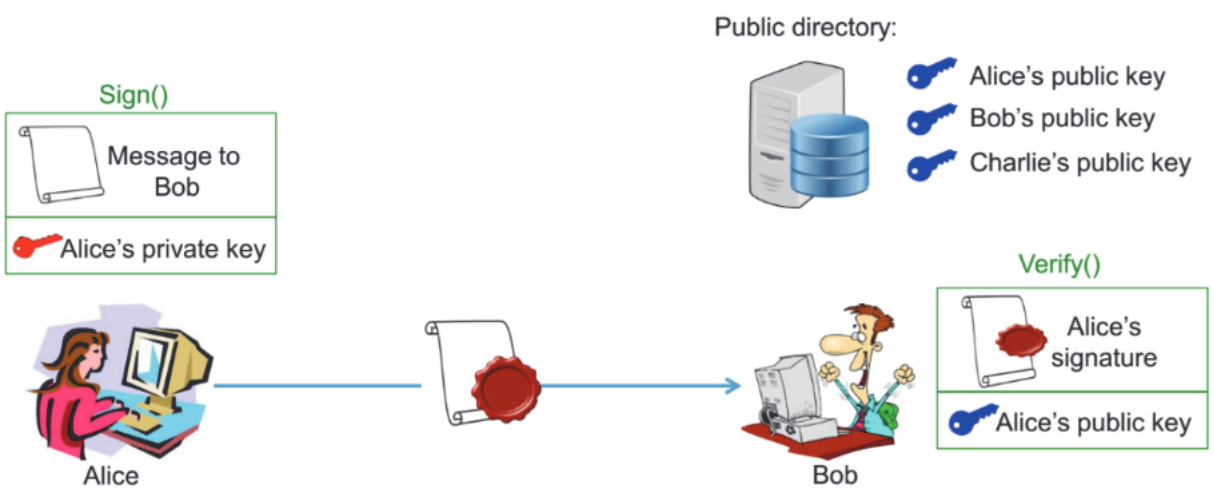



**Figure 1.9:** A graphical illustration of a digital signature scheme.

# RSA Digital Signature (Textbook)

- *Key Generation*:
  - On input of a security parameter *n*, the KeyGen.RSA($1^n$) function outputs a prime *p* and a prime *q* of *n* bits
  - Computes $N = pq$
  - An integer e that satisfies the relation gcd(e, $\varphi(N)$) = 1
  - An integer *d* that satisfies the relation $ed = 1 \bmod \varphi(N)$

$$(sk{=}d, pk = \langle N,e \rangle) \leftarrow \text{KeyGen.RSA}(1^n)$$

- *Signing*:
  - Sign.RSA() takes as input the integer *d* (the private signing key *sk*) and a message to be signed *m*

$$Sig_m \leftarrow m^d \bmod N$$

  - The message *m* is less than the value of *N* for the verification to succeed

# RSA Digital Signature (Textbook)

- *Verification*:
  - Verify.RSA() takes as input the tuple $\langle\ Sig_{m'}, m'\ \rangle$ and the integer e (the public verifying key *pk*), and produces 1 ("true") if and only if

$$Sig_{m'}^{e} \bmod N = m'$$

- The correctness requirement of the RSA digital signature scheme is

$$Sig_{m}^{e} \bmod N = (m^{d})^{e} \bmod N = m^{1+k\varphi(n)} \bmod N$$

$$= m^{1}\ m^{k\varphi(n)} \bmod N = m^{1} = m$$

- The security of the RSA digital signature is based on the RSA factorization problem
  - By referring to the Key Generation algorithm, the RSA factorization problem states that given a very large number $N = pq$, where $p$ and $q$ are both large prime numbers, there is no known efficient method to find out its two prime factors $p$ and $q$

# RSA Digital Signature (Textbook)

- The reason that this RSA digital signature scheme in this section is called a "textbook" scheme is because it is insecure for several reasons
  - In one form of attack known as a no-message attack, an attacker can arbitrarily choose a public key e of a legitimate user, generate a pseudorandom value as the signature *Sig*, compute $m = Sig^e \bmod N$ and produce the tuple $\langle Sig, m \rangle$ as a supposedly valid signature of the user on the message *m*
  - By definition of how the verification algorithm works, this tuple will also be verifiable using public key e of the victim
  - In this case, the attacker does not need to know the private key, and the message could just as well be gibberish

# RSA Digital Signature (Textbook)

- One solution to the vulnerability is to hash the message before signing it
- The signature would therefore be generated on the hash value rather than directly on the message
- In this case, the security of the solution now relies on the security of the underlying hash function:

$$Sig_m = Hash(m)^d \ mod \ N$$

- Due to property of the hash function, signing on hash value of a message is as good as signing on the message itself, as any changes to the message will result in a completely different hash value
  - Hash value is a compact representation of the message, thus allowing more efficient signature generation compared to operating directly on a potentially large message
  - Signing in the above manner produces a deterministic signature
- Actual implementation of the RSA digital signature is the RSA Probabilistic Signature Scheme (PSS), which introduces a certain amount of pseudo randomness to the message before signing it, so that a different signature will be generated each time

