UNDERSTANDING BLOCKCHAIN WITH CRYPTOCURRENCY

Siva Kumar Ronanki

Syllabus Topics

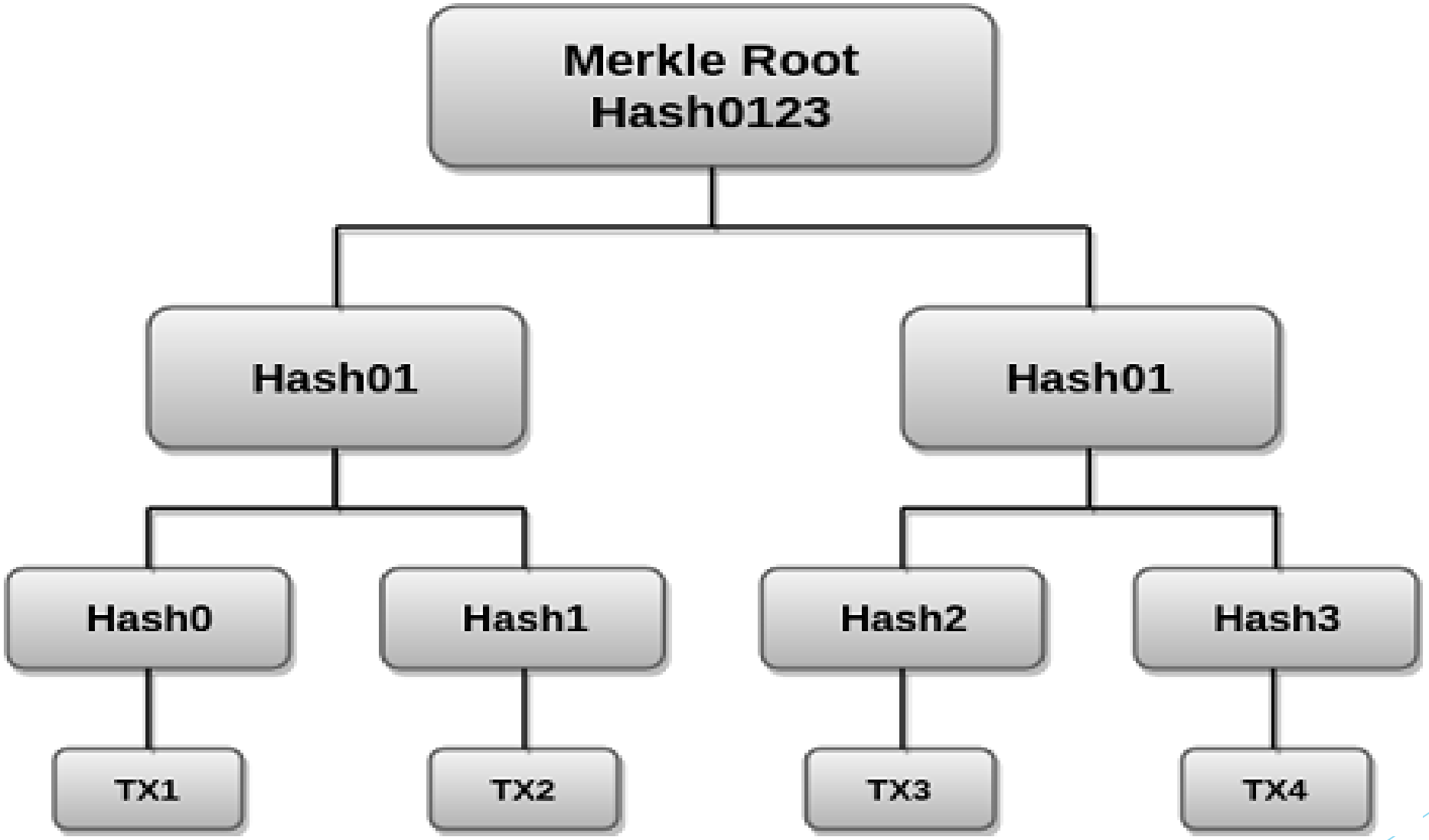
- Cryptographic Primitives: Merkle Tree, Patricia Tree, Digital Signatures, Elliptic Curve Digital signature algorithm (ECDSA).
- Bitcoin : Bitcoin, Bitcoin definition,, The transaction life cycle, The transaction structure, Types of transaction ,UTXO.
- Structure of Bitcoin : The structure of a block, The structure of a block header, The genesis block.

Merkle trees

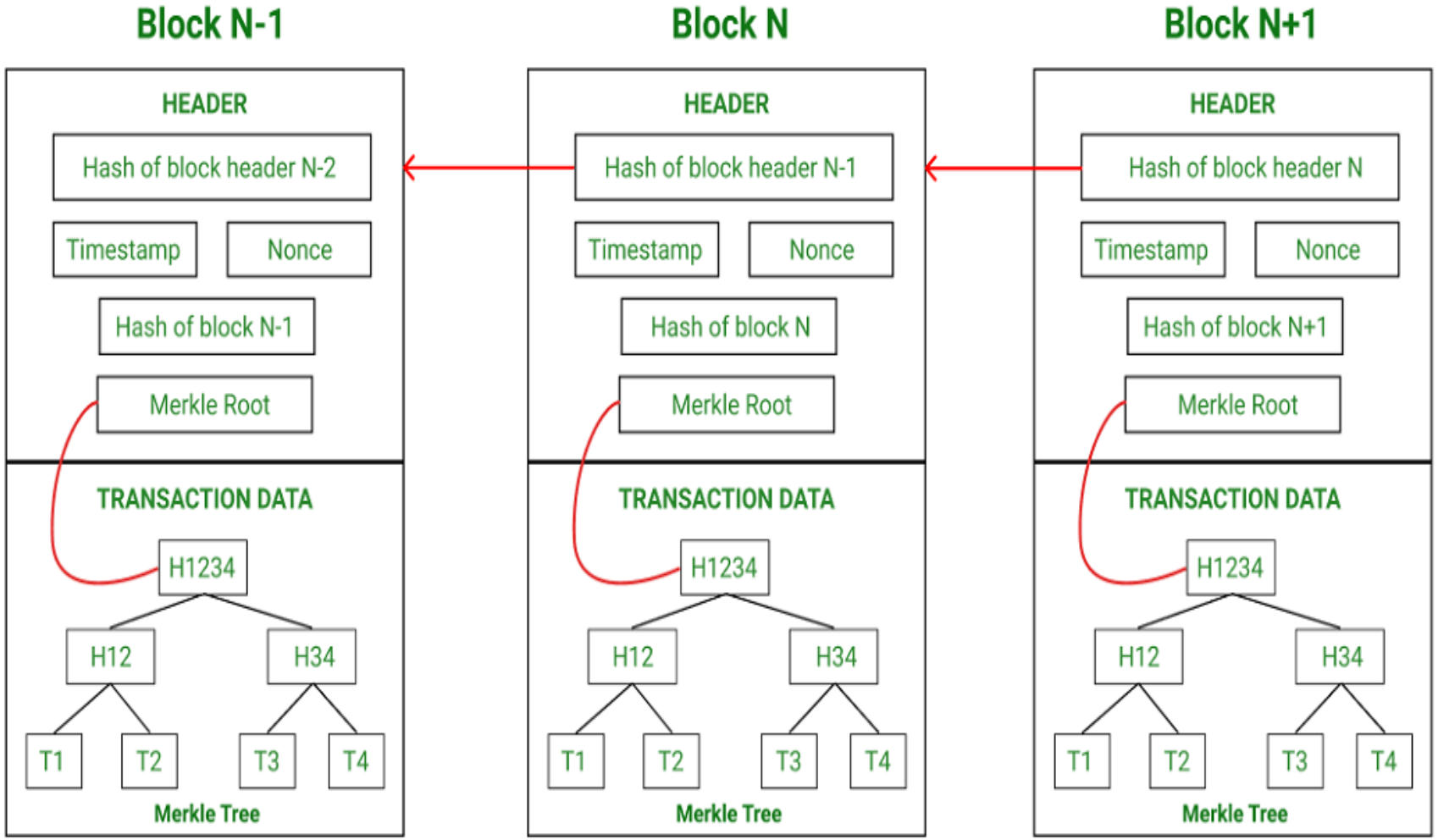
- Merkle tree is a fundamental part of blockchain technology. It is a mathematical **data structure** composed of hashes of different blocks of data, and which serves as a summary of all the transactions in a block.
- It also allows for efficient and secure verification of content in a large body of data. It also helps to verify the consistency and content of the data. Both Bitcoin and Ethereum use Merkle Trees structure. Merkle Tree is also known as **Hash Tree**.
- The concept of Merkle Tree is named after **Ralph Merkle**, who patented the idea in **1979**. Fundamentally, it is a data structure tree in which every **leaf node** labelled with the hash of a data block, and the **non-leaf node** labelled with the cryptographic hash of the labels of its child nodes. The leaf nodes are the lowest node in the tree.