# RSA Digital Signature (Textbook)

```
b'\ruyY\xbczU\xb9E\xad\xcb[\xfc\xa8\x06\xdc\xba\x15\x90+0|%u\xc8U\x17j\x0b\x116iSH\xe4
\\\x1cb\xb6ip\xeah\xd8l\xb6rL\x83\x8f[\xc0F\xa1q\xc9\x11\x8f\x18YT\xe3^\xc7O\x9a\x8e0\
x03\xb1X}\xebw\xd8\xdd\x06\xbbGX#\xf5b\xd0\xd1hN\xe8\x8d/\xbby\x1a\x96\xca>\xebAg(\x1d
\xa4\xc2\xdc\xea\x1cE\xb9\x1b\xf3\xa69\x82\\\xe3K\xdf\xac\'\x8c\x84\xbd\xadU\xd3\xb4\x
e5\xc1^b8\x7f\xc1\xa4\x96\x93\xe4\xae\x85~[\xc9\x0cK\xb4\x92\xf2\x1d4\x98%\x0b\xeb\x97
:\xc5\x1f\xa3\x1c\x85S\x00X>\xf2h;\x1f\xff\x7f\xae\xe1\xa6\xab\xf7"\xa7\x0f\xd0JE\xcaG
\x18K\x15A\xd48)\xdd\xcd\xf4\xfe\xdb\xcd\xe7+.I\xf5X\xe3q\xb0\x8e3\x15\xa4\xd1\x94\x10
\x99\x18Z\xa8*%C\xa1\xdf\x81[\xa6|\xe2\x9cU(\x11\xfc#m\x9f1{\xb4s\x8c\xc6\xc9\xcdNo, a
\x9b\xda[\x98\xa8\x80\x93\xd0\xe3'
```

```
uyY◆zU◆E◆[◆◆+0|%u◆Uj6iSH◆\b◆ip◆h◆l◆rL◆[◆F◆q◆◆YT◆^◆O◆0◆X}◆w◆◆GX#◆b◆hN◆/◆y◆>◆Ag
(◆E◆◆9◆\◆K7◆U4◆^b8◆駁~[◆K◆4◆%◆:◆◆◆S◆X>◆h;◆◆6g◆"◆◆JE◆GKA◆8)◆+.I◆X◆q◆3◆є◆Z◆*%C◆Ɪ[◆|◆U
(◆#m◆1{◆s◆No, a◆[◆
```

**Figure 1.14:** The resulting RSA digital signature in bytecode (upper part) and UTF-8 encoding (lower part).

- A successful verification indicates that the signer has indeed created the message because no one else has access to the private key to create the digital signature for that message

# Elliptic Curve Cryptography

- Elliptic Curve Digital Signature Algorithm (ECDSA) is adopted in Bitcoin and widely used in the blockchain system
- It is the elliptic curve analogue of the Digital Signature Algorithm (DSA), based on the form of cryptography that is called the Elliptic Curve Cryptography (ECC)
- ECDSA can be considered as an alternative, with enhanced security, of the first-generation public key cryptography methods such as RSA

# ECDSA vs. RSA

- ECDSA serves as a next-generation public key algorithm compared to the earlier one such as the RSA
- ECC is more secure than the RSA
  - The probability of the ECC algorithm being solved or the risks of ECC being broken are much lower than that of the RSA
- The security of any public key algorithm relies on the size and algorithm, so that having a public key, one cannot find the private key
  - Otherwise, it is not safe
- The reverse direction is not impossible, just computationally infeasible, because the possibility is very slim given the extremely large number
- Breaking an RSA key requires one to factor a large number, but breaking an ECDSA key requires one to solve the Elliptic Curve Discrete Logarithm Problem (ECDLP)
- Using ECDSA will provide the same level of security as RSA but with smaller keys (160-bit key length in ECC compared with 1024-bit key length in RSA)
  - Benefits of faster signature generation, less data for the network and less computing power needed

# ECDSA vs. RSA

| Aspect | ECC | RSA |
|---|---|---|
| Time | ECC takes full exponential time | RSA takes sub-exponential time |
| Security and key size | Same level of security with smaller key sizes | Same level of security with larger key sizes |
| Data size | Smaller | Larger |
| Encrypted message | Function of key size and data size; encrypted message is smaller in ECC | Function of key size and data size; encrypted message is larger in RSA |
| Computational power | Smaller | Larger |

*Source*: Khalique *et al.* (2010).

# Pay to Public Key Hash (P2PKH)

- To illustrate how hash and digital signature are used in Bitcoin, we look at the process of a method called Pay to Public Key Hash, or P2PKH

| Field | | Description |
|---|---|---|
| ver | | Version number |
| vin_sz | | Number of inputs |
| vout_sz | | Number of outputs |
| lock_time | | The time to wait before the coins can be spent |
| size | | Size of the transaction in bytes |
| in | prev_out | The ID of the transaction containing the input |
| | scriptSig | The sender's public key and digital signature on this transaction |
| out | value | The amount to be transferred to the receiver |
| | scriptPubKey | The receiver's public key hash |

This transaction has a unique transaction hash value

**Figure 1.16:** Structure of a Bitcoin transaction.

# Pay to Public Key Hash (P2PKH)

- In Figure 1.16, each transaction may contain
  - More than one input (e.g., 1 btc and 2 btc as inputs to make up for a 3-btc spend) or
  - More than one output (e.g., 1 btc as input and 0.4 btc, 0.6 btc to more than one recipients) or
  - Multiple inputs and multiple outputs
- Each transaction will also have a unique transaction hash value based on its content
  - The hash value is recorded in the block Merkle Hash Tree