How do Merkle trees work?

- A Merkle tree stores all the transactions in a block by producing a digital fingerprint of the entire set of transactions. It allows the user to verify whether a transaction can be included in a block or not.
- Merkle trees are created by repeatedly calculating hashing pairs of nodes until there is only one hash left. This hash is called the Merkle Root, or the Root Hash. The Merkle Trees are constructed in a bottom-up approach.
- Every leaf node is a hash of transactional data, and the non-leaf node is a hash of its previous hashes. Merkle trees are in a binary tree, so it requires an even number of leaf nodes. If there is an odd number of transactions, the last hash will be duplicated once to create an even number of leaf nodes.


Merkle Root
Hash0123
Hash01
Hash01
Hash0
Hash1
Hash2
Hash3
TX1
TX2
TX3
TX4

- Merkle Root is stored in the **block header**. The block header is the part of the bitcoin block which gets hash in the process of mining. It contains the hash of the last block, a Nonce, and the Root Hash of all the transactions in the current block in a Merkle Tree.
- So having the Merkle root in block header makes the transaction **tamper-proof**. As this Root Hash includes the hashes of all the transactions within the block, these transactions may result in saving the disk space.


Block N-1
HEADER
Hash of block header N-2
Timestamp
Nonce
Hash of block N-1
Merkle Root
TRANSACTION DATA
H1234
H12
H34
T1
T2
T3
T4
Merkle Tree
Block N
HEADER
Hash of block header N-1
Timestamp
Nonce
Hash of block N
Merkle Root
TRANSACTION DATA
H1234
H12
H34
T1
T2
T3
T4
Merkle Tree
Block N+1
HEADER
Hash of block header N
Timestamp
Nonce
Hash of block N+1
Merkle Root
TRANSACTION DATA
H1234
H12
H34
T1
T2
T3
T4
Merkle Tree

- The Merkle Tree maintains the **integrity** of the data. If any single detail of transactions or order of the transaction's changes, then these changes reflected in the hash of that transaction.
- This change would cascade up the Merkle Tree to the Merkle Root, changing the value of the Merkle root and thus invalidating the block. So everyone can see that Merkle tree allows for a quick and simple test of whether a specific transaction is included in the set or not.

Merkle trees have three benefits:

- It provides a means to maintain the integrity and validity of data.
- It helps in saving the memory or disk space as the proofs, computationally easy and fast.
- Their proofs and management require tiny amounts of information to be transmitted across networks.

Use-Cases of Merkle Tree

It is used in Git to handle projects by programmers from all around the world.

It's also open-source and implements Merkle Tree to allow computers to join and use a centralized file system.

It's part of the technique that generates verifiable certificate transparency logs.

amazon DynamoDB

During the data replication process, they are used by these No-SQL distributed databases to control discrepancies.

Patricia Trees

- **Practical Algorithm to Retrieve Information Coded in Alphanumeric (Patricia)**, also known as Radix tree.
- An extension to the binary trie where the branch node and element node are combined to form a new node(augmented node).
- Applications : NLP, Search engines.
- Merkle-Patricia tree, based on the definitions of Patricia and Merkle, is a tree that has a root node that contains the hash value of the entire data structure.

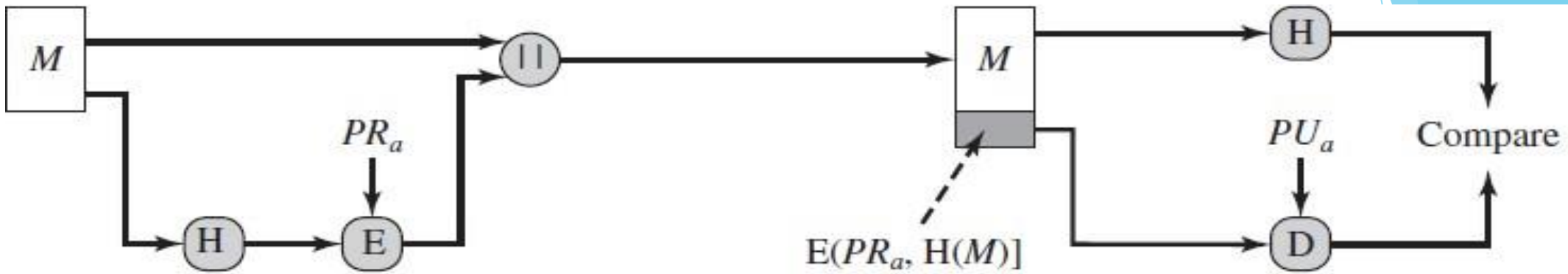
Patricia Trees Algorithm

- Perform the search on the key suppose search terminates at **S**, then create a node with key & assign bit number(j) to it which is equal to 1st bit in which **S** & key differ.
- Now assign perform search on the key with j-1 bits.
- Suppose the search stops at **S** after making a move from **P** to **S.**
- Then insert key as a child of **P.**
- If j^{th} bit of key is 1,then it's right pointer will be self pointer. Else if j^{th} bit is 0,then it's left pointer will be self pointer and the other pointer points to **S.**