# Pay to Public Key Hash (P2PKH)

- Use a fiat transaction example to illustrate how the equivalent bitcoin transaction looks
- Suppose there are four users, Alice, Bob, Charlie and Eve, and the flow of their fiat transactions with each other is shown below

| | *Alice* | *Bob* | *Charlie* | *Eve* |
|---|---|---|---|---|
| *Starting balance* | $5 | - | $1 | $1 |
| *End-of-R1 balance* | - | → $5 | $1 | $1 |
| *End-of-R2 balance* | - | - | → $5 + $1 | $1 |
| *End-of-R3 balance* | $5 + $1 | ← - | - | $1 |
| *End-of-R4 balance* | $2 | - | - → | $4 + $1 |
| *End-of-R5 balance* | $1 + $1 | ← - | - | $5 |

# Pay to Public Key Hash (P2PKH)

- The equivalent bitcoin transactions' flow can be modelled as follows
- In Round 1 (R1), the transaction of 5 btc from Alice to Bob will be formatted as shown in Figure 1.17 (we omit other fields and the provision of transaction fees for simplicity)
  - Suppose this (Alice-to-Bob) transaction has a unique hash value of 7b844fe6a2ce9b1c7ea2f02bfb802a095ad3352a092ac83aef0562ee5952b1d7

| Field | | Description |
|---|---|---|
| vin_sz | | 1 |
| vout_sz | | 1 |
| in | prev_out | <A specific transaction hash> |
| | scriptSig | Alice's public key and digital signature |
| out | value | 5 btc |
| | scriptPubKey | Bob's public key hash |

Transaction hash:
7b844fe6a2ce9b
1c7ea2f02bfb802
a095ad3352a092
ac83aef0562ee5
952b1d7

**Figure 1.17:** Structure of the Alice-to-Bob transaction.

# Pay to Public Key Hash (P2PKH)

- In R2, as Bob sends the 5 btc he has received from Alice to Charlie, he would formulate his transaction as shown in Figure 1.18
- As Bob intends to spend the 5 btc he has gotten from Alice, he must specify that the input to this (Bob-to-Charlie) transaction comes from a previous (i.e., Alice-to-Bob) transaction with the hash value 7b844fe6a2ce9b1c7ea2f02bfb802a095ad3352a092ac83aef0562ee5952b1d7

| Field | | Description |
|---|---|---|
| vin_sz | | 1 |
| vout_sz | | 1 |
| in | prev_out | 7b844fe6a2ce9b1c7ea2f02bfb802a095ad3352a092ac83aef0562ee5952b1d7 |
| | scriptSig | Bob's public key and digital signature |
| out | value | 5 btc |
| | scriptPubKey | Charlie's public key hash |

Transaction hash: 317b9591b0a9d74afacd5735812d236681e5111982d2d57be21a598ad1cba628

**Figure 1.18:** Structure of the Bob-to-Charlie transaction.

# Pay to Public Key Hash (P2PKH)

- In R3, Charlie's transaction to Alice contains two inputs (5 btc and 1 btc), of which the 5-btc input comes from the previous Bob-to-Charlie transaction
- Charlie's transaction is formulated as shown in Figure1.19

| Field | | Description |
|---|---|---|
| vin_sz | | 2 |
| vout_sz | | 1 |
| in0 | prev_out0 | 317b9591b0a9d74afacd5735812d236681e5111982d2d57be21a598ad1cba628 |
| | scriptSig0 | Charlie's public key and digital signature |
| in1 | prev_out1 | <A specific transaction hash> |
| | scriptSig1 | Charlie's public key and digital signature |
| out | value | 6 btc |
| | scriptPubKey | Alice's public key hash |

Transaction hash: d7a7e3cd01e4731dacd10bd20c218acebdcd2ea3d271c1f5a46b886c2698cd92

**Figure 1.19:** Structure of the Charlie-to-Alice transaction.

# Pay to Public Key Hash (P2PKH)

- In R4, Alice intends to pay Eve 4 btc
- However, in contrast to fiat currency transactions, Alice has in her wallet 6 btc instead of 5 btc +1 btc due to aggregation
- Thus, in her transaction, Alice has to specify that she intends to pay Eve 4 btc, and pay herself 2 btc
  - The concept of "change" in bitcoin (see Figure 1.20)

| Field | | Description |
|---|---|---|
| vin_sz | | 1 |
| vout_sz | | 2 |
| in | prev_out | d7a7e3cd01e4731dacd10bd20c218acebdcd2 ea3d271c1f5a46b886c2698cd92 |
| | scriptSig | Alice's public key and digital signature |
| out0 | value0 | 4 btc |
| | scriptPubKey0 | Eve's public key hash |
| out1 | value1 | 2 btc |
| | scriptPubKey1 | Alice's public key hash |

Transaction hash: be3c7b2c461219 050cc716717778 c552c3e57841b2 26b89912a33ee4 b897fe4b

**Figure 1.20:** Structure of the Alice-to-Eve transaction.

# Pay to Public Key Hash (P2PKH)

- Notice a few things in the example above
  - A bitcoin transaction can contain one or more inputs and one or more outputs
  - Once a previous transaction (e.g., Alice-to-Bob) has been recorded on the blockchain, it effectively transfers the ownership of the 5 btc from Alice to Bob
  - When Bob intends to spend the 5 btc he has gotten from Alice, he must specify the source of the 5 btc in the "prev_out" field and provide his proof of ownership of the 5 btc by stating his public key and digital signature on this transaction

# Pay to Public Key Hash (P2PKH)

- In order for miners to validate whether Bob has ownership of the 5 btc, they will do the following checks:
  1. Extract Bob's public key from the "scriptSig" field in the Bob-to-Charlie transaction
  2. Calculate the hash of Bob's public key obtained in Step 1
  3. Find the transaction specified in the "prev_out" field in the Bob-to-Charlie transaction, i.e., the Alice-to-Bob transaction
  4. Verify that the hash calculated in Step 2 is equal to the recipient's public key hash specified in the "scriptPubKey" field of the Alice-to-Bob transaction
  5. If they are equal, use the public key obtained in Step1 to verify Bob's digital signature in the "scriptSig" field for the Bob-to-Charlie transaction
  6. If verification succeeds, conclude that Bob has ownership of the 5 btc and thus can spend the 5 btc
  7. Add the Bob-to-Charlie transaction to a block, and run the PoW algorithm

# Pay to Public Key Hash (P2PKH)

- The fourth point to note in this process is that Bob's public key is not disclosed until he creates the Bob-to-Charlie transaction and announces his intention to spend the 5 btc in his wallet
- The usage of public key hash to specify the receiver utilizes properties of the hash function
  - An attacker could not pretend to be Bob because the public key hash reveals nothing about the pre-image
- Only Bob can provide the correct public key that hashes to the public key hash specified by Alice
- Finally, once the miners verified that the hash of Bob's public key is equal to the public key hash specified by Alice, the miners will use Bob's public key to verify his digital signature on the Bob-to-Charlie transaction
  - When the verification succeeds, miners conclude that Bob has ownership of the 5 btc in the wallet

# Pay to Public Key Hash (P2PKH)

- The relationship between a user's private key, public key and wallet address
- For security purposes, private keys are truly random strings
- From the private key, a user's public key is calculated using a one-way (irreversible) and predetermined manner
- The public key is then hashed and encoded to form the user's bitcoin wallet address
- As such, the conclusion that "Bob has ownership of the wallet containing the 5 btc if his digital signature on the transaction can be verified using his public key" is based on the fact that Bob can only provide the (correct) public key if and only if he has the private key to create the digital signature
- Consequently, only Bob can give Alice the (correct) public key hash
- The act of specifying the receiver's public key hash in a bitcoin transaction is a mechanism called "Pay to Public Key Hash (P2PKH)"

# Public Key Infrastructure

- In a digital signature, the signature and message are verified using the sender's (i.e., signer) public key to prove that the message originates from the signer
- A public key is widely disseminated and is a pseudorandom number that by itself does not contain any identifiable information about its owner
- A potential security loophole — how can a verifier be sure that the public key belongs to the right signer?
- Without a method of verification, an attacker could exploit this loophole by intercepting the public key and replacing it with his own, thereby compromising the objective of data authentication and accountability in the digital signature
- There is a need for us to ensure that a public key belongs to a specific user
- Public key infrastructure (PKI) solves this problem by providing a method for associating a public key with a specific user in a secure manner

# Public Key Infrastructure

- The PKI framework comprises certificates, certification authorities and certificate revocation

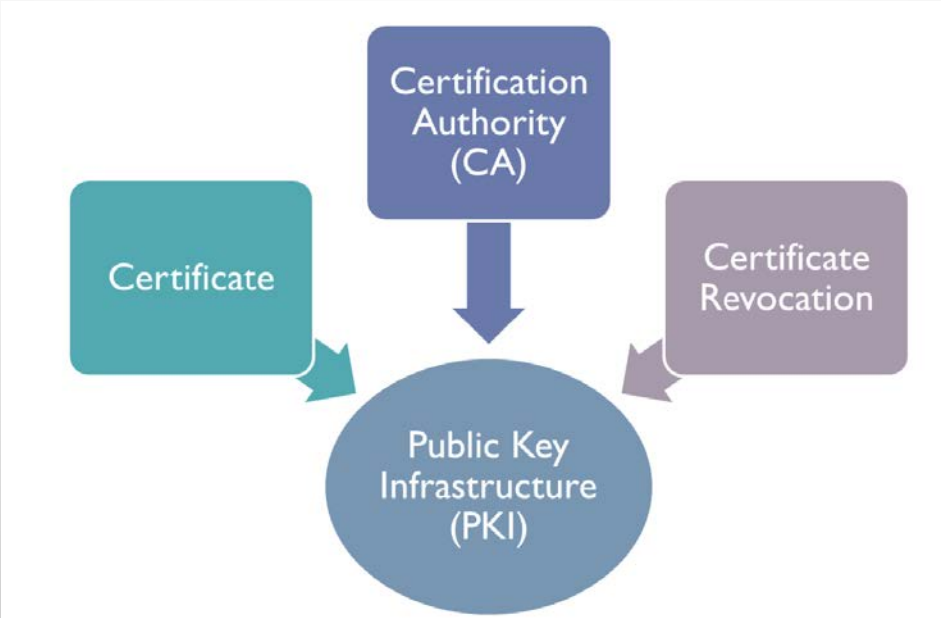



**Figure 1.21:** Components of a public key infrastructure.

# Public Key Infrastructure

- A *certificate* is a digital document that serves as proof that a particular public key belongs to a specific user (the subject)
  - Include version number, serial number, type of digital signature scheme used, issuer's name, validity period, subject's name, subject's public key and issuer's signature
- Aside from users, certificates can also state a website's information
  - Ex: Facebook's certificate