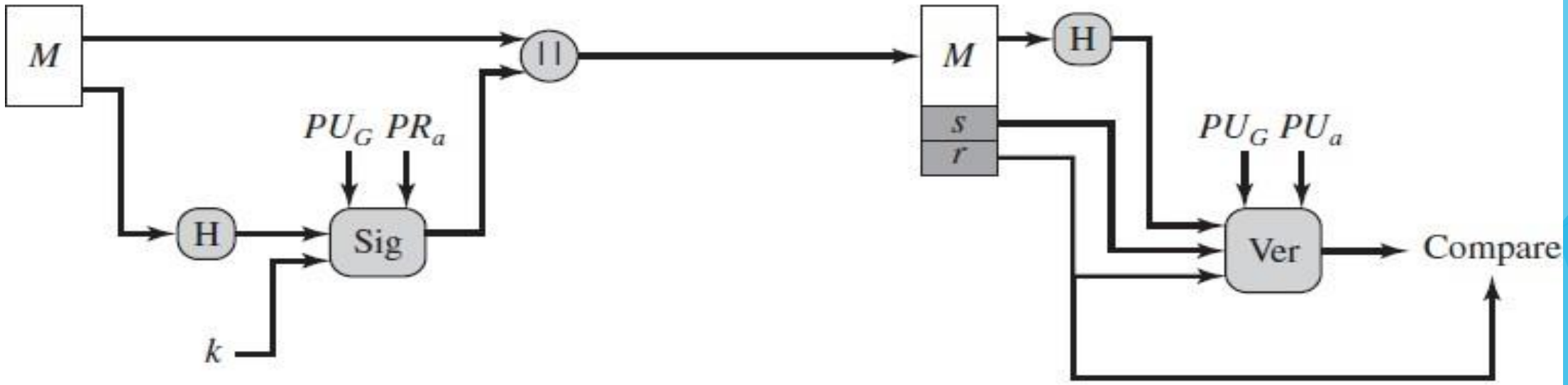
Digital signatures

- Digital signatures provide a means of associating a message with an entity from which the message has been originated. Digital signatures are used to provide data origin authentication and nonrepudiation. They are calculated in two steps. High-level steps of an **RSA digital signature scheme** is given as follows:
- 1. Calculate the hash value of the data packet. This will provide the data integrity guarantee as hash can be computed at the receiver's end again and matched with the original hash to check whether the data has been modified in transit. Technically, message signing can work without hashing the data first, but is not considered secure.
- 2. The second step signs the hash value with the signer's private key. As only the singer has the private key, the authenticity of the signature and the signed data is ensured.

- Digital signatures have some important properties, such as **authenticity**, **unforgeability**, and **non reusability**.
- Authenticity means that the digital signatures are verifiable by a receiving party.
- The unforgeability property ensures that only the sender of the message is able to use the signing functionality using the private key. In other words, no one else should be able to produce the signed message that has been produced by the legitimate sender.
- Non reusability means that the digital signature cannot be separated from a message and used for another message again. The operation of a generic digital signature function is shown in the following diagram:



(a) RSA approach



(b) DSA approach

Figure 13.3 Two Approaches to Digital Signatures

Global Public-Key Components

p prime number where $2^{L-1} < p < 2^L$
for $512 \leq L \leq 1024$ and L a multiple of 64;
i.e., bit length of between 512 and 1024 bits
in increments of 64 bits

q prime divisor of $(p - 1)$, where $2^{N-1} < q < 2^N$
i.e., bit length of N bits

g $= h(p - 1)/q \bmod p$,
where h is any integer with $1 < h < (p - 1)$
such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