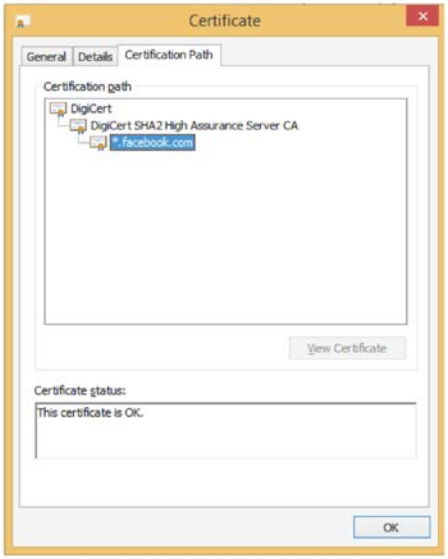



**Figure 1.22:** Example of certification path for facebook.com.

# Public Key Infrastructure

- To obtain a digital certificate, we must approach a third-party *certification authority* (CA, also known as the issuer)
- Certification takes the form of the CA's digital signature appended at the end of the certificate
- CAs are usually organized according to geographical distribution
  - For example, Netrust is a CA that covers the Southeast Asia region and is the only accredited CA in Singapore
  - We know which CAs to trust as one CA vouches for another
  - This takes the form of a hierarchical relationship, with a root CA at the top vouching for intermediate CAs that issue certificates to end users

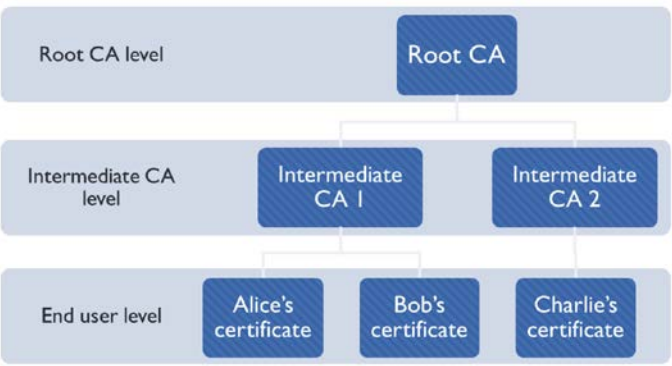



**Figure 1.23:** The certification path for facebook.com.

# Public Key Infrastructure

- A list of such root CAs is hard-coded into our browsers
- Whenever we install a browser and visit a website, our browsers trace the website's certification path back to the root CA
  - Examples of root CAs include Comodo, Symantec, GoDaddy, GlobalSign, DigiCert and Verizon
- Certificates are revoked when they expire, if the user's credentials change or if the user's private key is lost or compromised
- To avoid inadvertently sending confidential messages to a compromised user, we maintain repositories of revoked certificates to perform checks on the status of a certificate
  - One way to do this is for root CAs to maintain a certificate revocation list (CRL) that users can check against
  - However, over time, the CRL may grow to such a magnitude that it becomes either difficult to maintain or inefficient to perform a search
  - Alternatively, the online certificate status protocol (OCSP) uses IP to obtain the status of a certificate
    - This is a much more efficient method that is supported in most browsers used today

# Public Key Infrastructure

- The PKI is a setting in which we can ensure user authentication and build trusted, secure communication networks
- Its usage can be found in our daily applications as well as private blockchain settings such as in Project Ubin
- In public blockchain, however, a public key is not associated with an identity in the conventional sense
- The public blockchain is typically pseudonymous, meaning that it provides some, but not full, anonymity
  - In the Bitcoin blockchain, all transactions are linked to public key hashes, which on their own do not provide any identifying information pertaining to the wallet's owner
  - That said, it can be possible to prove a user's ownership of a wallet given certain information, such as her possession of a private key corresponding to the public wallet address

# Public Key Infrastructure

- In the Bitcoin blockchain, the usage of public and private keys to ensure bitcoin ownership in a decentralized manner is executed through methods such as Pay-to-Public-Key-Hash (P2PKH), which relies on the security of hash function and digital signature
- Double spending is prevented with a global view of the transaction hashes on the distributed ledger

# Privacy

- User privacy in a blockchain rests on the untraceability and unlinkability of transactions
- To recap, untraceability means we should be able to conceal the details of a transaction so that an observer cannot follow the trail
- Unlinkability, in general, means that two events occurring under the observation of an attacker should appear unrelated to the observer
- Earlier, we said that blockchain technology is pseudonymous, as opposed to anonymous
  - By simply observing the blockchain, there is no identifying information linked to the wallet addresses or transactions
  - However, we are able to trace the source of every transaction, and therefore, able to obtain a graphical visualization of transaction paths
  - Through prior knowledge or social engineering, it is possible to link an identity to a wallet address
  - We need security schemes in place to protect user privacy