x random or pseudorandom integer with $0 < x < q$

User's Public Key

$y = g^x \bmod p$

User's Per-Message Secret Number

k random or pseudorandom integer with $0 < k < q$

Signing

$r = (g^k \bmod p) \bmod q$

$s = [k^{-1} (H(M) + xr)] \bmod q$

Signature $= (r, s)$

Verifying

$w = (s')^{-1} \bmod q$

$u_1 = [H(M')w] \bmod q$

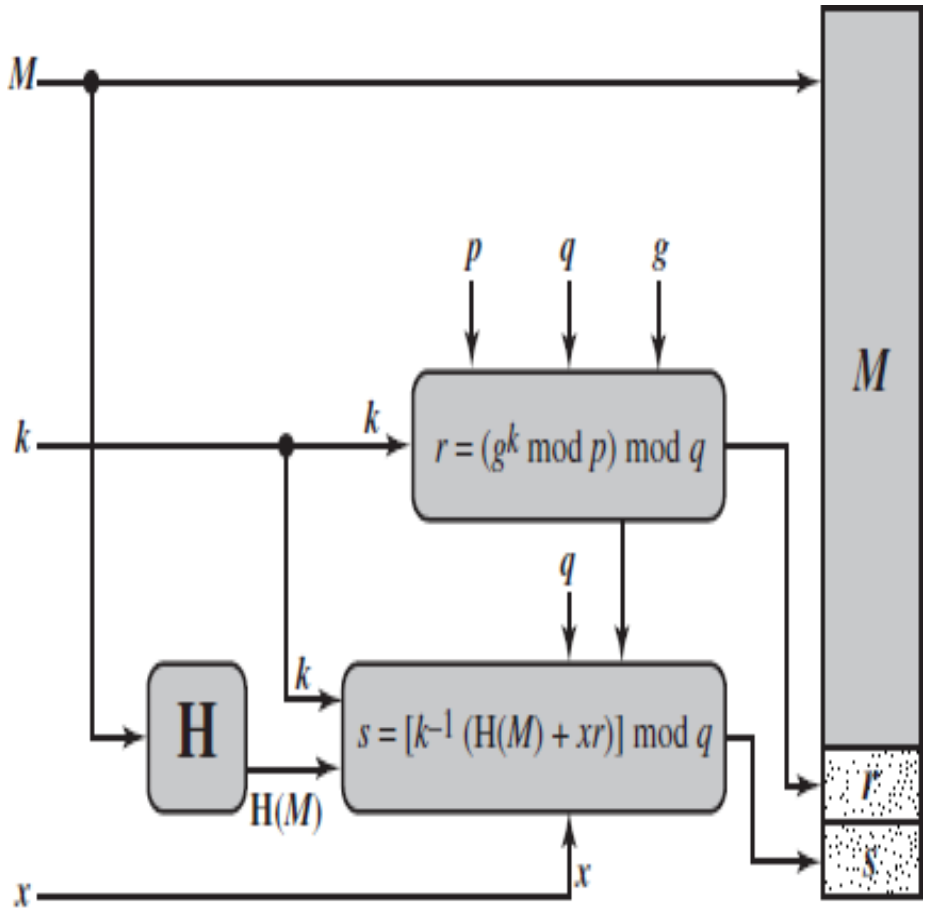
$u_2 = (r')w \bmod q$

$v = [(g^{u1} y^{u2}) \bmod p] \bmod q$

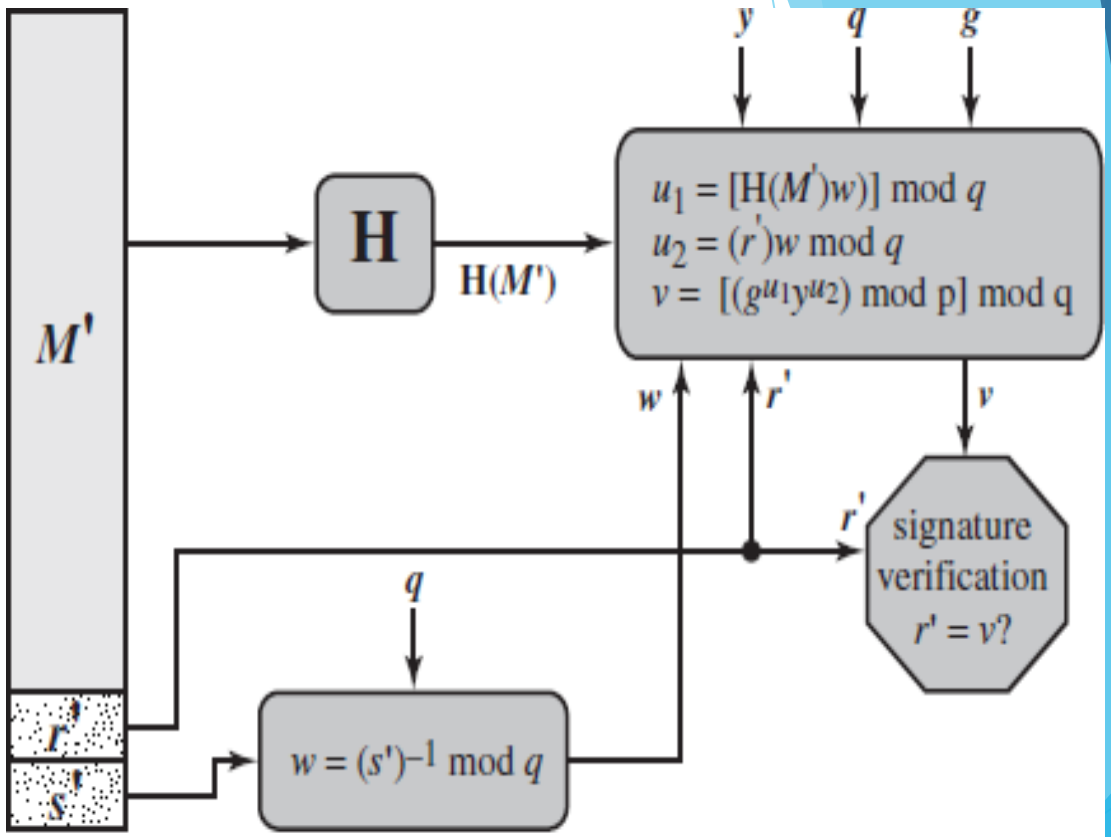
TEST: $v = r'$

M = message to be signed
H(M) = hash of M using SHA-1
M', r', s' = received versions of M, r, s

Figure 13.4 The Digital Signature Algorithm (DSA)



(a) Signing



(b) Verifying

Figure 13.5 DSA Signing and Verifying

- Digital signing (left) and verification process (right) (Example of RSA digital signatures) : If a sender wants to send an authenticated message to a receiver, there are two methods that can be used. These two approaches to use digital signatures with encryption are introduced here.
- **Sign then encrypt** : In this approach, the sender digitally signs the data using the private key, appends the signature to the data, and then encrypts the data and the digital signature using the receiver's public key. This is considered a more secure scheme as compared to the encrypt then sign scheme described next.
- **Encrypt then sign:** In this approach, the sender encrypts the data using the receiver's public key and then digitally signs the encrypted data. Note In practice, a digital certificate that contains the digital signature is issued by a certificate authority (CA) that associates a public key with an identity.
- Various schemes, such as RSA, Digital Signature Algorithm, and Elliptic Curve Digital Signature Algorithm-based digital signature schemes are used in practice. RSA is the most commonly used; however, with the traction of elliptic curve cryptography, ECDSA-based schemes are also becoming quite popular.

Elliptic Curve Digital Signature Algorithm (ECDSA)

Key Generation

Each signer must generate a pair of keys, one private and one public. The signer, let us call him Bob, generates the two keys using the following steps:

1. Select a random integer d, $d \in [1, n - 1]$
2. Compute $Q = dG$. This is a point in Eq(a, b)
3. Bob's public key is Q and private key is d.

Elliptic Curve Digital Signature Algorithm

Digital Signature Generation and Authentication

With the public domain parameters and a private key in hand, Bob generates a digital signature of 320 bytes for message m using the following steps:

- 1. Select a random or pseudorandom integer k, $k \in [1, n - 1]$
- 2. Compute point $P = (x, y) = kG$ and $r = x \bmod n$. If $r = 0$ then goto step 1
- 3. Compute $t = k^{-1} \bmod n$
- 4. Compute $e = H(m)$, where H is one of the SHA-2 or SHA-3 hash
- functions.
- 5. Compute $s = k^{-1}(e + dr) \bmod n$. If $s = O$ then then goto step 1
- 6. The signature of message m is the pair (r, s).

Elliptic Curve Digital Signature Algorithm

verifies the signature

Alice knows the public domain parameters and Bob's public key. Alice is presented with Bob's message and digital signature and verifies the signature using the following steps:

- 1. Verify that r and s are integers in the range 1 through n - 1
- 2. Using SHA, compute the 160-bit hash value e = H(m)
- 3. Compute w = s^{-1} mod n
- 4. Compute u1 = ew and u2 = rw
- 5. Compute the point X = (x1, y1) = u1G + u2Q
- 6. If X = 0, reject the signature else compute v = x1 mod n
- 7. Accept Bob's signature if and only if v = r