# Privacy

- Suppose an attacker observes the following transaction traces at time *t*

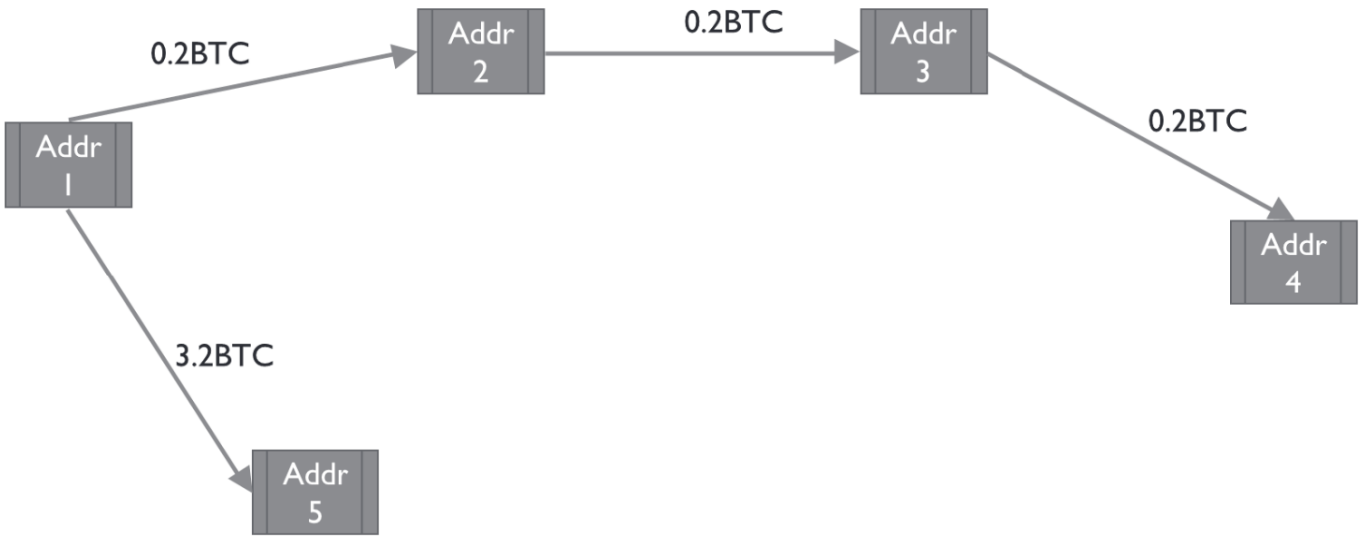



**Figure 1.33:** Trace of transactions that can be obtained from a public blockchain.

# Privacy

- At time $t+1$, a new transaction takes place

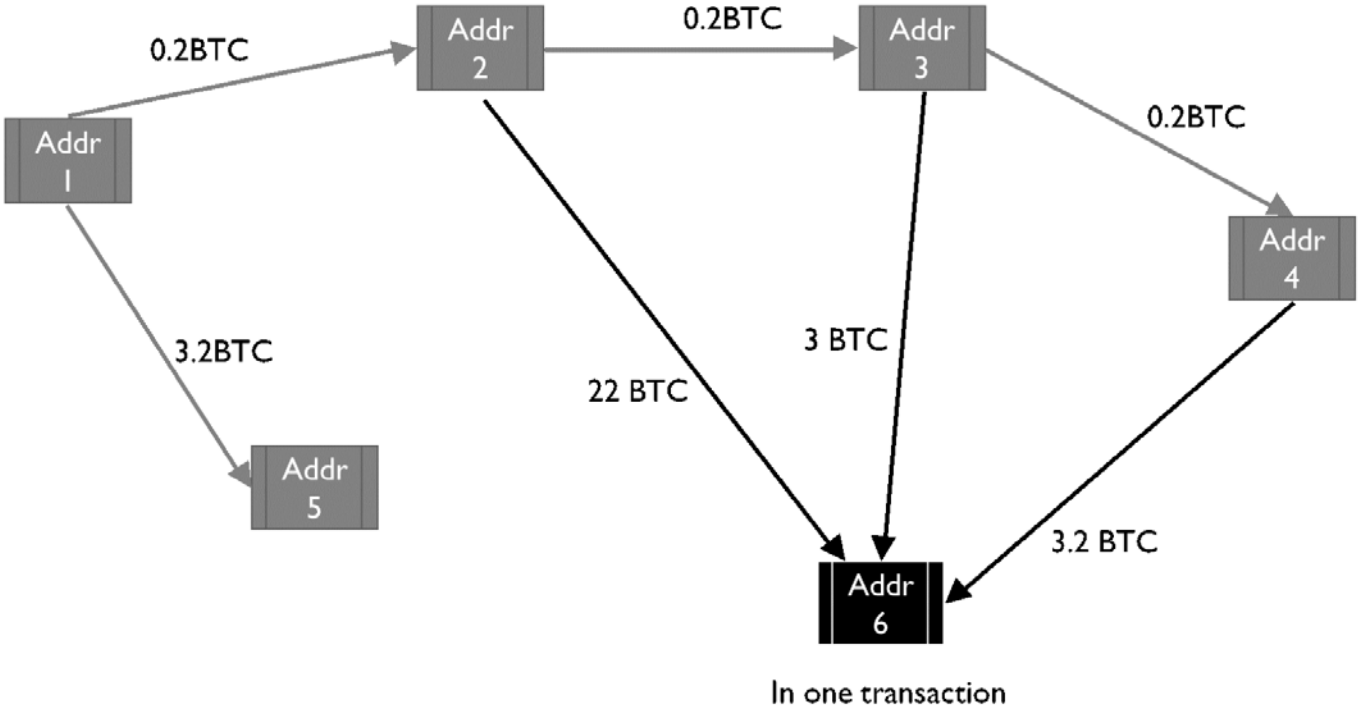



**Figure 1.34:** A new transaction that takes user inputs from three different addresses may reveal user private information.