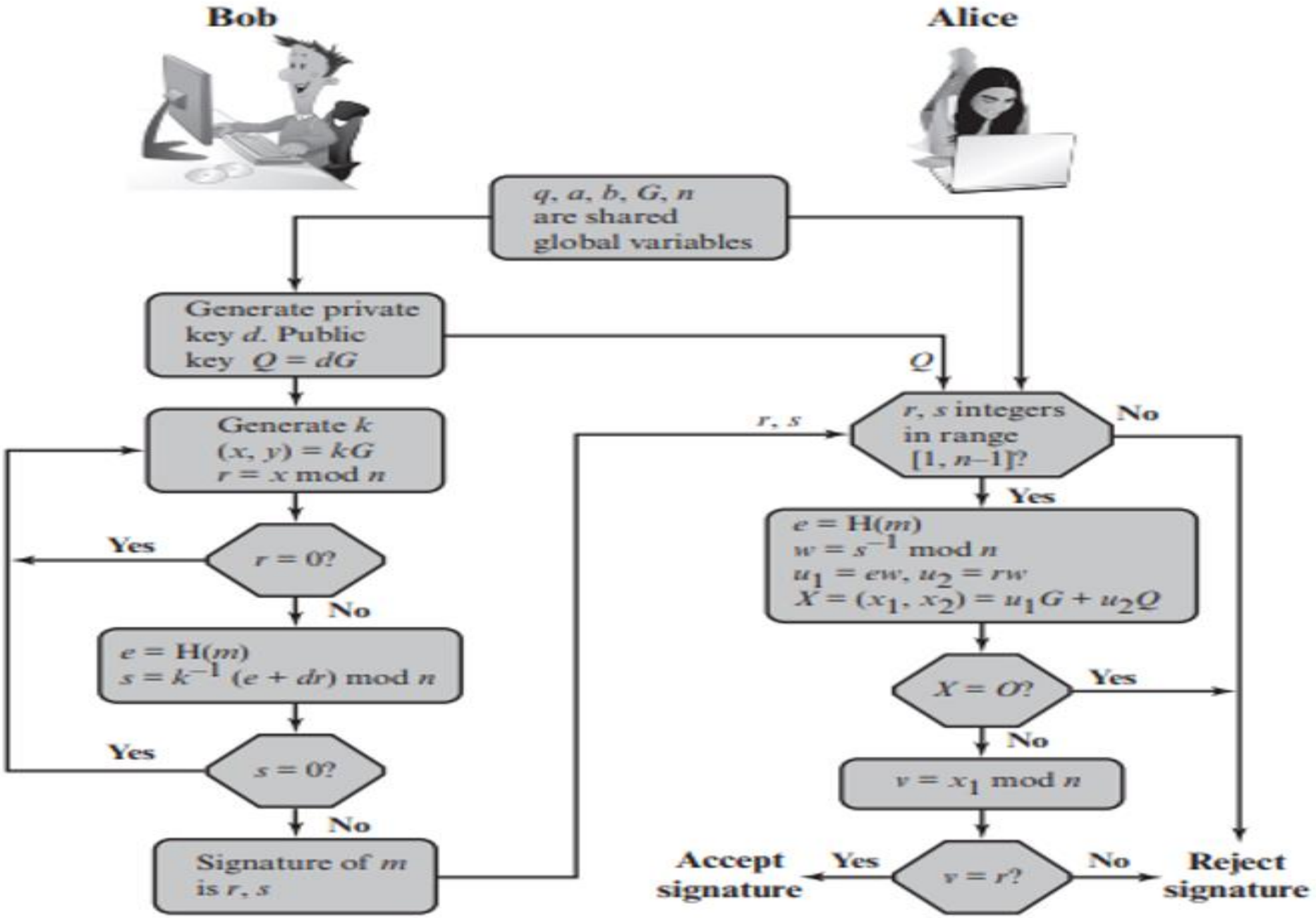


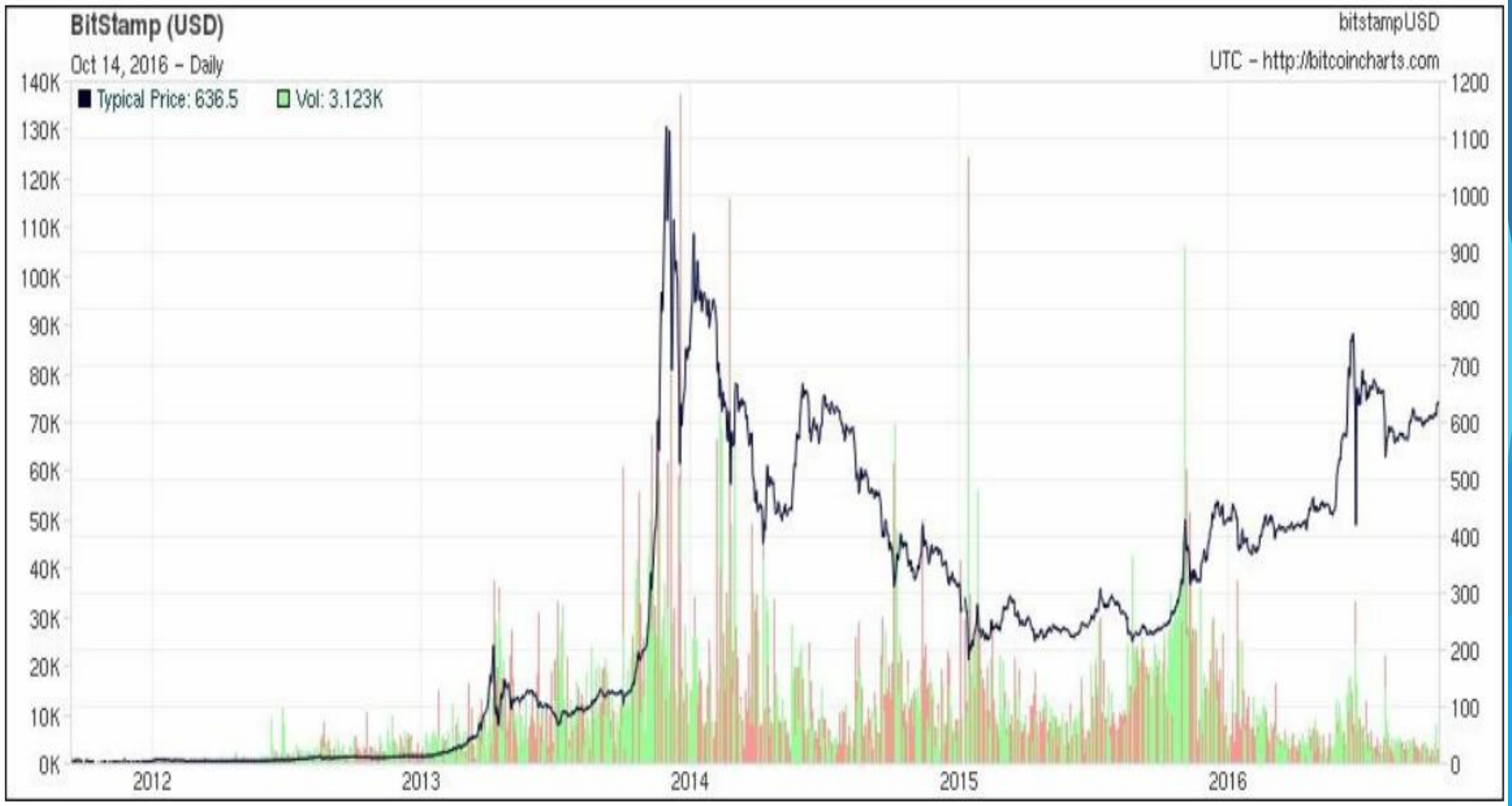
Figure 13.5 ECDSA Signing and Verifying

Bitcoin

- Bitcoin is the first application of the blockchain technology. In this chapter, readers will be introduced to bitcoin technology in detail.
- Bitcoin has started a revolution with the introduction of the very first fully decentralized digital currency, and one that has proven to be extremely secure and stable. This has also sparked a great interest in academic and industrial research and introduced many new research areas.
- Since its introduction in 2008, bitcoin has gained much popularity and is currently the most successful digital currency in the world with billions of dollars invested in it. It is built on decades of research in the field of cryptography, digital cash, and distributed computing. In the following section, a brief history is presented in order to provide the background required to understand the foundations behind the invention of bitcoin.