# Privacy

- An observer can deduce, with certain probability, that addresses 2, 3 and 4 belong to the same user
- In fact, such an attack is similar to a dusting attack, where an attacker sends users tiny amounts of cryptocurrencies to their wallets
- After that, the attacker performs analysis of these wallet addresses in an attempt to identify which ones belong to the same user

# Privacy

- Understanding user privacy as a form of confidentiality might lead us to consider encryption as a solution
- The two main types of data in a Bitcoin transaction are the transaction amount and the addresses of the sender and receiver
- Immediately, some problems are evident with encrypting this information
  - Encryption alone does not allow blockchain to function properly or securely as a decentralized P2P payment system, and encryption alone does not ensure untraceability and unlinkability
  - For example, encrypting transaction data or wallet addresses does not provide untraceability if the same wallet is reused
  - Furthermore, encrypting transaction data and wallet addresses without providing additional information will impede miners in the effort of validating transactions (e.g., check for double-spending and proof of ownership)

# Privacy

- Blockchain developers have managed to develop algorithms that ensure user privacy while addressing these concerns
  - Some possible methods: CoinJoin, stealth addresses, ring signature and Ring Confidential Transaction (RingCT) in Monero and zero-knowledge proofs

# CoinJoin

- A trustless method that allows multiple bitcoin spenders to come together and mix their input and output, so that observers cannot easily trace how they are spending their bitcoin
- Suppose Alice has received 2 btc from Bob, which she now wants to spend
- But, Bob can easily track the 2 btc to see which addresses Alice pays the amount to
- For privacy, Alice decides to spend her 2 btc in a CoinJoin transaction
- Alice finds at least one other user who agrees to spend the common denomination of 2 btc
- Among them, they identify a user who will facilitate the transaction
- They then provide their respective input of 2 btc to the facilitator, as well as their respective public key hashes specifying where their output should go
- All the users then sign the transaction and broadcast it to the network

# CoinJoin

- This single transaction spends the agreed common denomination of bitcoin from each user at one go
- Purely by observing the CoinJoin transaction, other users, including Bob, will not be able to identify which particular public key hash belongs to Alice
- Bob will lose trace of Alice's spending behaviour
- In effect, CoinJoin employs a mixing technique that gives users plausible deniability regarding how they spend their bitcoin

# CoinJoin

- The more users are involved in the CoinJoin transaction, the more difficult it will be to trace the transactions of any one user
- This method depends on a mixture and obfuscation of data, but does not involve additional cryptographic techniques
- It therefore retains certain vulnerabilities
- CoinJoin does not preclude any of the users involved in the transaction from announcing to the network which public key hash belongs to them, thereby making it easier for observers to deduce the public key hashes of other users in the transaction
- Users involved in the CoinJoin transaction will know which public key hashes belong to the other users
- The solution to this is CoinShuffle, a method for encrypting public key hashes

# Zero-Knowledge Proofs

- Zcash is the first public blockchain to implement a full privacy protection scheme on transactions
- It encrypts all transaction data, and uses a zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) algorithm to provide user anonymity and transaction traceability and unlinkability in a more efficient manner than the traditional zero-knowledge proof
- Traditional design of a zero-knowledge proof is interactive and requires multiple iterations to be convincing
- The non-interactive zk-SNARKs algorithm used in Zcash is more succinct in terms of proof size and efficient in terms of verification time

# Zero-Knowledge Proofs

- Typically, when we want to convince someone that we possess knowledge of a secret statement, such as a password, we do so by exchanging the statement with a verifying authority or user who also possesses knowledge of it
- But, what if the user who is verifying the information does not — and should not, for privacy — possess knowledge of the secret statement itself?
- Zero-knowledge proofs allow us to prove to a verifier that a secret statement we hold is true, without actually revealing any information about that statement to the verifier

# Zero-Knowledge Proofs

- Such proofs have three properties:
  1. *Completeness*. If the prover is honest, the prover will eventually convince the verifier
  2. *Soundness*. The prover can only convince the verifier if the statement is true
  3. *Zero-knowledge*. The verifier learns nothing about the statement aside from the fact that it is true
- We proceed with a high-level overview of Zcash based on the Zcash protocol specifications (Hopwood *et al.*, 2019)
- The value of a Zcash transaction can be either transparent or shielded
- Transparent transfers are essentially the same as Bitcoin and shielded transfer is the key differentiating factor for Zcash
- There are two types of addresses in Zcash, namely private (z-address) or transparent (t-address)
  - Transactions can be between two t-addresses, two z-addresses, or between a z-address and a t-address