- Digital currencies have always been an active area of research for many decades. Early proposals to create digital cash go as far back as the early 1980s. In 1982, David Chaum proposed a scheme that used blind signatures to build untraceable digital currency.
- In this scheme, a bank would issue digital money by signing a blind and random serial number presented to it by the user. The user could then use the digital token signed by the bank as currency. The limitation in this scheme was that the bank had to keep track of all used serial numbers. This was a central system by design and required to be trusted by the users.
- Later on in 1990, David Chaum proposed a refined version named e-cash that not only used blinded signature, but also some private identification data to craft a message that was then sent to the bank. This scheme allowed the detection of double spending but did not prevent it. If the same token was used at two different locations, then the identity of the double spender would be revealed. e-cash could only represent a fixed amount of money.
- Adam Back's hashcash, introduced in 1997, was originally proposed to thwart e-mail spam. The idea behind hashcash was to solve a computational puzzle that was easy to verify but comparatively difficult to compute.

- In 2008, a paper on bitcoin, **Bitcoin**: A Peer-to-Peer Electronic Cash System was written by Satoshi Nakamoto.
- The first key idea introduced in the paper was that purely peer-to-peer electronic cash that does need an intermediary bank to transfer payments between peers.
- Bitcoin is built on decades of cryptographic research such as the research in Merkle trees, hash functions, public key cryptography, and digital signatures.
- Moreover, ideas such as BitGold, b-money, hashcash, and cryptographic time stamping provided the foundations for bitcoin invention. All these technologies are cleverly combined in bitcoin to create the world's first decentralized currency.
- The key issue that has been addressed in bitcoin is an elegant solution to the Byzantine Generals problem along with a practical solution of the double-spend problem. The value of bitcoin has increased significantly since 2011, as shown in the following graph:


BitStamp (USD)
bitstampUSD
Oct 14, 2016 – Daily
UTC – http://bitcoincharts.com
Typical Price: 636.5
Vol: 3.123K
140K
130K
120K
110K
100K
90K
80K
70K
60K
50K
40K
30K
20K
10K
0K
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
2012
2013
2014
2015
2016

- Bitcoin price and volume since 2012 (on logarithmic scale) The regulation of bitcoin is a controversial subject and as much as it is a libertarian's dream, law enforcement agencies and governments are proposing various regulations to control it, such as BitLicense issued by New York's state department of financial services.
- This is a license issued to businesses that perform activities related to virtual currencies. The growth of Bitcoin is also due to so-called Network Effect. Also called demand-side economies of scale, it is a concept that basically means more users who use the network, the more valuable it becomes.
- Over time, an exponential increase has been seen in the Bitcoin network growth. Even though the price of bitcoin is quite volatile, it has increased significantly over the last few years. Currently (at the time of writing this), bitcoin price is 29,098USD

- Bitcoin definition Bitcoin can be defined in various ways; it's a protocol, a digital currency, and a platform. It is a combination of peer-to-peer network, protocols, and software that facilitate the creation and usage of the digital currency named bitcoin.
- Note that Bitcoin with a capital B is used to refer to the Bitcoin protocol, whereas bitcoin with a lowercase b is used to refer to bitcoin, the currency. Nodes in this peer-to-peer network talk to each other using the Bitcoin protocol.
- Decentralization of currency was made possible for the first time with the invention of bitcoin. Moreover, the double spending problem was solved in an elegant and ingenious way in bitcoin. Double spending problem arises when, for example, a user sends coins to two different users at the same time and they are verified independently as valid transactions.

- Keys and addresses Elliptic curve cryptography is used to generate public and private key pairs in the Bitcoin network. The bitcoin address is created by taking the corresponding public key of a private key and hashing it twice, first with the SHA256 algorithm and then with RIPEMD160.
- The resultant 160-bit hash is then prefixed with a version number and finally encoded with a Base58Check encoding scheme. The bitcoin addresses are 26-35 characters long and begin with digit 1 or 3. A typical bitcoin address looks like a string shown here:

1ANAguGG8bikEv2fYsTBnRUmx7QUcK58wt

Transactions : Transactions are at the core of the bitcoin ecosystem. Transactions can be as simple as just sending some bitcoins to a bitcoin address, or it can be quite complex depending on the requirements. Each transaction is composed of at least one input and output.

- Inputs can be thought of as coins being spent that have been created in a previous transaction and outputs as coins being created. If a transaction is minting new coins, then there is no input and therefore no signature is needed.
- If a transaction is to sends coins to some other user (a bitcoin address), then it needs to be signed by the sender with their private key and a reference is also required to the previous transaction in order to show the origin of the coins.
- Coins are, in fact, unspent transaction outputs represented in Satoshis. Transactions are not encrypted and are publicly visible in the blockchain. Blocks are made up of transactions and these can be viewed using any online blockchain explorer.

The transaction life cycle

- 1. A user/sender sends a transaction using wallet software or some other interface.
- 2. The wallet software signs the transaction using the sender's private key.
- 3. The transaction is broadcasted to the Bitcoin network using a flooding algorithm.
- 4. Mining nodes include this transaction in the next block to be mined.
- 5. Mining starts once a miner who solves the Proof of Work problem broadcasts the newly mined block to the network. Proof of Work is explained in detail later in this chapter.
- 6. The nodes verify the block and propagate the block further, and confirmation starts to generate.
- 7. Finally, the confirmations start to appear in the receiver's wallet and after approximately six confirmations, the transaction is considered finalized and confirmed. However, six is just a recommended number; the transaction can be considered final even after the first confirmation. The key idea behind waiting for six confirmations is that the probability of double spending is virtually eliminated after six confirmations.

The transaction structure

- A transaction at a high level contains metadata, inputs, and outputs. Transactions are combined to create a block. The transaction structure is shown in the following table:

Field	Size	Description
Version Number	4 bytes	Used to specify rules to be used by the miners and nodes for transaction processing.
Input counter	1 bytes - 9 bytes	The number of inputs included in the transaction.
list of inputs	variable	Each input is composed of several fields, including Previous transaction hash, Previous Txout-index, Txin-script length, Txin-script, and optional sequence number. The first transaction in a block is also called a coinbase transaction. It specifies one or more transaction inputs.
Out-counter	1 bytes - 9 bytes	A positive integer representing the number of outputs.
list of outputs	variable	Outputs included in the transaction.
lock_time	4 bytes	This defines the earliest time when a transaction becomes valid. It is either a Unix timestamp or a block number.

- **MetaData**: This part of the transaction contains some values such as the size of the transaction, the number of inputs and outputs, the hash of the transaction, and a lock_time field. Every transaction has a prefix specifying the version number.
- **Inputs**: Generally, each input spends a previous output. Each output is considered an Unspent Transaction Output (UTXO) until an input consumes it.
- **Outputs**: Outputs have only two fields, and they contain instructions for the sending of bitcoins. The first field contains the amount of Satoshis, whereas the second field is a locking script that contains the conditions that need to be met in order for the output to be spent. More information on transaction spending using locking and unlocking scripts and producing outputs is discussed later in this section.
- **Verification**: Verification is performed using bitcoin's scripting language.