# Zero-Knowledge Proofs

- Transactions between two t-addresses are essentially the same as Bitcoin transactions where the sender, receiver and transaction amount are public information
- Transactions between two z-addresses are private, where the sender, receiver and transaction amount are encrypted
- A transaction from a t-address to a z-address is a shielding transaction, whereas a transaction from a z-address to a t-address is a deshielding transaction
- In Zcash, each user possesses the keys

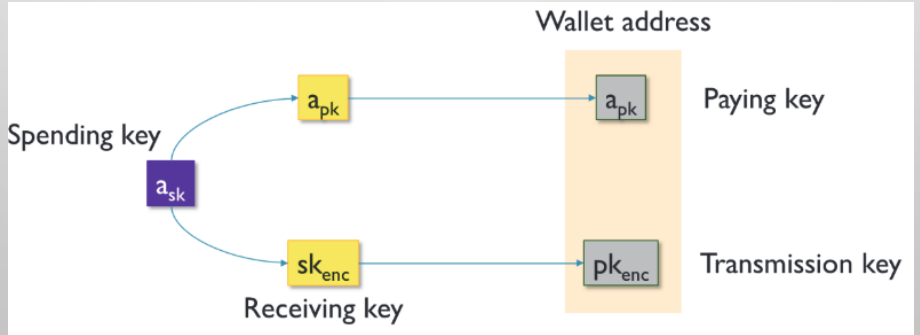



**Figure 1.35:** Keys in Zcash.

# Zero-Knowledge Proofs

- In Zcash, the concept of JoinSplit Description is discussed
- A JoinSplit Description contains, among others, one or more encrypted Notes for shielded transfers and values for transparent transfers (Figure 1.36)

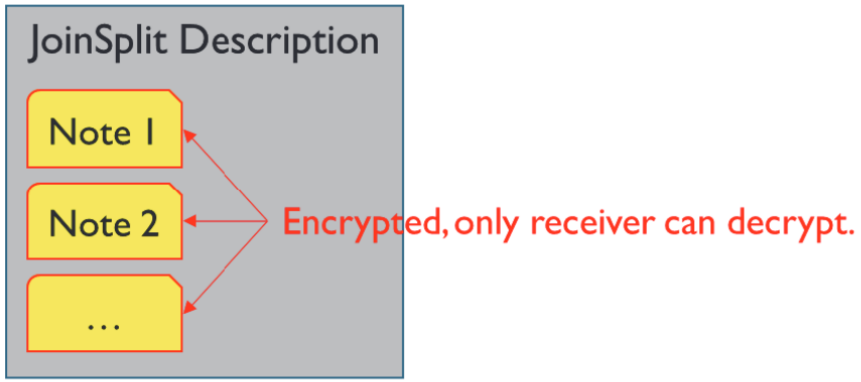



**Figure 1.36:** Illustration of a JoinSplit Description containing encrypted Notes.

# Zero-Knowledge Proofs

- It also contains nullifiers and commitments for shielded transfers, an ephemeral public key to derive encryption key used to decrypt the Notes, and zero-knowledge proofs for all necessary components
- Shielded transfers are carried in encrypted Notes
- Each Note contains the recipient's paying key $a_{pk}$, the value of transfer *v*, a $\rho$ to be used by the recipient to compute the nullifier when spending the Note and a commitment trapdoor *rcm*
- When a sender, e.g., Alice, creates a transaction to Bob, the output of this transaction must be accompanied by a commitment
  - Commitment is calculated as Hash(10110000|| $a_{pk}$ ||*v*||$\rho$||*rcm*), where $a_{pk}$ belongs to Bob and || denotes concatenation
  - This commitment is also recorded on a Merkle Hash Tree

# Zero-Knowledge Proofs

- As Bob intends to spend the output given to him by Alice, at which the output is now the input to Bob's transaction, Bob would need to prove that the commitment of the input exists in the Merkle Hash Tree without revealing which commitment it is
- He would also need to calculate a nullifier for the input
  - The nullifier is calculated as Hash(1110|| $a_{sk}$ ||$\rho$)
  - This nullifier is added to the nullifier set and used by the miners to verify if there is a double-spending
  - An input where the nullifier exists in the nullifier set is a double-spending
  - Notice that because the Note is encrypted such that only the intended recipient can decrypt, only the recipient will be able to obtain $\rho$ to calculate the nullifier
- Encryption of Notes is performed using an ephemeral secret key that only the receiver can derive
- Zcash uses a key agreement protocol that allows only the sender and the intended receiver to agree on a common secret without pre-sharing a secret key

# Zero-Knowledge Proofs

- With the above operations in mind, zk-SNARK is used in Zcash for a sender of a transaction to prove that, among others,
  - For the input — a Note commitment exists and the Merkle Path is valid; the nullifier is computed in a correct manner so that double-spending can be accurately detected
  - For the output — the Note commitment is computed correctly, and the nullifier is unique

# Zero-Knowledge Proofs

- The proof using zk-SNARKs is beyond the scope of this chapter
- It involves a multitude of techniques including homomorphic encryption, arithmetic circuit constructions to model cryptographic functions and secure creation of common reference string
- Refer to a step-by-step introduction by the Zcash team (Electronic Coin Company, 2020) and the Zcash protocol specifications (Hopwood *et al.*, 2019) for more details