Types of transaction

There are various scripts available in bitcoin to handle the value transfer from the source to the destination. These scripts range from very simple to quite complex depending upon the requirements of the transaction. Standard transaction types are discussed here. Standard transactions are evaluated using IsStandard() and IsStandardTx() tests and only standard transactions that pass the test are generally allowed to be mined or broadcasted on the bitcoin network. However, nonstandard transactions are valid and allowed on the network.

- **Pay to Public Key Hash (P2PKH)**: P2PKH is the most commonly used transaction type and is used to send transactions to the bitcoin addresses.
- In a simplified manner, "Pay to Public Key Hash" (P2PKH) is a method used in Bitcoin transactions to send bitcoins to someone. Instead of directly using someone's public key, which is a long string of numbers and letters, P2PKH uses a shorter and more secure version called a hash of the public key. This adds a layer of protection and privacy to the transaction.

- **Pay to Script Hash (P2SH):** "Pay to Script Hash" (P2SH) is another method used in Bitcoin transactions to send bitcoins. It allows more advanced and flexible transaction conditions by using a script (a set of instructions) instead of a simple public key or public key hash. P2SH is commonly used for features like multi-signature transactions, time-locked transactions, and other smart contract capabilities.
- **MultiSig (Pay to MultiSig):**"Pay to Multi-Sig" (P2MS) is a feature in Bitcoin transactions that allows funds to be sent to an address requiring multiple digital signatures to spend the bitcoins. It is commonly used to enhance security and create shared wallets that require approval from multiple parties before any transaction is executed.
- **Null data/OP_RETURN**: In Bitcoin transactions, the "Null data" or "OP_RETURN" is an opcode that allows users to embed small pieces of data directly into the blockchain without transferring any bitcoins. It serves various purposes, such as storing metadata, timestamps, or other non-financial information on the Bitcoin blockchain.

UTXO

- **Unspent Transaction Output (UTXO)** is an unspent transaction output that can be spent as an input to a new transaction. Other concepts related to transactions in bitcoin are described below.

Transaction fee : Transaction fees are charged by the miners. The fee charged is dependent upon the size of the transaction. Transaction fees are calculated by subtracting the sum of the inputs and the sum of the outputs.

- The fees are used as an incentive for miners to encourage them to include a user transaction in the block the miners are creating.
- All transactions end up in the memory pool, from where miners pick up transactions based on their priority to include them in the proposed block.
- The calculation of priority is introduced later in this chapter; however, from a transaction fee point of view, a transaction with a higher fee will be picked up sooner by the miners.
- There are different rules based on which fee is calculated for various types of actions, such as sending transactions, inclusion in blocks, and relaying by nodes(They perform duplication checks, validate signatures and other steps, and than transmit only valid messages). Fees are not fixed by the Bitcoin protocol and are not mandatory; even a transaction with no fee will be processed in due course but may take a very long time.

- **Contracts** :As defined in the bitcoin core developer guide, contracts are basically transactions that use the bitcoin system to enforce a financial agreement.(lightning layer)
- This is a simple definition but has far-reaching consequences as it allows users to design complex contracts that can be used in many real-world scenarios.
- Contracts allow the development of a completely decentralized, independent, and reduced risk platform. Various contracts, such as escrow, arbitration, and micropayment channels, can be built using the bitcoin scripting language.
- The current implementation of a script is very limited, but various types of contracts are still possible to develop. For example, the release of funds only if multiple parties sign the transaction or perhaps the release of funds only after a certain time has elapsed. Both of these scenarios can be realized using multiSig and transaction lock time options.

- **Transaction malleability :**Transaction malleability in bitcoin was introduced due to a bug in the bitcoin implementation. Due to this bug, it becomes possible for an adversary to change the Transaction ID of a transaction, thus resulting in a scenario where it would appear that a certain transaction has not been executed. This can allow scenarios where double deposits or withdrawals can occur. In other words, this bug allows the changing of the unique ID of a bitcoin transaction before it is confirmed. If the ID is changed before confirmation, it would seem that the transaction did not happen at all, which can then allow double deposits or withdrawal attacks.
- **Transaction pools :**Also known as memory pools, these pools are basically created in local memory by nodes in order to maintain a temporary list of transactions that are not yet confirmed in a block. Transactions are included in a block after passing verification and based on their priority.

- **Transaction verification :**This verification process is performed by bitcoin nodes. The following is described in the bitcoin developer guide:
- 1. Check the syntax and ensure that the syntax of the transaction is correct.
- 2. Verify that inputs and outputs are not empty.
- 3. Check whether the size in bytes is less than the maximum block size, which is 1 MB currently.
- 4. The output value must be in the allowed money range (0 to 21 million BTC).
- 5. All inputs must have a specified previous output, except for coinbase transactions, which should not be relayed.
- 6. Verify that nLockTime must not exceed 31-bits. For a transaction to be valid, it should not be less than 100 bytes. Also, the number of signature operands in a standard signature should be less than or not more than 2.

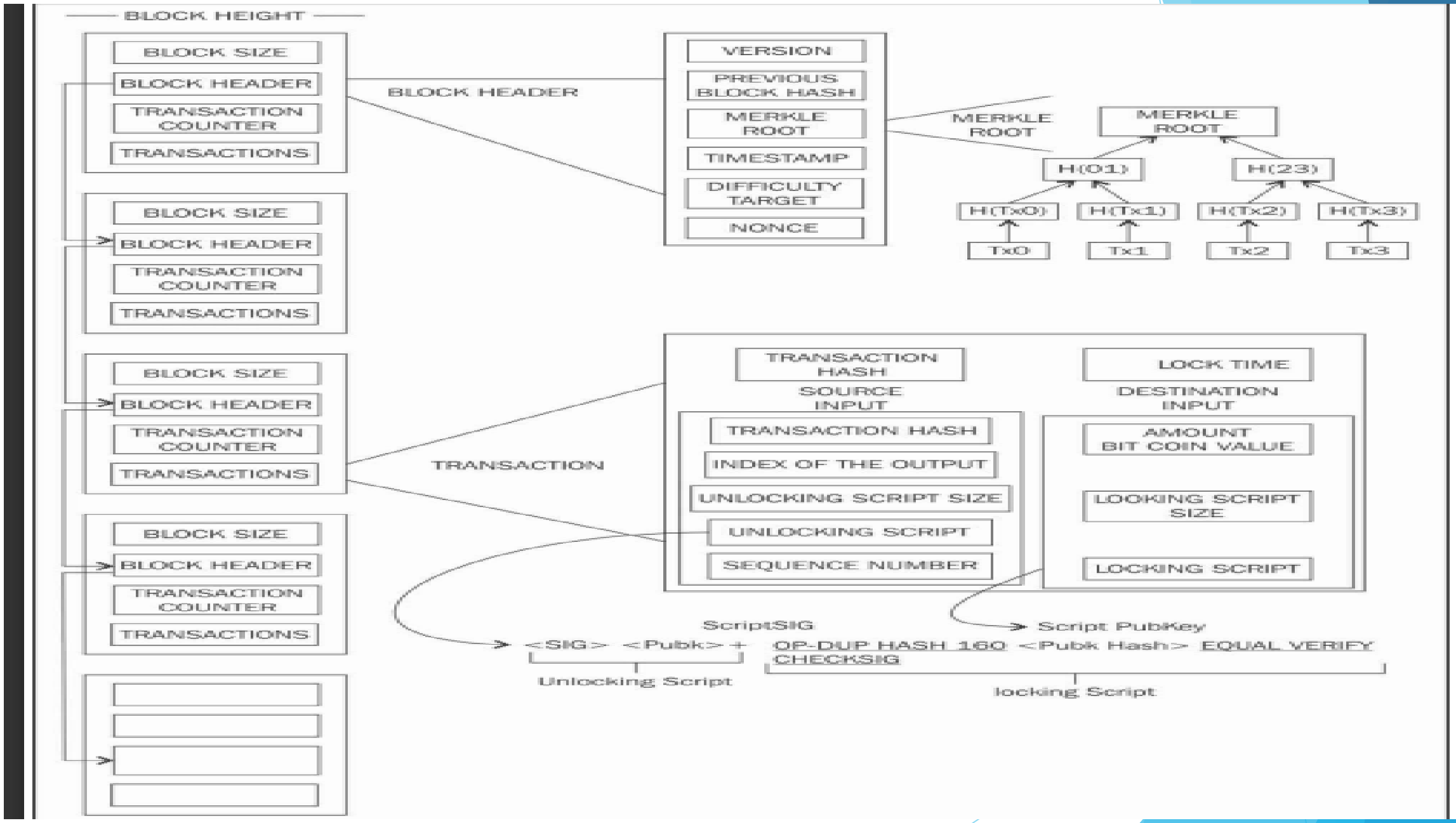
- 7. A transaction is rejected if there is already a matching transaction in the pool or in a block in the main branch.
- 8. The transaction will be rejected if the referenced output for each input exists in any other transaction in the pool.
- 9. For each input, there must exist a referenced output transaction. This is searched in the main branch and the transaction pool to find whether the output transaction is missing for any input, and this will be considered an orphan transaction. It will be added to the orphan transactions pool if a matching transaction is not in the pool already.
- 10. For each input, if the referenced output transaction is the coinbase, it must have at least 100 confirmations; otherwise, the transaction will be rejected.
- 11. For each input, if the referenced output does not exist or has been spent already, the transaction will be rejected.
- 12. Using the referenced output transactions to get input values, verify that each input value, as well as the sum, is in the allowed range of 0-21 million BTC.
- 13. Reject the transaction if the sum of input values is less than the sum of output values.
- 14. Reject the transaction if the transaction fee would be too low to get into an empty block

The structure of a block

Bytes	Name	Description
80	Block header	This includes fields from the block header described in the next section.
variable	Transaction counter	The field contains the total number of transactions in the block, including the coinbase transaction.
variable	Transactions	All transactions in the block.

The structure of a block header

Bytes	Name	Description
4	Version	The block version number that dictates the block validation rules to follow.
32	previous block header hash	This is a double SHA256 hash of the previous block's header.
32	merkle root hash	This is a double SHA256 hash of the merkle tree of all transactions included in the block.
4	Timestamp	This field contains the approximate creation time of the block in the Unix epoch time format. More precisely, this is the time when the miner has started hashing the header (the time from the miner's point of view).
4	Difficulty target	This is the difficulty target of the block.
4	Nonce	This is an arbitrary number that miners change repeatedly in order to produce a hash that fulfills the difficulty target threshold.


BLOCK HEIGHT
BLOCK SIZE
BLOCK HEADER
TRANSACTION COUNTER
TRANSACTIONS
BLOCK HEADER
VERSION
PREVIOUS BLOCK HASH
MERKLE ROOT
TIMESTAMP
DIFFICULTY TARGET
NONCE
MERKLE ROOT
MERKLE ROOT
H(01)
H(23)
H(Tx0)
H(Tx1)
H(Tx2)
H(Tx3)
Tx0
Tx1
Tx2
Tx3
TRANSACTION
TRANSACTION HASH
SOURCE INPUT
LOCK TIME
DESTINATION INPUT
TRANSACTION HASH
INDEX OF THE OUTPUT
UNLOCKING SCRIPT SIZE
UNLOCKING SCRIPT
SEQUENCE NUMBER
AMOUNT BIT COIN VALUE
LOCKING SCRIPT SIZE
LOCKING SCRIPT
ScriptSIG
Script PubKey
<SIG> <Pubk> +
OP-DUP HASH 160 <Pubk Hash> EQUAL VERIFY CHECKSIG
Unlocking Script
locking Script

The genesis block

- This is the first block in the bitcoin blockchain. The genesis block was hardcoded in the bitcoin core software. It is in the chainparams.cpp file.

https://github.com/bitcoin/bitcoin/blob/master/src/chainparams.cpp

```
48   *     CTxOut(nValue=50.00000000, scriptPubKey=0x5F1DF16B2B704C8A578D0B)
49   *   vMerkleTree: 4a5e1e
50   */
51  static CBlock CreateGenesisBlock(uint32_t nTime, uint32_t nNonce, uint32_t nBits, int32_t nVersion, const CAmount& genesisReward)
52  {
53      const char* pszTimestamp = "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks";
54      const CScript genesisOutputScript = CScript() << ParseHex("04678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb649
55      return CreateGenesisBlock(pszTimestamp, genesisOutputScript, nTime, nNonce, nBits, nVersion, genesisReward);
56  }
57
58  /**
```

- Bitcoin provides protection against double spending by enforcing strict rules on transaction verification and via mining. Blocks are added in the blockchain only after strict rule checking and successful Proof of Work solution. Block height is the number of blocks before a particular block in the blockchain. The current height (at the time of writing this) of the blockchain is 800238 blocks. Proof of Work is used to secure the blockchain.
- Each block contains one or more transactions, out of which the first transaction is a coinbase transaction.
- Orphan blocks are also called detached blocks and were accepted at one point in time by the network as valid blocks but were rejected when a proven longer chain was created that did not include this initially accepted block. They are not part of the main chain and can occur at times when two miners manage to produce the blocks at the same time.

- If an adversary manages to gain 51% control of the network hash rate (computational power), then they can impose their own version of transaction history. Forks in blockchain can occur with the introduction of changes in the Bitcoin protocol. In case of soft fork, only previous valid blocks are no longer acceptable, thus making soft fork backward compatible.
- In case of soft fork, only miners are required to upgrade to the new client software in order to make use of the new protocol rules. Planned upgrades do not necessarily create forks because all users should have updated already. A hard fork, on the other hand, invalidates previously valid blocks and requires all users to upgrade. New transaction types are sometimes added as a soft fork, and any changes such as block structure change or major protocol changes results in hard fork.